A computational framework for telemedicine

Gregor von Laszewski, Ian Foster, George K. Thiruvathukal, Brian Toonen
Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue

Abstract

Emerging telemedicine applications require the ability to exploit diverse, geographically distributed resources. These applications use high-speed networks to integrate supercomputers, large databases, archival storage devices, advanced visualization devices, and/or sophisticated instruments. This form of *networked virtual supercomputers* is also known as *metacomputers* and is beeing used by many other scientific applications areas. In this article, we analyze requirements necessary for a telemedicine computing infrastructure and compare them with requirements found in a typical metacomputing environment. We will show that metacomputing environments can be used to enable a more powerful and unified computational infrastructure for telemedicine. The Globus metacomputing environment can provide the necessary basis to enable a large scale telemdical infrastructure. Globus is designed in a modular fashion and can be extended to support the specific requirements for telemedicine.

Keywords: Globus, telemedicine, and metacomputing.

Contents

1 Introduction **3**

2 Telemedicine **4**

3 Infrastructure requirements for telemedicine **7**

4 Metacomputing Environments **8**

5 Metacomputing **8**
5.1 Metacomputing Applications 11
5.2 Metasystem Characteristics 11

6 The Globus Metacomputing Infrastructure Toolkit **12**
6.1 The Globus Toolkit 13
6.2 Support for Resource-Aware Services and Applications 14

7 Globus Toolkit Components **15**
7.1 Communications 15
7.2 Metacomputing Directory Service 17
7.3 Authentication Methods 17
7.4 Data Access Services 18

8 Higher-Level Services **18**
8.1 Parallel Programming Interfaces 20
8.2 Unified Certificate-based Authentication 20

9 Globus Testbeds and Experiences **21**

10 Requirements **22**

11 Future opertunities **22**

12 Conclusion **22**

13 Acknowledgements **22**

A computational framework for telemedicine

Gregor von Laszewski, Ian Foster, George K. Thiruvathukal, Brian Toonen

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue

Abstract

Emerging telemedicine applications require the ability to exploit diverse, geographically distributed resources. These applications use high-speed networks to integrate supercomputers, large databases, archival storage devices, advanced visualization devices, and/or sophisticated instruments. This form of *networked virtual supercomputers* is also known as *metacomputers* and is beeing used by many other scientific applications areas. In this article, we analyze requirements necessary for a telemedicine computing infrastructure and compare them with requirements found in a typical metacomputing environment. We will show that metacomputing environments can be used to enable a more powerful and unified computational infrastructure for telemedicine. The Globus metacomputing environment can provide the necessary basis to enable a large scale telemdical infrastructure. Globus is designed in a modular fashion and can be extended to support the specific requirements for telemedicine.

Keywords: Globus, telemedicine, and metacomputing.

1 Introduction

In the last decade, dramatic advances in software and hardware technology have changed the landscape for computing. Today, personal computers have reached a performance level which one could have only dreamed of, a couple of years ago. The increase in processor speed is accompanied by the availability of memory to economical prizes. New graphic cards let a PC perform at almost workstation speed. On the high end, vector-supercomputers are outperformed by distributed memory parallel architectures. Furthermore, changes in software engineering are marked by the acceptance of object oriented programming concepts and languages.

Computer networks have evolved from local area networks, over medium area networks, to wide area networks. The new hardware enables the distribution of information in a global "World Wide Web". The World Wide Web has established itself as a functioning computing environment, accessible by the ever-increasing number of online users. Starting from the desire to provide a mechanism for exchanging data between scientists, it has reached the potential to become the computing platform of the future. Hardware advances in the network technology, like the introduction of the ATM technology (Asynchronous Transfer Mode), provide the necessary backbone for the fast information exchange between computers.

Currently, the WWW is most frequently used for exchanging data, and allowing online users to access information stored at remote sites. Besides redistributing information, the WWW can be used to redistribute computations on different compute

servers. Computations can be mapped to idle compute servers or unique compute resources which otherwise would not be available. An environment managing the resources of many cooperating computers is known as metacomputing environmnet. Without doubt, a functioning metacomputer, controlled by such an environment, will influence not only developments in telemedicine but also many other research fields. Many of the computing components, which have been originally developed for the scientific research community, can be reused for a computational infrastructure in telemedicine and tele-informatics.

One of the goals of this paper it to examine the possibility to reuse existing metacomputing technologies for telemedicine. First, we will define the term telemedicine and derive necessary requirements that influence the design of a compute infrastructure. We show that this infrastructure is similar to existing metacomputing testbeds as used in the Globus metacomputing framework. We analyze the Globus metacomputing framework and indicate where it can help in order to support telemedical applications.

2 Telemedicine

Telemedicine is an emerging discipline, which utilizes the newest technologies to transmit medical information with the help of electric signals. From this broad definition, it is clear that the applications in telemedicine are of a wide spectrum. Telemedicine is used in the following areas:

- Tele consulting and assistance
  - Remote supervision
  - Remote consulting
  - Medical Video Conferencing
- Virtual medical libraries
  - Document distribution via WWW
  - Medical databases
  - Research databases
- Virtual medical stores
  - Accounting
  - Product databases
  - Service Brokers

This list is not comprehensive but shows examples where telemedicine is used and will be used in future. Today telemedicine is in wide practical use. Tele consulting between doctors in geographically disperse locations is daily done by e.g., video, e-mail, and telephone. Video conferencing is used for diagnosing, as well as, educational purposes. Material containing medical information is nowadays often distributed via WWW. The functionality of the WWW in the area of telemedicine reaches from buying medical products to a visit of a virtual telemedical office. Telemedicine is successfully used to reach underserved areas. Some of the leading goals of telemedicine are to enable

- an increase of the availability of services,

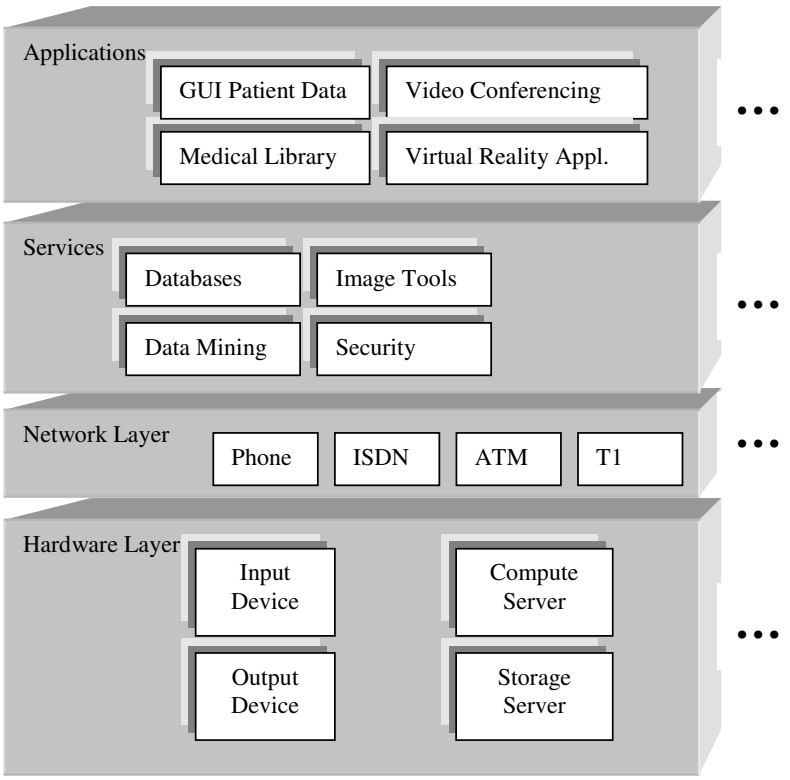

Figure 1: Different Layers of a telemedicine environment

- instant access to data,
- secure access and exchange of data,
- user friendly access,
- a high quality of service, and
- to reduce the cost for the health service.

infrastructure.

Figure 10 depicts a layered abstraction of a telemedicine infrastructure. The lowest level includes the available hardware components for input, output, storage and processing of medical information. Compute servers range from PCs to workstations and graphic engines, as well as, supercomputers. Input devices can be as simple as PCs for the input of patient data records. More complex and expensive devices like X-ray and computer tomography devices are also available in a telemedical infrastructure. This can go as far as to include synchrotrons for bio-medical research[?]. The huge amount of data has to be stored on large storage devices. Output devices are also of a wide variety, ranging from specialized printers and monitors to virtual reality CAVES. The different devices are connected via a network using different technology

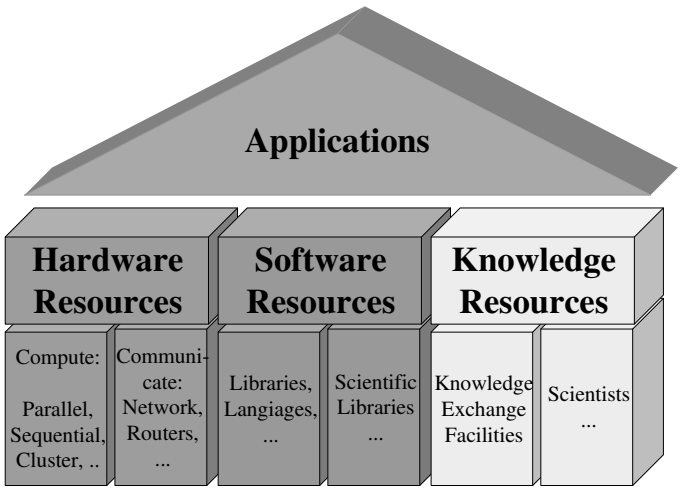

Figure 2: Resources needed to implement applications in a metacomputing environment

dependent on the service that should be provided. The network layer and the hardware layer are accessed via services. This enables a necessary abstraction form the underlying hardware in order to achieve interoperability between different compute platforms and communication protocols. An example for such a high level service are database services and their application interfaces that allows access to a possible distributed database. The highest level of abstraction is defined by the application layer. For many users of telemedicine it is not important to know the details of the compute and network environment. Thus, many details can be hidden from the end user of a telemedicine environment. Usually the user will communicate via a sophisticated and user specific graphical user interface with the application or service she is interested in. This means that the internal structure of a telemedicine environment can be hidden from the end user.

From the computational point of view it is important to analyse the needs necessary to enable:

- medical information gathering/input,
- medical information storage,
- medical information processing,
- medical information exchange, and
- medical information visualization/output.

To enable this kind of information processing one needs the following . (Figure 10).

Table 1: Storage requirements for imaging techniques and transmission times while using different network transport medias/services

pixels hor vert	image depth	average number of images/exam	calc memory	average storage requirement	Kbits 28.8	Kbits 128	Mbits 1.54	Mbits 45.3	mbits 1.5	mbit 155
512	12	30	12	15	69.4	15.6	9.7	0.3	10.0	0.10
256	12	50	3	6.5	30.1	6.8	4.2	0.1	4.3	0.04
1000	8	20	45	20	92.6	20.8	13.0	0.4	13.3	0.13
1000	8	15	45	15	69.4	15.6	9.7	0.3	10.0	0.10
512	6	36	12	9	41.7	9.4	5.8	0.2	6.0	0.06
128	8	26	1	0.4	1.9	0.4	0.3	0.01	0.3	0.00
2000	10	4	180	32	148.1	33.3	20.8	0.7	21.3	0.21
4000	12	4	720	128	592.6	133.3	83.1	2.8	85.3	0.83
					in min	in min	in sec	in sec	in sec	in se

3 Infrastructure requirements for telemedicine

Information exchange. The success of telemedecine is largely dependent on a broad network infrastructure. The network technologies used in a telemedical environment include asynchronous transfer mode (ATM), satellites, asymmetrical digital subscriber line (ADSL), and cable modems. Lower bandwidth technologies are standard analog phone lines, and digital networks with integrated services digital network (ISDN). The proper selection of a communication service is also determined by the type(s) of information, the amount of information, and the urgency of the information exchange which is illustrated in Figure 1. The choice is determined by a cost versus benefit factor.

Information processing, vizualization and output. The computational power needed for some telemedical applications is large. An example is the computational power of a single workstation is today still not able to render a large quantity of the computational power of supercomputers makes it possible to enable real-time steering during examinations (Figure 3).

Other examples are can be found in the X-ray analysis of large molecular structures to improve the knoledge to find medical treatments. Image detection algorithms, which build only the first step in a series of calculations, take today on an Origin 2000 a day to complete[?].

Another application that is of great impact in the health industrie is the analysis of gene sequences. A small comparison of about 9 genome sequenzes of will result in a total of 16.9 million comparisons. Each of the comparisons will take a considerable time to finish (under 1 minute dependent on the processor used). Such a calculataion would create about 20Gbyte of data to be stored during the calculation[?].

Clearly utilizing the computational power of a metacomputing environment will help twofold. First, the turnaroud time of a calculation is smaller. Thus, experiments can be analysed more quickly. Second, due to the available compute power bigger problems can be solved in the same amount of time.

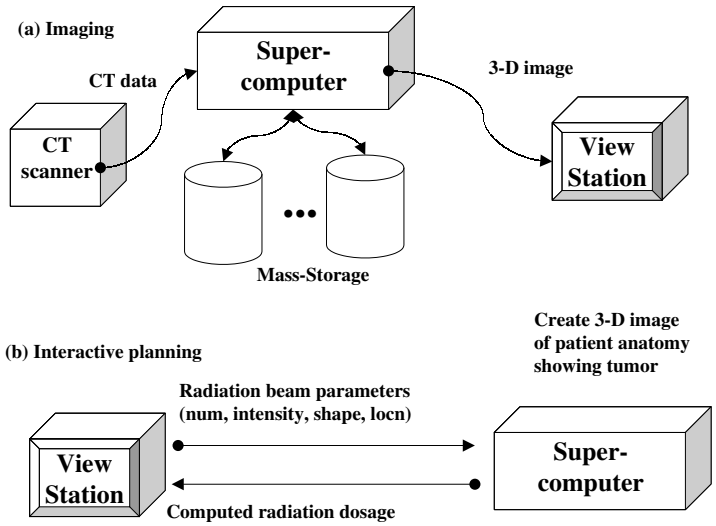

Figure 3: Usage of supercomputers in imaging and computational steering

4 Metacomputing Environments

Telemedicine requires unique capabilities that are not available in a single computer. Applications like telemedicine motivate the construction of *networked virtual supercomputers*, or *metacomputers* [2], execution environments in which high-speed networks are used to connect supercomputers, databases, scientific instruments, and advanced display devices, perhaps located at geographically distributed sites. In principle, networked virtual supercomputers can both increase accessibility to supercomputing capabilities and enable the assembly of unique capabilities that could not otherwise be created in a cost-effective manner.

5 Metacomputing

We use the term *metacomputer* to denote a networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high-speed networks. Figure 4 illustrates an example of such a system.

Metacomputing, like more mainstream applications of distributed computing, is motivated by a need to access resources not located within a single computer system. Frequently, the driving force is economic: the resources in question—for example, supercomputers—are too expensive to be replicated. Alternatively, an application may require resources that would not normally be co-located, because a particular configuration is required only rarely: for example, a collaborative telemedical environment that connects several virtual reality systems, design databases, and supercomputers required to work on a particular problem, like large medical image rendering. Finally,

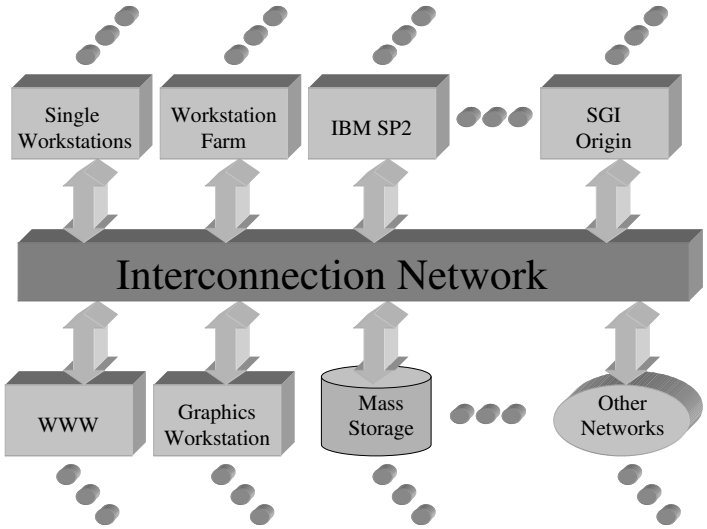

Figure 4: A metacomputer utilizing different compute resources

certain unique resources—such as specialized databases and people—cannot be replicated. In each case, the ability to construct networked virtual supercomputers can provide qualitatively new capabilities that enable new approaches to problem solving.

Experience with high-speed networking testbeds has demonstrated convincingly that there are indeed applications of considerable scientific and economic importance that can benefit from metacomputing capabilities. For example, the I-WAY networking experiment, which connected supercomputers and other resources at 17 different sites across North America, saw 60 groups develop applications in areas as diverse as large-scale scientific simulation [22, 23], collaborative engineering [4, 5], and supercomputer-enhanced scientific instruments [16].

Metacomputers have much in common with both distributed and parallel systems, yet also differ from these two architectures in important ways. Like a distributed system, a networked supercomputer must integrate resources of widely varying capabilities, connected by potentially unreliable networks and often located in different administrative domains. However, the need for high performance can require programming models and interfaces radically different from those used in distributed systems. As in parallel computing, metacomputing applications often need to schedule communications carefully to meet performance requirements. However, the heterogeneous and dynamic nature of metacomputing systems limits the applicability of current parallel computing tools and techniques.

These considerations suggest that while metacomputing can build on distributed and parallel software technologies, it also requires significant advances in mechanisms, techniques, and tools. The Globus project is intended to accelerate these advances. In a first phase, we are developing and deploying a *metacomputing infrastructure toolkit*

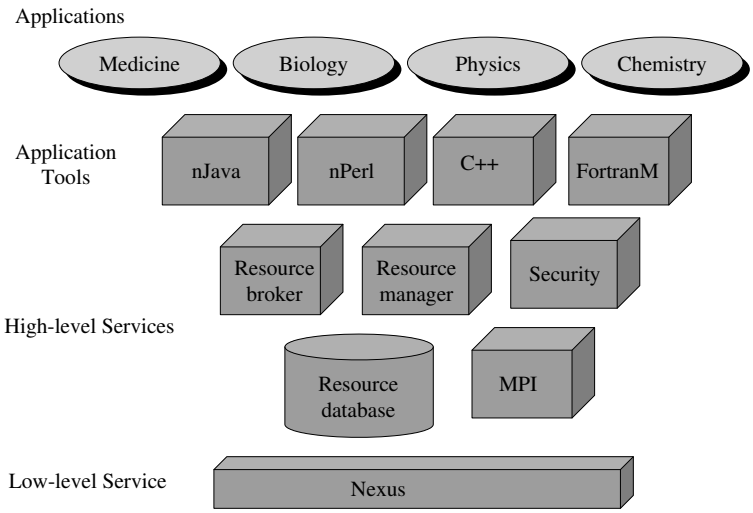

Figure 5: Components of the Globus toolkit

providing basic capabilities and interfaces in areas such as communication, information, resource location, resource scheduling, authentication, and data access. Together, these toolkit components define a *metacomputing abstract machine* on which can be constructed a range of alternative infrastructures, services, and applications (Figure ??). We ourselves are building parallel programming tools and resource discovery and scheduling services, and other groups are working in other areas.

Our long-term goal in the Globus project is to address the problems of configuration and performance optimization in metacomputing environments. These are challenging issues, because of the inherent complexity of metacomputing systems, the fact that resources are often only identified at runtime, and the dynamic nature of resource characteristics. We believe that successful applications must be able to configure themselves to fit the execution environment delivered by the metacomputing system, and then adapt their behavior to subsequent changes in resource characteristics. We are investigating the design of higher-level services layered on the Globus toolkit that enable the construction of such adaptive applications. We refer collectively to these services as forming an Adaptive Wide Area Resource Environment, or AWARE.

The rest of the article is as follows. In Section 5, we introduce general characteristics of metacomputing systems and requirements for metacomputing infrastructure. In Section ??, we describe the Globus architecture and the techniques used to support resource-aware applications. In Section 7, we describe major toolkit components. In Sections 8 and 9, we describe higher-level services constructed with the toolkit and testbeds that have deployed toolkit components. We conclude in Section 12 with a discussion of current system status and future plans.

5.1 Metacomputing Applications

Scientists and engineers are just beginning to explore the new applications enabled by networked supercomputing classes.

1. *Desktop supercomputing.* These applications couple high-end graphics capabilities with remote supercomputers and/or databases. This coupling connects users more tightly with computing capabilities, while at the same time achieving distance independence between resources, developers, and users. Examples are high quality rendering of medical images.
2. *Smart instruments.* These applications connect users to instruments such as microscopes, MRIs ,or satellite downlinks that are themselves coupled with remote supercomputers. This computational enhancement can enable both quasi-realtime processing of instrument output and interactive steering.
3. *Collaborative environments.* A third set of applications couple multiple virtual environments so that users at different locations can interact with each other and with supercomputer simulations [4, 5].
4. *Distributed supercomputing.* These applications couple multiple computers to tackle problems that are too large for a single computer or that can benefit from executing different problem components on different computer architectures [20, 22, 23]. We can distinguish scheduled and unscheduled modes of operation. In *scheduled* mode, resources, once acquired, are dedicated to an application. In *unscheduled* mode, applications use otherwise idle resources that may be reclaimed if needed; Condor [18] is one system that supports this mode of operation. In general, scheduled mode is required for tightly coupled simulations, particularly those with time constraints, while unscheduled mode is appropriate for loosely coupled applications that can adapt to time-varying resources. A possible example to use such an environment is the distributed solution of gene sequencing problems which will have increased importance in the near future[?].

5.2 Metasystem Characteristics

- *Scale and the need for selection.* Due to the extreme distributed nature of telemedical applications, it is important to implement a telemedical infrastructur ein a large scale. Resource selection must be provided to allow the access of a subset of resources.
- *Heterogeneity at multiple levels.* Both the computing resources used to construct virtual supercomputers and the networks that connect these resources are often highly heterogeneous. Heterogeneity arises at multiple levels, ranging from physical devices, through system software, to scheduling and usage policies.
- *Unpredictable structure.* Traditionally, high-performance applications have been developed for a single class of system with well- known characteristics—or even for one particular computer. In contrast, metacomputing applications like telemedicine, require to execute in a wide range of environments, constructed dynamically from available resources. Geographical distribution and complexity are other factors that make it difficult to determine system characteristics such as network bandwidth and latency *a priori*.

- *Dynamic and unpredictable behavior.* Traditional high- performance systems use scheduling disciplines such as space sharing or gang-scheduling to provide exclusive—and hence predictable— access to processors and networks. In metacomputing environments, resources—especially networks—are more likely to be shared. One consequence of sharing is that behavior and performance can vary over time. For example, in wide area networks built using the Internet Protocol suite, network characteristics such as latency, bandwidth and jitter may vary as traffic is rerouted. Large-scale metasystems may also suffer from network and resource failures. In general, it is not possible to guarantee even minimum quality of service requirements.
- *Multiple administrative domains.* The resources used by metacomputing applications are not owned or administered by a single entity. The need to deal with multiple administrative entities complicates the already challenging network security problem, as different entities may use different authentication mechanisms, authorization schemes, and access policies. The need to execute user- supplied code at different sites introduces additional concerns.

Fundamental to all of these issues is the need for mechanisms that allow applications to obtain real-time information about system structure and state, use that information to make configuration decisions, and be notified when information changes. Required information can include network activity, available network interfaces, processor characteristics, and authentication mechanisms. Decision processes can require complex combinations of these data in order to achieve efficient end-to- end configuration of complex networked systems.

6 The Globus Metacomputing Infrastructure Toolkit

A number of pioneering efforts have produced useful services for the metacomputing application developer. For example, Parallel Virtual Machine (PVM) [12] and the Message Passing Interface (MPI) [15] provide a machine-independent communication layer, Condor [18] provides a uniform view of processor resources, Legion [13] builds system components on a distributed object-oriented model, and the Andrew File System (AFS) [21] provides a uniform view of file resources. Each of these systems has been proven effective in large-scale application experiments.

Our goal in the Globus project is not to compete with these and other related efforts, but rather to provide basic infrastructure that can be used to construct portable, high-performance implementations of a range of such services. To this end, we focus on (a) the development of low- level mechanisms that can be used to implement higher-level services, and (b) techniques that allow those services to observe and guide the operation of these mechanisms. If successful, this approach can reduce the complexity and improve the quality of metacomputing software by allowing a single low-level infrastructure to be used for many purposes, and by providing solutions to the configuration problem in metacomputing systems.

To demonstrate that the Globus approach is workable, we must show that it is possible to use a single set of low-level mechanisms to construct efficient implementations of diverse services on multiple platforms. In the following, we first introduce the Globus toolkit and then describe the mechanisms that allow higher-level services to observe and guide the operation of toolkit components.

6.1 The Globus Toolkit

The Globus toolkit comprises a set of modules. Each module defines an *interface*, which higher-level services use to invoke that module's mechanisms, and provides an *implementation*, which uses appropriate low-level operations to implement these mechanisms in different environments.

Currently identified toolkit modules are as follows. These descriptions focus on requirements; Sections 7 and 9 address implementation status.

- *Resource location and allocation.* This component provides mechanisms for expressing application resource requirements, for identifying resources that meet these requirements, and for scheduling resources once they have been located. Resource location mechanisms are required because applications cannot, in general, be expected to know the exact location of required resources, particularly when load and resource availability can vary. Resource allocation involves scheduling the resource and performing any initialization required for subsequent process creation, data access, etc. In some situations—for example, on some supercomputers—location and allocation must be performed in a single step.
- *Communications.* This component provides basic communication mechanisms. These mechanisms must permit the efficient implementation of a wide range of communication methods, including message passing, remote procedure call, distributed shared memory, stream-based, and multicast. Mechanisms must be cognizant of network quality of service parameters such as jitter, reliability, latency, and bandwidth.
- *Unified resource information service.* This component provides a uniform mechanism for obtaining real-time information about metasystem structure and status. The mechanism must allow components to post as well as receive information. Support for scoping and access control is also required.
- *Authentication interface.* This component provides basic authentication mechanisms that can be used to validate the identity of both users and resources. These mechanisms provide building blocks for other security services such as authorization and data security that need to know the identity of parties engaged in an operation.
- *Process creation.* This component is used to initiate computation on a resource once it has been located and allocated. This task includes setting up executables, creating an execution environment, starting an executable, passing arguments, integrating the new process into the rest of the computation, and managing termination and process shutdown.
- *Data access.* This component is responsible for providing high- speed remote access to persistent storage such as files. Some data resources such as databases may be accessed via distributed database technology or the Common Object Request Broker Architecture (CORBA). The Globus data access module addresses the problem of achieving high performance when accessing parallel file systems and network-enabled I/O devices such as the High Performance Storage System (HPSS).

Together, the various Globus toolkit modules can be thought of as defining a *metacomputing virtual machine*. The definition of this virtual machine simplifies application development and enhances portability by allowing programmers to think of geographically distributed, heterogeneous collections of resources as unified entities.

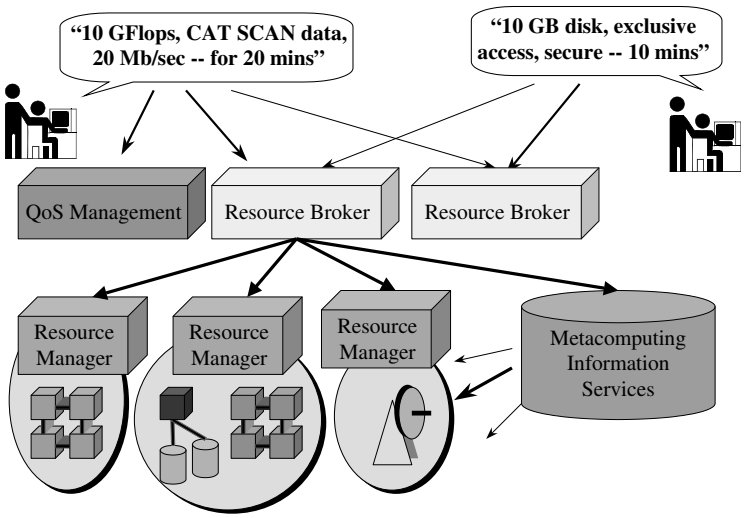

Figure 6: Resource selection and monitoring

6.2 Support for Resource-Aware Services and Applications

Metacomputing applications often need to operate networking and computing resources at close to maximum performance. Hence, metacomputing environments must allow programmers to observe differences in system resource characteristics and to guide how these resources are used to implement higher-level services. Achieving these goals without compromising portability is a significant challenge for the designer of metacomputing software.

We use the Globus communication module to illustrate some of these issues. This module must select, for each call to its communication functions, one of several low-level mechanisms. On a local area network, communication might be performed with TCP/IP, while in a parallel computer, specialized high-performance protocols typically offer higher bandwidth and lower latencies. In a wide area environment, specialized ATM protocols can be more efficient. The ability to manage protocol parameters (TCP packet size, network quality of service) further complicates the picture. The choice of low- level mechanism used for a particular communication is a nontrivial problem that can have significant implications for application performance.

Globus toolkit modules address this problem by providing interfaces that allow the selection process to be exposed to, and guided by, higher-level tools and applications. These interfaces provide rule-based *selection*, resource property *inquiry*, and *notification* mechanisms.

- *Rule-based selection.* Globus modules can identify selection points at which choices from among alternatives (resources, parameter values, etc.) are made.

Associated with each selection point is a default selection rule provided by the module developer (e.g., "use TCP packet size X," "use TCP over ATM"). A rule replacement mechanism allows higher-level services to specify alternative strategies ("use TCP packet size Y," "use specialized ATM protocols").

- *Resource property inquiry.* Information provided by the unified information service (Section 7) can be used to guide selection processes within both Globus modules and applications that use these modules. For example, a user might provide a rule that states "use ATM interface if load is low, otherwise Internet," hence using information about network load to guide resource selection.
- *Notification.* A notification mechanism allows a higher-level service or application to specify constraints on the quality of service delivered by a Globus service and to name a call-back function that should be invoked if these constraints are violated. This mechanism can be used, for example, to switch between networks when one becomes loaded.

Higher-level services and applications can use Globus selection, inquiry, and notification mechanisms to *configure* computations efficiently for available resources, and/or to *adapt* behavior when the quantity and/or quality of available resources changes dynamically during execution. For example, consider an application that performs computation on one computer and transfers data over a wide area network for visualization at remote sites. At startup time, the application can determine available computational power and network capacity and configure its computational and communication structures appropriately (e.g., it might decide to use compression for some data but not others). During execution, notification mechanisms allow it to adapt to changes in network quality of service.

We use the term Adaptive Wide Area Resource Environment (AWARE) to denote a set of application interfaces, higher-level services, and adaptation policies that enable specific classes of applications to exploit a metacomputing environment efficiently. We are investigating AWARE components for several applications, and anticipate developing AWARE toolkits for different classes of metacomputing application.

7 Globus Toolkit Components

We now describe in more detail the Globus communications, information service, authentication, and data access services. In each case, we outline how the component maps to different implementations, is used to implement different higher-level services, and supports the development of AWARE services and applications.

7.1 Communications

The Globus communications module is based on the Nexus communication library [11]. Nexus defines five basic abstractions: nodes, contexts, threads, communication links, and remote service requests. The Nexus functions that manipulate these abstractions constitute the Globus communication interface. This interface is used extensively by other Globus modules and has also been used to construct various higher-level services, including parallel programming tools (Section 8.1). The Active Messages [19] and Fast Messages [24] systems have similarities in goals and approach, but there are also significant differences [7].

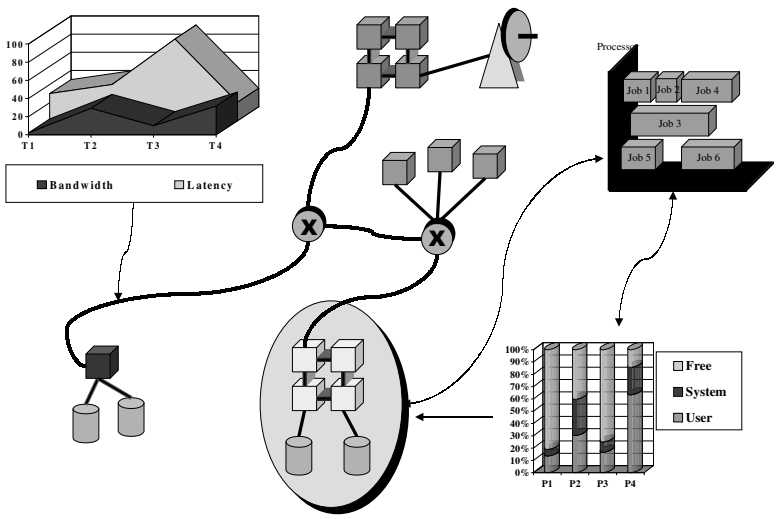

Figure 7: Different Layers of a telemedicine environment

Nexus programs bind communication startpoints and endpoints to form communication links. If multiple startpoints are bound to an endpoint, incoming communications are interleaved, in the same manner as messages sent to the same node in a message passing system. If a startpoint is bound to multiple endpoints, communication results in a multicast operation. A startpoint can be copied between processors, causing new communication links to be created that mirror the links associated with the original startpoint. This support for copying means that startpoints can be used as global names for objects. These names can be communicated and used anywhere in a distributed system.

A communication link supports a single communication operation: an asynchronous *remote service request* (RSR). An RSR is applied to a startpoint by providing a procedure name and a data buffer. For each endpoint linked to the startpoint, the RSR transfers the data buffer to the address space in which the endpoint is located and remotely invokes the specified procedure, passing the endpoint and the data buffer as arguments. A local address can be associated with an endpoint, in which case startpoints associated with the endpoint can be thought of as "global pointers to that address.

The Nexus interface and implementation support rule-based selection (Section 6.2) of the methods—such as protocol, compression method, and quality of service—used to perform communication [7]. Different communication methods can be associated with different communication links, with selection rules determining which method should be used when a new link is established. These mechanisms have been used to support multiple communication protocols [7] and selective use of secure communication [10] in heterogeneous environments.

7.2 Metacomputing Directory Service

As noted above, metacomputing environments depend critically on access to information about the underlying networked supercomputing system. Required information can include the following.

- Configuration details about resources such as the amount of memory, CPU speed, number of nodes in a parallel computer, or the number and type of network interfaces available.
- Instantaneous performance information, such as point-to-point network latency, available network bandwidth, and CPU load.
- Application-specific information, such as memory requirements or program structures found effective on previous runs.

Different data items will have different scopes of interest and security requirements, but some information at least may potentially be required globally, by any system component. Information may be obtained from multiple sources: for example, from standard information services such as the Network Information Service (NIS) or Simple Network Management Protocol (SNMP); from specialized services such as the Network Weather Service [27]; or from external sources such as the system manager or an application.

Globus defines a single, unified access mechanism for this wide range of information, called the Metacomputing Directory Service (MDS) [6]. Building on the data representation and application programming interface defined by the Lightweight Directory Access Protocol (LDAP), MDS defines a framework in which can be represented information of interest in distributed computing applications. Information is structured as a set of entries, where each entry comprises zero or more attribute-value pairs. The type of an entry, called its object class, specifies mandatory and optional attributes (see Figure 8).

MDS information can be maintained within conventional LDAP servers. However, the performance and functionality requirements of metacomputing applications motivate a number of extensions. A proxy mechanism supports the integration of other data services, such as SNMP and NIS, and allows remote access. We are currently investigating approaches to caching and replication, and the definition of a notification interface that will allow higher-level services or applications to request notifications when specified conditions become true.

7.3 Authentication Methods

The initial version of the Globus authentication module supported password, Unix RSH, and Secure Socket Layer authentication. To increase the degree of abstraction at the toolkit interface, we are moving towards the use of the Generic Security System (GSS) [17]. GSS defines a standard procedure and API for obtaining credentials (passwords or certificates), for mutual authentication (client and server), and for message-oriented encryption and decryption. GSS is independent of any particular security mechanism and can be layered on top of different security methods, such as Kerberos and SSL.

GSS must be altered and extended to meet the requirements of metacomputing environments. As a metacomputing system may use different authentication mechanisms in different situations and for different purposes, we require a GSS implementation that supports the concurrent use of different security mechanisms. In addition,

```
GlobusHost OBJECT CLASS
  SUBCLASS OF GlobusResource
  MUST CONTAIN {
    hostName          :: cis,
    type              :: cis,
    vendor            :: cis,
    model             :: cis,
    OStype            :: cis,
    OSversion         :: cis
  }
  MAY CONTAIN {
    networkNode        :: dn,
    totalMemory        :: cis,
    totalSwap          :: cis,
    dataCache          :: cis,
    instructionCache   :: cis
  }
```

```
GlobusResource OBJECT CLASS
        SUBCLASS OF top
        MUST CONTAIN {
          administrator :: dn
        }
        MAY CONTAIN {
          manager       :: dn,
          provider      :: dn,
          technician    :: dn,
          description   :: cis,
          documentation :: cis
        }
```

Figure 8: Simplified versions of the MDS object classes `GlobusHost` and `GlobusResource`

inquiry and selection functions are needed so that higher-level services implementing specific security policies can select from available low- level security mechanisms.

7.4 Data Access Services

Services that provide metacomputing applications with access to persistent data can face stringent performance requirements and must support access to data located in multiple administrative domains. Distributed file systems such as the Network File System and Distributed File System address remote access to some extent but have not been designed for high-performance applications. Parallel file systems and I/O libraries have been designed for performance but not for distributed execution.

To address these problems, the Globus data access module defines primitives that provide remote access to parallel file systems. This remote I/O (RIO) interface (Figure **??**) is based on the abstract I/O device (ADIO) interface [26]. ADIO defines an interface for opening, closing, reading and writing parallel files. It does not define semantics for caching, file replication, or parallel file descriptor semantics. Several popular I/O systems have been implemented efficiently on ADIO [26]. RIO extends ADIO by adding transparent remote access and global naming using a URL- based naming scheme. RIO remote access features use Nexus mechanisms.

8 Higher-Level Services

In the preceding section, we described the core interfaces and services provided by the Globus toolkit. These interfaces are not intended for application use. Rather, they are intended to be used to construct higher-level *policy* components. These policy components can serve the function of middleware, on which yet higher-level

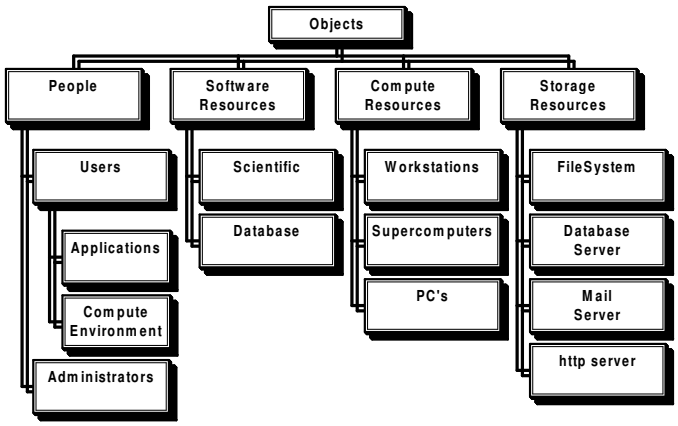

Figure 9: Classification of metacomputing objects

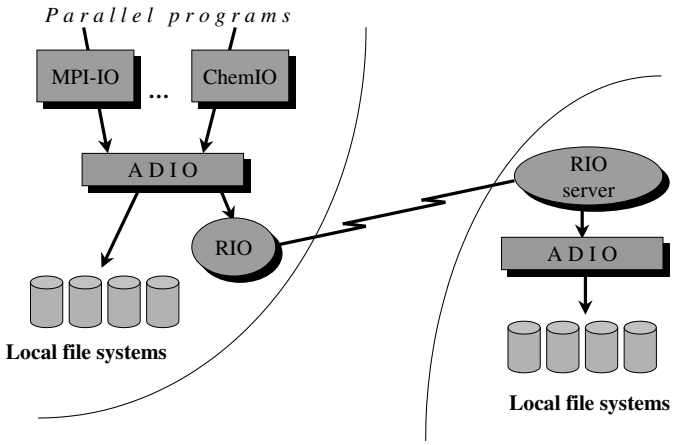

Figure 10: Remote IO mechanisms for high-performance acess to remote file systems

components are constructed to create application-level interfaces. In the following, we describe two such middleware components.

8.1 Parallel Programming Interfaces

Numerous parallel programming interfaces have been adapted to use Globus authentication, process creation, and communication services, hence allowing programmers to develop metacomputing applications using familiar tools. These interfaces include a complete implementation of MPI (and hence tools layered on top of MPI, such as many High Performance Fortran systems); Compositional C++ [3], a parallel extension to C++; Fortran M, a task- parallel Fortran; nPerl, a version of the Perl scripting language extended with remote reference and remote procedure call mechanisms; and NexusJava, a Java class library that supports remote procedure calls.

We say a few words here about the Globus implementation of MPI. A "Nexus device" supports the abstract device interface used within the MPICH implementation of MPI [14]. This device uses Nexus RSRs to implement the basic data transfer operations required by MPI [9]; higher-level MPI functionality then transfers unchanged from MPICH. In the initial MPI/Nexus implementation, an overhead for a zero-length message of 60 μsec was noted on an IBM SP2 with Power 1 processors (raw MPICH zero-length message cost is 83.8 μsec, and raw Nexus RSR cost is 82.8 μsec); the overhead for larger messages is insignificant. A recent redesign of both Nexus and the MPICH abstract device interface has succeeded in eliminating most of this overhead. In addition to this use of Nexus, the Globus implementation of MPI uses Globus mechanisms for authentication and process startup [10]. These components are sufficiently integrated that in the I-WAY a user could allocate a heterogeneous collection of resources and then start a program simply by typing "`impirun`." Under the covers, of course, Globus mechanisms are used to select appropriate authentication, process creation, and communication mechanisms.

8.2 Unified Certificate-based Authentication

The Globus authentication interface can be used to implement a range of different security policies. We are currently investigating a policy that defines a global, public key-based authentication space for all users and resources. That is, we provide a centralized authority that defines system-wide names ("accounts") for users and resources. These names allow an application to use a single "user id" and "password" for all resources. They also permit the application to verify the identity of requested resources. Note that this policy does not address authorization: resources can use their usual mechanisms to determine the users to which they will grant access.

While not practical for large-scale, open environments, the use of a centralized authority to identify users and resources is appropriate for limited-scale testbed environments such as the I-WAY and GUSTO (see below). The approach has the significant advantage that it can be implemented easily with current certificate-based authentication protocols, such as that provided in the Secure Socket Library (SSL). Note that while names (that is, certificates) are issued by a centralized certificate authority, the authentication of users and services involves only the agents being authenticated; it does not require any interaction with the issuing authority.

In the longer term, authentication and authorization schemes must address the requirements of larger, dynamic, heterogeneous communities, in which trust relationships span multiple administrative domains and can be irregular and selective. Some

member organizations will be more trusting of specific members than others; still others may be competitors. Some members may feel the need to control all trust relationships explicitly, even if it means that fewer community assets are available for their use; this may stimulate the evolution of sub-communities with their own set of trust and authorization relationships. Community members willing to delegate to other members the ability to extend relationships on their behalf may more fully enjoy the benefits of membership in the greater community.

Our certificate-based policy can be extended to support limited forms of trust delegation. Globus resource certificates can be given to sites with multiple resources (or users). These sites can in turn set up a local certificate authority, which signs certificates that it issues with the certificate issued by the Globus authority. This situation is acceptable if the user (or resource) trusts the administration of the site issuing the Globus-signed certificate.

9 Globus Testbeds and Experiences

We have referred above to the I-WAY experiment, in which a number of Globus components were first deployed as part of the I-Soft software environment [8]. These components included the Nexus communication library, Kerberos-based authentication, a process creation mechanism, and a centralized scheduler for compute resources. While inadequate in a number of respects (e.g., nonscalable, no scheduling of network resources), this experiment did demonstrate the advantages of the Globus approach of providing basic mechanisms. I-WAY applications developed with a variety of parallel tools (MPI, CAVEcomm, CC++, etc.) were ported to the I-WAY environment by adapting those tools to use Globus mechanisms for authentication, process creation, and communication. If Globus had enforced a particular parallel programming interface, considerable effort would have been required to adapt existing applications to use this interface.

We are currently working with the high-performance computing community to define additional testbeds. We describe here just one of these, the Globus Ubiquitous Supercomputing Testbed (GUSTO). GUSTO is intended initially at least as a computer science rather than an application testbed, meaning that the initial development focus is on deploying and evaluating basic mechanisms for authentication, scheduling, communication, and information infrastructure. In this respect it complements efforts such as the I-WAY that focus more on applications.

GUSTO is intended to span approximately fifteen sites, encompassing a number of supercomputers (IBM SP2, Silicon Graphics Power Challenge, etc.), and workstations. Basic connectivity is via the Internet, although some machines are also connected via OC3 (155 Mb/sec) ATM networks. Eventually, we expect most GUSTO sites to be accessible over the National Science Foundation's OC3 vBNS network. Authentication is based on the uniform certificate mechanism described above. The initial information service is provided by an information server that maintains information about resource configuration and current network characteristics [6]. This information is updated dynamically, providing a more accurate view of system configuration than was available in I-Soft [8].

10 Requirements

11 Future opertunities

Interdisciplinary research

weather forecast - medical conditions environmental research - helath conditions geographical data bases ...

12 Conclusion

The Globus project is attacking the metacomputing software problem from the bottom up, by developing basic mechanisms that can be used to implement a variety of higher-level services. Communication, resource location, resource allocation, information, authentication, data access, and other services have been identified, and considerable progress has been made toward constructing quality implementations. The definition, development, application, evaluation, and refinement of these components are ongoing processes that we expect to proceed for the next two years at least. We hope to involve more of the metacomputing community in this process, by adapting relevant higher- level services (e.g., application-level scheduling [1], performance steering [25], object-based libraries [13]) to use Globus mechanisms, and by participating in the construction of additional testbeds (e.g., GUSTO).

The Globus project is also addressing the configuration problem in metacomputing systems, with the goal of producing an Adaptive Wide Area Resource Environment that supports the construction of adaptive services and applications. We have introduced selection, information, and notification mechanisms and have defined Globus component interfaces so that these mechanisms can be used to guide the configuration process. Preliminary experiments with dynamic communication selection suggest that these configuration mechanisms can have considerable value [7].

In summary, we list three areas in which we believe the Globus project has already made contributions and in which we hope to see considerable further progress:

- The definition of a core metacomputing system architecture on which a range of alternative metacomputing environments can be built.
- The development of a framework that allows applications to respond to dynamic behaviors in the underlying metacomputing environment, and the definition and evaluation of various adaptation policies.
- The demonstration in testbeds such as the I-WAY and GUSTO that useful higher-level services can be layered effectively on top of the interfaces defined by the Globus toolkit, and that automatic configuration mechanisms can be used to enhance portability and performance.

Telemedicine.

13 Acknowledgements

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, *Application-level scheduling on distributed heterogeneous networks*, in Proceedings of Supercomputing '96, ACM Press, 1996.

[2] C. Catlett and L. Smarr, *Metacomputing*, Communications of the ACM, 35 (1992), pp. 44–52.

[3] K. M. Chandy and C. Kesselman, *CC++: A declarative concurrent object oriented programming notation*, in Research Directions in Object Oriented Programming, The MIT Press, 1993, pp. 281–313.

[4] D. Diachin, L. Freitag, D. Heath, J. Herzog, W. Michels, and P. Plassmann, *Remote engineering tools for the design of pollution control systems for commercial boilers*, International Journal of Supercomputer Applications, 10 (1996), pp. 208–218.

[5] T. L. Disz, M. E. Papka, M. Pellegrino, and R. Stevens, *Sharing visualization experiences among remote virtual environments*, in International Workshop on High Performance Computing for Computer Graphics and Visualization, Springer-Verlag, 1995, pp. 217–237.

[6] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, *A directory service for configuring high-performance distributed computations*, in Proc. 6th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1997, pp. 365–375.

[7] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, *Managing multiple communication methods in high-performance networked computing systems*, Journal of Parallel and Distributed Computing, 40 (1997), pp. 35–48.

[8] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke, *Software infrastructure for the I-WAY high-performance distributed computing experiment*, in Proc. 5th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1996, pp. 562–571.

[9] I. Foster, J. Geisler, and S. Tuecke, *MPI on the I-WAY: A wide-area, multimethod implementation of the Message Passing Interface*, in Proceedings of the 1996 MPI Developers Conference, IEEE Computer Society Press, 1996, pp. 10–17.

[10] I. Foster, N. Karonis, C. Kesselman, G. Koenig, and S. Tuecke, *A secure communications infrastructure for high-performance distributed computing*, in Proc. 6th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1997, pp. 125–136.

[11] I. Foster, C. Kesselman, and S. Tuecke, *The Nexus approach to integrating multithreading and communication*, Journal of Parallel and Distributed Computing, 37 (1996), pp. 70–82.

[12] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Network Parallel Computing*, MIT Press, 1994.

[13] A. Grimshaw and W. Wulf, *Legion – a view from 50,000 feet*, in Proc. 5th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1996, pp. 89–99.

[14] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Parallel Computing, 22 (1996), pp. 789–828.

[15] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, The MIT Press, 1994.

[16] C. Lee, C. Kesselman, and S. Schwab, *Near-realtime satellite image processing: Metacomputing in CC++*, Computer Graphics and Applications, 16 (1996), pp. 79–84.

[17] J. Linn, *Generic security service application program interface*, Internet RFC 1508, (1993).

[18] M. Litzkow, M. Livney, and M. Mutka, *Condor - a hunter of idle workstations*, in Proc. 8th Intl Conf. on Distributed Computing Systems, 1988, pp. 104–111.

[19] A. Mainwaring, *Active Message applications programming interface and communication subsystem organization*, tech. rep., Dept. of Computer Science, UC Berkeley, Berkeley, CA, 1996.

[20] C. Mechoso et al., *Distribution of a Coupled-ocean General Circulation Model across high-speed networks*, in Proceedings of the 4th International Symposium on Computational Fluid Dynamics, 1991.

[21] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Smith, *Andrew: A distributed personal computing environment*, Communications of the ACM, 29 (1986), pp. 184–201.

[22] J. Nieplocha and R. Harrison, *Shared memory NUMA programming on the I-WAY*, in Proc. 5th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1996, pp. 432–441.

[23] M. Norman, P. Beckman, G. Bryan, J. Dubinski, D. Gannon, L. Hernquist, K. Keahey, J. Ostriker, J. Shalf, J. Welling, and S. Yang, *Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing*, International Journal of Supercomputer Applications, 10 (1996), pp. 131–140.

[24] S. Pakin, M. Lauria, and A. Chien, *High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet*, in Proceedings of Supercomputing '95, IEEE Computer Society Press, 1996.

[25] D. Reed, C. Elford, T. Madhyastha, E. Smirni, and S. Lamm, *The Next Frontier: Interactive and Closed Loop Performance Steering*, in Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing, Aug. 1996, pp. 20–31.

[26] R. Thakur, W. Gropp, and E. Lusk, *An abstract-device interface for implementing portable parallel-I/O interfaces*, in Proceedings of The 6th Symposium on the Frontiers of Massively Parallel Computation, October 1996.

[27] R. Wolski, *Dynamically forecasting network performance using the network weather service*, Tech. Rep. TR-CS96-494, U.C. San Diego, October 1996.