Application Experiences with the Globus Toolkit

Sharon Brunett[‡] Karl Czajkowski[†] Ian Foster[*]
Carl Kesselman[†] Jason Leigh[§] Steven Tuecke[*]

[*]Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

[†]Information Sciences Institute
University of Southern California
Marina Del Rey, CA 90292-6695

[‡]Center for Advanced Computing Research
California Institute of Technology
Pasadna, CA 91125

[§]Electronic Visualization Laboratory
University of Illinois at Chicago
Chicago, IL 60607

1 Introduction

The Globus grid toolkit is a collection of software components designed to support the development of applications for high-performance distributed computing environments, or "computational grids" [14]. The Globus toolkit is an implementation of a "bag of services" architecture, which provides application and tool developers not with a monolithic system but rather with a set of stand-alone services. Each Globus component provides a basic service, such as authentication, resource allocation, information, communication, fault detection, and remote data access. Different applications and tools can combine these services in different ways to construct "grid-enabled" systems.

The Globus toolkit has been used to construct the Globus Ubiquitous Supercomputing Testbed, or GUSTO: a large-scale testbed spanning 20 sites and included over 4000 compute nodes for a total compute power of over 2 TFLOPS. Over the past six months, we and others have used this testbed to conduct a variety of application experiments, including multi-user collaborative environments (tele-immersion), computational steering, distributed supercomputing, and high throughput computing.

The goal of this paper is to review what has been learned from these experiments regarding the effectiveness of the toolkit approach. To this end, we describe two of the application experiments in detail, noting what worked well and what worked less well. The two applications are a distributed supercomputing application, SF-Express, in which multiple supercomputers are harnessed to perform large distributed interactive simulations; and a tele-immersion application, CAVERNsoft, in which the focus is on connecting multiple people to a distributed simulated world.

We believe that the results of these experiments indicate that the Globus toolkit architecture is effective, at least for the applications considered to date. Large, complex applications can be either adapted to execute in a grid environment (e.g., SF-Express) or developed from scratch for grid computing (e.g., CAVERNsoft) without unusual difficulty, and with a saving in cost, complexity, and usability relative to similar codes developed without our toolkit. The

DISCLAIMER

Table 1: Core Globus services. As of early 1998, these include only those services deemed essential for an evaluation of the Globus design philosophy on realistic applications and in medium-scale grid environments. Other services such as accounting, auditing, and instrumentation will be addressed in future work

Service	Name	Description
Resource management	GRAM	Resource allocation and process management
Communication	Nexus	Unicast and multicast communication services
Security	GSI	Authentication and related security services
Information	MDS	Distributed access to structure and state information
Health and status	HBM	Monitoring of health and status of system components
Remote data access	GASS	Remote access to data via sequential and parallel interfaces
Executable management	GEM	Construction, caching, and location of executables

experiments also point to areas in which further work is required: for example, code and data management, and fault tolerance.

In the rest of the paper, we provide a brief description of the Globus toolkit and describe two applications in detail: the SF-Express and CAVERNsoft and describe how these applications use Globus services. We conclude with a discussion of lessons learned and future work.

We note that the Globus toolkit is just one of several approaches being pursued to the construction of grid applications; others include Legion [16], Web technologies [15, 25], and CORBA [17]. Space does not permit comparisons with these alternative approaches.

2 The Globus Toolkit

The Globus toolkit comprises the core services listed in Table 1, plus a selection of higher-level services defined in terms of these core services. Each core service defines an Application Program Interface (API) that provides a uniform interface to a local service. For example, the Globus Resource Allocation Manager (GRAM) service provides an API for requesting that computations be started on a computational resource, and for managing those computations once they are started [10]. Higher-level services use core services to implement more complex global functionality. For example, resource brokers and co-allocators use services provided by GRAMs and by the Globus information service (the Metacomputing Directory Service, or MDS [12]) to locate available resources and to start computations across computations of resources, respectively. Application-specific scheduling techniques [3] could also be used; see also [8, 26, 22, 1].

The full paper will provide a more detailed description of the Globus toolkit. Individual components are described in detail in other papers [14, 13, 12, 10].

3 SF-Express

The first application that we consider, SF-Express, is a distributed interactive simulation (DIS) application that harnesses multiple supercomputers to meet the computational demands of

large-scale network-based simulation environments [21]. A large simulation may involve many tens of thousands of entities and requires thousands of processors. Globus services can be used to locate, assemble, and manage those resources. For example, in one experiment in November 1997, SF-Express was run on 852 processors distributed over six GUSTO sites [6].

An SF-Express computation is distributed across a large number of *simulation nodes*, where each node is responsible for simulating the behavior of entities assigned to it. During the simulation, entity state information, such as the position and velocity, is exchanged. Current DIS implementations broadcast state update information, an approach that is not practical for the large-scale simulations. SF-Express uses a technique called *interest management* to reduce the amount of communication required. Simulation nodes are organized into groups and a *router node* is associated with each group. Routers keep track of which simulation nodes contain entities that can potentially interact with one another, and entity state updates are sent only between those routers responsible for interacting simulation nodes. Entities are assigned to nodes in a way that preserves physical locality to further reduce communication requirements.

3.1 SF-Express Experiences

SF-Express is interesting as a test case for the Globus toolkit because a distributed implementation existed before work started on the use of Globus services. As illustrated in Table 2 and explained in the following, a Globus-based implementation was constructed (and continues to be constructed) by incorporating Globus services incrementally to improve existing functionality or add new functionality. This incremental process made it possible to study the impact on complexity and usability of incorporating Globus components.

Resource allocation and security. Prior to the use of Globus services, simply starting SF-Express on multiple supercomputers was a painful task. The user had to log in to each site in turn and recall the arcane commands needed to allocate resources and start a program. This obstacle to the use of distributed resources was overcome by encoding resource allocation requests in terms of the GRAM API. GRAM and associated GSI services could then be used to handle authentication, resource allocation, and process creation at each site.

Co-allocation. SF-Express requires that the allocation of the resources used by its simulation nodes and routers, and the starting of the relevant processes, occur simultaneously. A Globus co-allocation service coordinates multiple GRAM requests, constructing an unified job from individual GRAM requests, or subjobs, and prevents application components from starting until all the required resources have been obtained. As shown in Figure 1, the changes to SF-Express required to interface to GRAM and the co-allocation service are minimal.

Fault detection. The nondeterministic nature of the DIS simulation algorithm means that a simulation can sometimes continue in a useful fashion even if a component has failed. For this reason, we use the Globus heartbeat monitor (HBM) to provide fault-detection and notification. This Globus service provides a wide-area mechanism for monitoring the state of the components of a computation, notifying a status monitor of failure.

Remote data access. SF-Express generates two types of file I/O: it reads a variety of databases and configuration files and writes an error log. We use the Global Access to Secondary Storage (GASS) service to simplify access to error logs. Each machine participating in

Table 2: A grid-aware version of SF-Express is being constructed incrementally: Globus services are incorporated one by one to improve functionality and reduce application complexity. The Status field indicates code status as of early 1998: techniques are in use (Y), experimental or partial use (y), or remain to be applied in the future (blank).

Services	How used	Benefits	Status
GRAM, GSI	Start SF-Express on supercomputers	Avoid need to log in to and schedule each system	Y
+ Co-allocator	Distributed startup and management	Avoid application-level check-in and shutdown	Y
+ MDS	Use MDS information to configure computation	Performance, portability	y
+ Resource broker	Use broker to locate appropriate computers	Code reuse, portability	y
+ Nexus	Encode communication as Nexus RSRs	Uniformity of interface, access to unreliable comms	y
+ HBM	Routers check in with application-level monitor	Provide degree of fault tolerance	Y
+ GASS	Use to centralize error logging, access terrain database files, etc.	Avoid need to prestage data files	y
+ GEM	Use to generate and stage executables	Avoid configuration problems	

```
main(int argc, char **argv) {                    /*  1 */
  router_info my_router;                         /*  2 */
  router_info routers[MAX_ROUTERS];              /*  3 */
  /* Wait for rest of nodes to start */
  ca_barrier();                                  /*  5 */
  /* Connect up routers *
  my_router.addr = router_addr();                /*  7 */
  my_router.port = router_port();                /*  8 */
  /* Get locations of other routers */
  sfe_exchange_info(&my_router,routers);         /* 10 */

  /* Start MPI in local set of nodes */
  MPI_Init(argc,argv);                           /* 13 */
  /* Rest of code */
```

Figure 1: The Globus-enhanced SF-Express startup code extends the code used on a single parallel computer with two new calls. The co-allocator library function `ca_barrier` (line 5) is added to ensure that the simulation proceeds only when all resources have been obtained. The function `sfe_exchange_info` (line 10) uses GRAM and co-allocator functions to gather and broadcast the IP addresses and port numbers of all router nodes

SF-Express generates a log file indicating the status of the simulation on that machine. Prior to using Globus, an SF-Express user had to log into the machine running a piece of the simulation to examine the contents of the log. Using GASS, which supports append-mode file writes, we can write logfile entries to a remote location. GASS also supports automatic fetching of files when they are opened, in a similar fashion to UFO [2], and program-controlled prestaging of files. We plan to use this facility to provide access to the read-only files accessed by SF-Express.

3.2 SF-Express Lessons Learned

The Globus-based version of SF-Express was developed with relatively little effort and provided improved functionality relative to the original code. The experience also revealed areas in which existing Globus services could be improved and suggested additional opportunities for use of Globus services in SF-Express.

The primary Globus deficiency revealed by this work relates to the implementation of the co-allocation service. The co-allocation service initially designed for Globus provided a static co-allocation model in which a request failed if any component of the request failed. While this model proved to vastly simplify SF-Express startup, it had the deficiency that a startup problem on any one computer required that we terminate the healthy components of the job and restart the computation. Yet in practice, individual GUSTO components failed frequently. Interestingly, software problems rather than hardware and network failures were the leading cause of difficulty. Examples of failure states that we observed include system paralysis due to generation of a large core file; failure of local scheduling systems; intermittent application crashes (due to bugs in the original SF-Express code); and operator error. These problems were especially troublesome because SF-Express has a startup time of over 15 minutes. Building on this experience, we have designed a more flexible, dynamic co-allocation model in which the contents of a co-allocation request can be modified up until the point the program starts to execute.

Experience suggested three additional areas in which Globus components could be used in an SF-Express implementation: configuration (using MDS information for autoconfiguration, hence improving portability and performance); resource brokering (providing an SF-Express-specific resource broker, hence reducing the need for human involvement in the resource selection process); and communication (using Globus communication services to access multicast and quality of service mechanisms, hence improving scalability and simulation performance. Space does not permit further discussion of these issues in this extended abstract.

4 CAVERN

The second application that we consider is CAVERNsoft [18], a software infrastructure designed to support the rapid development of tele-immersive applications. In tele-immersion, immersive virtual reality environments are used over networks to provide shared access to simulated virtual spaces for design, collaboration, entertainment, education, etc. [11] The producers and consumers of the virtual environment as well as the datasets and simulations on which the virtual space is based are frequently geographically distributed, placing heavy demands on distributed computing support.

CAVERNsoft supports tele-immersive application development by providing runtime support for the definition, update, and access of shared virtual worlds. Its layered architecture

has at its core an Information Resource Broker (IRB) that supports the maintenance of shared databases, and above this libraries for the manipulation of avatars and manipulation of audio and video streams.

4.1 CAVERNsoft Experiences

The initial version of CAVERNsoft makes extensive use of the Globus toolkit's communication service, and hence we focus our discussion on this aspect of the system. We also note opportunities that we have identified for the use of other services.

Communication in tele-immersive applications is complicated by the variety of flows that need to be handled. DeFanti and Stevens [11] identify nine distinct types of flow (control, text, audio, video, tracking, database, simulation, haptics, rendering), each with distinctive requirements in terms of both performance and the mechanisms that can be used to implement the flows. For example, tracking information need not be propagated reliabily but can almost always benefit from multicast, while database updates require reliable communication but cannot always use multicast capabilities.

Historically, tele-immersion systems (and other similar applications) have either used a single low-level communication protocol for all flows (e.g., TCP/IP [24], ISIS [4, 7]), or have used a mixture of different, often specialized APIs for different flows [20, 23, 19]. Neither approach is ideal. We believe that a better approach is to code to a single API that allows both high-level specification of communication structure and independent specification of the mechanisms used to achieve that communication. Nexus, the communication component of the Globus toolkit, meets this requirement.

The Nexus communication library allows applications to define communication links over which can be performed asynchronous remote procedure calls called remote service requests (RSRs). Associated operations allow us to select the underlying communication protocol used for a particular RSR according to when, where, and what is being communicated [13]. This means that if two components of a CAVERNsoft application are located on different nodes of a parallel computer, Nexus operations can be mapped onto efficient local communication methods, such as MPI; if the components are located on different computers, Nexus communication operations can be mapped into unreliable, wide-area communication protocols. More importantly, this flexibility means that CAVERNsoft can specify all communication operations in terms of a single abstraction (and API) and then vary the method used according to the type of flow that the communication is associated with. For example, tracking events can be performed with an unreliable multicast protocol, while database updates are propagated with reliable unicast or multicast.

Nexus also allows quality of service (QoS) specifications to be associated with communication links. These specifications can then be translated into a RSVP [5] or similar [9] reservations if the underlying network supports this capability. MDS information can be used to determine the capabilities and utilization of the underlying networks, and hence to evaluate trade offs between different protocols.

4.2 CAVERNsoft Lessons Learned

CAVERNsoft is a second-generation tele-immersion system, designed in response to lessons learned with an earlier prototype [23]. A comparison of these two systems shows that the use of Nexus in CAVERNsoft reduces the complexity of the communication code, while also increasing

its portability. A single clean abstraction and API is used for all communication operations, regardless of type and application-specific details, and hence of the actual mechanisms used to implement the flow on a particular computer. This simplicity and portability are precisely the results we aimed to achieve with the Globus toolkit.

Having completed construction of this initial CAVERNsoft prototype, we are considering a number of extensions that will involve the use of additional Globus toolkit components. A first step is to use Globus security infrastructure and resource management mechanisms to handle authentication and resource allocation on distributed resources; currently, these tasks are handled in a rather ad-hoc manner. A next task will be to use MDS to guide optimized configuration decisions for the IRB implementation. We also anticipate that the Globus instrumentation service will be of use.

5 Conclusions

We have used the Globus toolkit to implement a variety of distributed computing applications, two of which we have described here. Each application typically uses a different set of grid services. Nevertheless, all have in common that an existing application code or application structure was modified for grid execution fairly easily by introducing appropriate components chosen from the Globus "bag of services." This means that the application did not have to be entirely rewritten before it could operate in a grid environment: services could be introduced into an application incrementally, with functionality increasing at each step. In this respect, we believe that our initial experiments with the toolkit have been a success and suggest that the approach should be pushed further.

Our experiments also teach us lessons about the grid environment, most notably the importance of fault tolerance. While detecting and dealing with failure are known to be critical issues in distributed systems, we have been astonished by the range of error conditions that we have encountered. Fortunately, we find that relatively simple techniques can render applications signicantly more robust. In this respect, the integrated, network accessible information service provided by Globus (MDS) proved to be valuable as a mechanism for detecting and recovering from failure. MDS information allowed us develop a range of general and application-specific high-level tools such as resource brokers and status monitors.

Future work will focus on further refining the current Globus services by studying their use in additional applications. We are continuing to work with both SF-Express and CAVERNsoft to make them more *grid-aware*. We are also working to extend the Globus toolkit to incorporate additional services, notably in the area of executable management. Our goal is to make these and other applications robust and simple enough so that the use of computational grids becomes commonplace.

Acknowledgments

We gratefully acknowledge the contributions of other members of the Globus team: in particular, Joe Bester, Joe Insley, Nick Karonis, Gregor von Laszewski, Stuart Martin, Warren Smith, and Brian Toonen at Argonne National Laboratory; Steve Fitzgerald and Mei-Hui Su at USC/ISI; and Craig Lee and Paul Stelling at The Aerospace Corporation.

This work was supported in part 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; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; and by the National Science Foundation.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.

[2] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. Extending the operating system at the user level: The UFO global file system. In *1997 Annual Technical Conference on UNIX and Advanced Computing Systems (USENIX'97)*, January 1997.

[3] 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.

[4] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[5] R. Braden, L. Zhang, D. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 functional specification. Internet Draft, 1996.

[6] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing distributed synthetic forces simulations in metacomputing environments. In *Proceedings of the Heterogeneous Computing Workshop*, 1998. to appear.

[7] C. Carlsson and O. Hagsand. DIVE - a multi-user virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, 1993.

[8] S. Chapin. Distributed scheduling support in the presence of autonomy. In *Proc. Heterogeneous Computing Workshop*, pages 22–29, 1995.

[9] Kay Connelly and Andrew A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In *Proceedings of SC'97*, November 1997.

[10] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[11] T. DeFanti and R. Stevens. Tele-immersin. In I. Foster and C. Kesselman, editors, *Computational Grids: The Future of High-Performance Distributed Computing*. Morgan Kaufmann Publishers, 1998.

[12] 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*, pages 365–375. IEEE Computer Society Press, 1997.

[13] 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*, 40:35–48, 1997.

[14] I. Foster and C. Kesselman. The Globus project: A progress report. In *Proceedings of the Heterogeneous Computing Workshop*, 1998. to appear.

[15] Geoffrey Fox and Wojtek Furmanski. Petaops and exaops: Supercomputing on the Web. *IEEE Internet Computing*, 1(2):38–46, 1997.

[16] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[17] Object Management Group and X/Open. Common object request broker: Architecture and specification, 1991.

[18] Jason Leigh, Andrew Johnson, and Thomas A. DeFanti. CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications*, 2(2):217–237, December 1997.

[19] Michael R. Macedonia and Michael J. Zyda. A taxonomy for networked virtual environments. In *Proceedings of the 1995 Workshop on Networked Realities*, Oct 1995.

[20] J. Mandeville, J. Furness, and T. Kawahata. Greenspace: Creating a distributed virtual environment for global applications. In *Proceedings of IEEE Networked Virtual Reality Workshop*. IEEE, 1995.

[21] P. Messina, S. Brunett, D. Davis, T. Gottschalk, D. Curkendall, L. Ekroot, and H. Siegel. Distributed interactive simulation for synthetic forces. In *Proceedings of the 11th International Parallel Processing Symposium*, 1997.

[22] B. C. Neuman and S. Rao. The Prospero resource manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice & Experience*, 6(4):339–355, 1994.

[23] Maria Roussos, Andrew Johnson, Jason Leigh, Christina Vasilakis, and Thomas G. Moher. Constructing collaborative stories within virtual learning landscapes. In *Proceedings of the European Conference on A.I. in Education*, pages 129–135, Sept 1996.

[24] Chris Shaw and Mark Green. The MR toolkit peers package and environment. In *Proceedings of the Virtual Reality Annual International Symposium. VRAIS'93*. IEEE Computer, 1993.

[25] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. WebOS: Operating system services for wide area applications. Technical Report UCB CSD-97-938, U.C. Berkeley, 1997.

[26] J. Weissman and A. Grimshaw. A federated model for scheduling in wide-area systems. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, 1996.