Globus:
A Metacomputing Infrastructure Toolkit

Ian Foster* Carl Kesselman†
http://www.globus.org/

Abstract

Emerging high-performance 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 scientific instruments to form *networked virtual supercomputers* or *metacomputers*. While the physical infrastructure to build such systems is becoming widespread, the heterogeneous and dynamic nature of the metacomputing environment poses new challenges for developers of system software, parallel tools, and applications. In this article, we introduce Globus, a system that we are developing to address these challenges. The Globus system is intended to achieve a vertically integrated treatment of application, middleware, and network. A low-level *toolkit* provides basic mechanisms such as communication, authentication, network information, and data access. These mechanisms are used to construct various higher-level metacomputing *services*, such as parallel programming tools and schedulers. Our long-term goal is to build an Adaptive Wide Area Resource Environment (AWARE), an integrated set of higher-level services that enable applications to adapt to heterogeneous and dynamically changing metacomputing environments. Preliminary versions of Globus components were deployed successfully as part of the I-WAY networking experiment.

1 Introduction

New classes of high-performance applications are being developed that require unique capabilities not available in a single computer. Such applications are enabled by 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.

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 [24, 25], collaborative engineering [5, 6], and supercomputer-enhanced scientific instruments [18].

*Mathematics and Computer Science Division, Argonne National Laboratory, foster@mcs.anl.gov
†Information Sciences Institute, University of Southern California, carl@isi.edu

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* 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 1). 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 2, we introduce general characteristics of metacomputing systems and requirements for metacomputing infrastructure. In Section 3, we describe the Globus architecture and the techniques used to support resource-aware applications. In Section 4, we describe major toolkit components. In Sections 5 and 6, we describe higher-level services constructed with the toolkit and testbeds that have deployed toolkit components. We conclude in Section 7 with a discussion of current system status and future plans.

2 Metacomputing

We use the term *metacomputer* to denote a networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high-speed networks. Figure 2 illustrates an example of such a system; a somewhat simplified version of this architecture was used in the I-WAY for real-time image processing of a data stream from a meteorological satellite [18]. In that application, data passed from the satellite downlink to a a cloud detection program running on a remote supercomputer, and then to a graphics computer for rendering.

Metacomputing, like more mainstream applications of distributed computing, is moti-

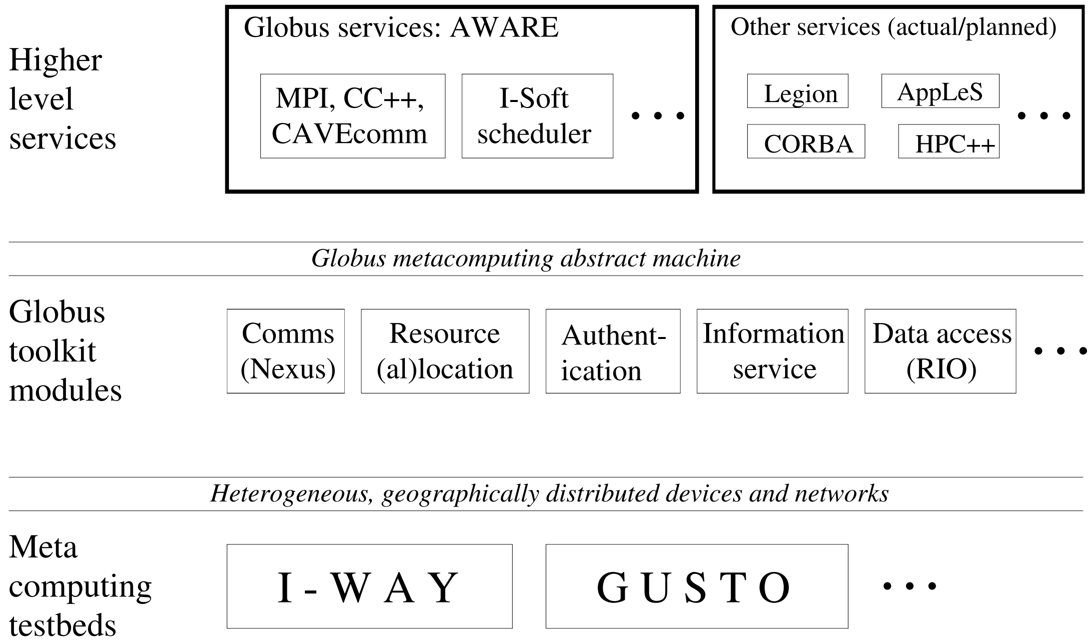

FIG. 1. *The Globus toolkit*

vated 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 engineering environment that connects the virtual reality systems, design databases, and supercomputers required to work on a particular engineering problem. Finally, 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.

2.1 Metacomputing Applications

Scientists and engineers are just beginning to explore the new applications enabled by networked supercomputing. The I-WAY experiment [4] identified four significant application 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.

2. *Smart instruments.* These applications connect users to instruments such as microscopes, telescopes, or satellite downlinks [18] that are themselves coupled with remote supercomputers. This computational enhancement can enable both quasi-realtime processing of instrument output and interactive steering.

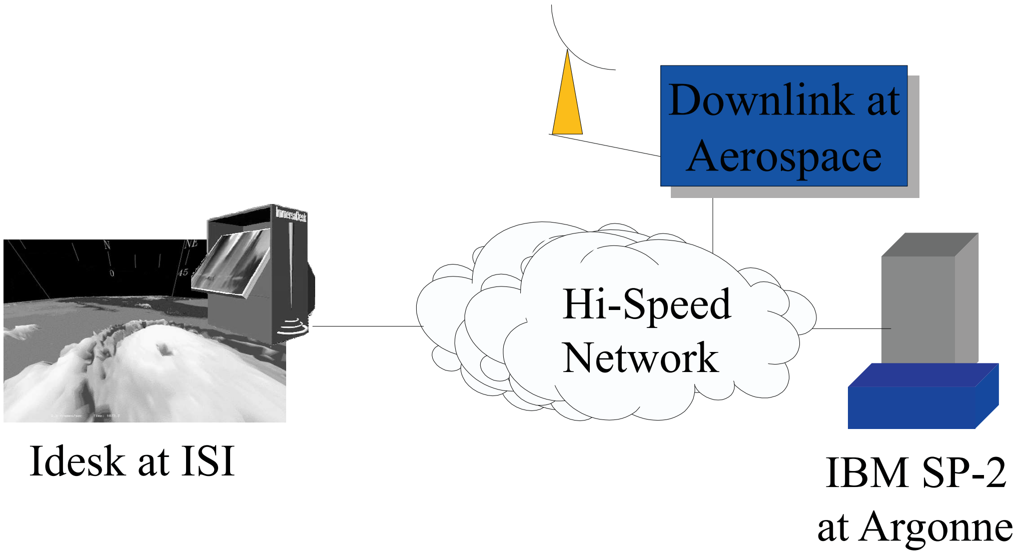

FIG. 2. *This figure shows a networked supercomputing system used during the I-WAY experiment for the real-time analysis of data from a meteorological satellite. The satellite downlink is located in El Segundo, California; the supercomputer at Argonne, Illinois; and the visualization engine and graphics device in Los Angeles.*

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 [5, 6].

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 [22, 24, 25]. 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 [20] 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.

2.2 Metasystem Characteristics

While the I-WAY and other networking testbeds have provided intriguing hints of what future metacomputing systems may look like and how they may be used, the definition and application of such systems remain research problems. Nevertheless, we can make general observations about their characteristics (see also [15]).

- *Scale and the need for selection.* To date, most metacomputing experiments have been performed on relatively small testbeds, with the 17-site I-WAY being the largest. In the future we can expect to deal with much larger collections, from which resources will be selected for particular applications according to criteria such as connectivity, cost, security, and reliability.

- *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 can arise 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 can be required 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 often 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.

3 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) [13] and the Message Passing Interface (MPI) [17] provide a machine-independent communication layer, Condor [20] provides a uniform view of processor resources, Legion [14] builds system components on a distributed object-oriented model, and the Andrew File System (AFS) [23] 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. As we report below, we have had some initial successes in the communication area: our Nexus communication library has been used to construct high-performance implementations of diverse parallel programming interfaces. However, this is certainly not a conclusive demonstration, and the final verdict on the Globus approach must await the results of further research.

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.

3.1 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 4 and 6 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.

3.2 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 4) 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.

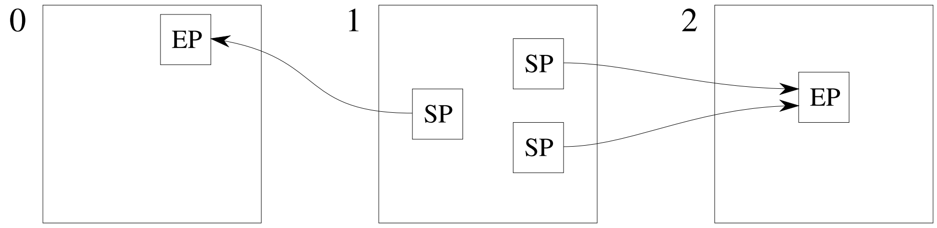

FIG. 3. *Nexus communication mechanisms. The figure shows three contexts (address spaces) and three communication links. Three startpoints in context 1 reference endpoints in contexts 0 and 2.*

- *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.

4 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.

4.1 Communications

The Globus communications module is based on the Nexus communication library [12]. Nexus defines five basic abstractions: nodes, contexts, threads, communication links, and remote service requests (Figure 3). 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 5.1). The Active Messages [21] and Fast Messages [26] systems have similarities in goals and approach, but there are also significant differences [8].

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 3.2) of the methods—such as protocol, compression method, and quality of service—used to perform communication [8]. 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 [8] and selective use of secure communication [11] in heterogeneous environments.

4.2 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 [29]; 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) [7]. 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 4).

```
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
  }
```

Fig. 4. *Simplified versions of the MDS object classes* `GlobusHost` *and* `GlobusResource`

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.

4.3 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) [19]. 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, inquiry and selection functions are needed so that higher-level services implementing specific security policies can select from available low-level security mechanisms.

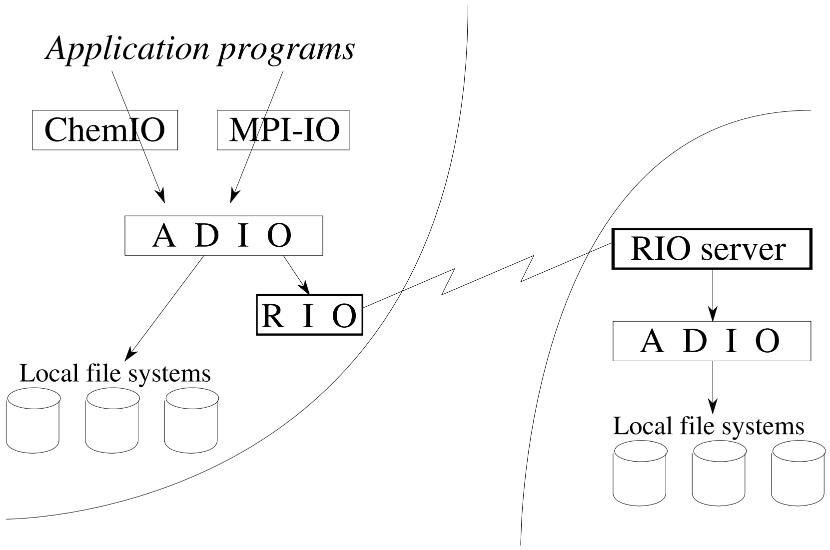

Fig. 5. *RIO mechanisms for high-performance access to remote file systems*

4.4 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 5) is based on the abstract I/O device (ADIO) interface [28]. 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 [28]. RIO extends ADIO by adding transparent remote access and global naming using a URL-based naming scheme. RIO remote access features use Nexus mechanisms.

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

5.1 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 [16]. This device uses Nexus RSRs to implement the basic data transfer operations required by MPI [10]; 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 [11]. 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.

5.2 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.

6 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 [9]. 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 [7]. This information is updated dynamically, providing a more accurate view of system configuration than was available in I-Soft [9].

7 Conclusions and Future Work

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 [27], object-based libraries [14]) 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 [8].

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.

Acknowledgments

We gratefully acknowledge the contributions made by Steve Tuecke, Jonathan Geisler, Craig Lee, Steve Schwab, Warren Smith, Jace Mogill, and John Garnett to the design and implementation of Globus components. It is also a pleasure to acknowledge the contributions of Tom DeFanti and Rick Stevens and the many other participants in the I-WAY project.

This work was supported by the National Science Foundation's Center for Research in Parallel Computation, under Contract CCR-8809615, by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523, and 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.

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] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss, *Overview of the I-WAY: Wide area visual supercomputing*, International Journal of Supercomputer Applications, 10 (1996), pp. 123–130.

[5] 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.

[6] 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.

[7] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, *A directory service for configuring high-performance distributed computations*, preprint, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1997.

[8] 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, (1997). To appear.

[9] 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.

[10] 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.

[11] I. Foster, N. Karonis, C. Kesselman, G. Koenig, and S. Tuecke, *A secure communications infrastructure for high-performance distributed computing*, Preprint ANL/MCS-P613-0996, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1996.

[12] 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.

[13] 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.

[14] A. Grimshaw and W. Wolf, *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.

[15] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds, Jr., *Legion: The next logical step toward a nationwide virtual computer*, Tech. Rep. CS-94-21, Department of Computer Science, University of Virginia, 1994.

[16] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, Technical Report ANL/MCS-TM-213, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1996.

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1995.

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

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

[20] 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.

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

[22] 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.

[23] J. Morris et al., *Andrew: A distributed personal computing environment*, Communications of the ACM, 29 (1986).

[24] 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.

[25] M. Norman et al., *Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing*, International Journal of Supercomputer Applications, 10 (1996), pp. 131–140.

[26] 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.

[27] 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.
[28] 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.
[29] R. Wolski, *Dynamically forecasting network performance using the network weather service*, Tech. Rep. TR-CS96-494, U.C. San Diego, October 1996.