

A System for Monitoring and Management of Computational Grids

Warren Smith
Computer Sciences Corporation
NASA Ames Research Center
wwsmith@nas.nasa.gov

Abstract

As organizations begin to deploy large computational grids, it has become apparent that systems for observation and control of the resources, services, and applications that make up such grids are needed. Administrators must observe resources and services to ensure that they are operating correctly and must control resources and services to ensure that their operation meets the needs of users. Users are also interested in the operation of resources and services so that they can choose the most appropriate ones to use. In this paper we describe a prototype system to monitor and manage computational grids and describe the general software framework for control and observation in distributed environments that it is based on.

1. Introduction

A recent trend in government and academic research is the development and deployment of computational grids [14, 22]. Computational grids are large-scale distributed systems that typically consist of high-performance compute, storage, and networking resources. Examples of such computational grids are the DOE Science Grid [3], the NSF Partnerships for Advanced Computing Infrastructure [6, 7], and the NASA Information Power Grid [29]. Most of the work to deploy these grids is in developing the software services to allow users to execute applications on large and diverse sets of distributed resources. These services include security, execution of remote applications, managing remote data, access to information about resources and services, and so on. There are several toolkits that provide these services, such as Globus [21], Legion [26], and Condor [30].

NASA is building a computational grid called the Information Power Grid (IPG) that is based upon the Globus toolkit. The IPG currently consists of resources and users at four NASA centers and our attempt to deploy a production grid of this size has highlighted the need for systems to observe and control the resources, services, and applications that make up such grids. We have found it difficult to ensure that the many resources in the IPG and the grid services executing on those resources are

performing correctly. We have also found it cumbersome to perform administrative tasks such as adding grid users to our resources. These observations have led to our development of a system to address these needs.

This paper provides an overview of our system for monitoring and managing a computational grid. It allows administrators to observe the status of the resources and services that make up a Globus-based computational grid, to perform actions to correct failures, and perform day-to-day administrative functions. This system is constructed using the CODE toolkit [35] that provides a secure, scalable, and extensible framework for making observations on remote computer systems, transmitting this observational data to where it is needed, performing actions on remote computer systems, and analyzing observational data to determine what actions should be taken. We begin our discussion with an overview of the CODE framework. Section 3 describes the current functionality of our grid monitoring and management system. Section 4 describes related work and Section 5 summarizes our work and presents future work.

2. CODE Framework

We have developed a software framework for Control and Observation in Distributed Environments, called CODE for obvious reasons [35]. We are using this framework to implement several useful grid services, including our grid monitoring and management system. This section provides an overview of the framework.

2.1. Architecture

We call CODE a framework because it contains the core code that is necessary for performing monitoring and management. Users only need to add components to this framework and start the framework running. For example, if a user wants to create a host monitor, she would create components to monitor processes, files, network communications, and so on. The user would then add these components to the framework and tell the framework to begin monitoring the host. This same process is used for adding components to perform management actions. In fact, the typical process will be

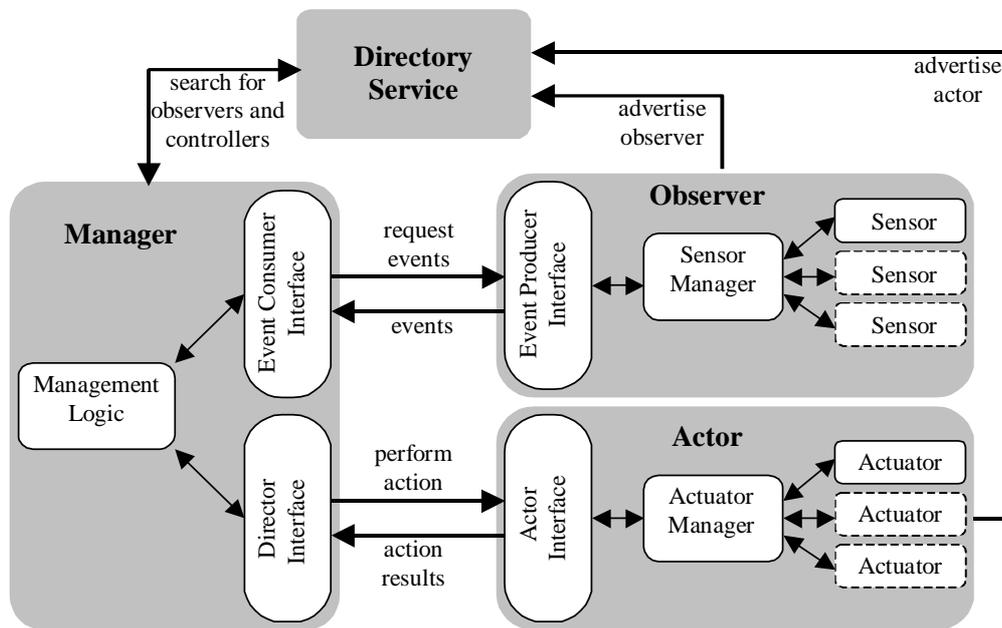


Figure 1. Architecture of the CODE framework.

easier because CODE provides a set of commonly used components for observing various properties and performing various actions and all a user will have to do is select which of these components to use.

The CODE architecture is shown in Figure 1. The components that are shown with a solid outline are those that are supplied by our framework, the components that are shown with a dashed outline are provided by the user, and the gray boxes show the logical grouping of the components in our framework into entities that may be on different hosts. The logical components of our framework are *observers* that perform observations and report observations, *actors* that perform actions, *managers* that receive observations, make decisions, and request actions, and a *directory service* for locating observers and actors.

An observer is a process on a computer system that provides information that can be measured from the system it is executing on. This could be information about the computer system, services or applications running on that computer system, or information that is not related to the computer system but that is accessible from it. Examples of this last type of information are scheduling queue information from a front-end system and the current use of a local area network. An observer provides information in the form of *events*. An event has a type and contains data in the form of <name, value> pairs. The values are typically of simple types such as string or integers, but can also be structures. An observer allows a manager to query for an event or to subscribe for a set of events. A subscription is useful, for example, if a user wants to be notified of the load on a system periodically or notified whenever some fault condition occurs. Access

to events is controlled based on user identity and user location on both a per-observer and a per-event type basis.

An observer consists of the following components:

- **Sensor.** A sensor is used to sense or measure some property. For example, a CPU load sensor would measure the CPU load of a host. A sensor is a passive component that performs measurements only when the sensor manager requests them. We are providing a set of sensors as part of our framework, but users will most likely need to implement sensors for their specific purposes.
- **Sensor Manager.** The sensor manager receives event requests or subscriptions from the event producer interface, uses the appropriate time to perform a measurement, and sends the result of the measurement to the event producer interface in the form of an event.
- **Event Producer Interface.** The event producer interface provides an interface for observers to access a distributed event service. This event service allows event subscriptions to be established between producers and consumers, allows consumers to query for events from producers, and allows producers to send events to consumers.

An actor is a process on a computer system that can be asked to perform actions. These actions are made from the actor process and could affect local or remote resources, services, and applications. Access to actions is controlled based on user identity and user location on both a per-actor and a per-action type basis. An actor consists of the following components:

- **Actuator.** An actuator is a component that can be used to perform a specific action. For example, an actuator can be used to start a daemon. An actuator is a passive component that performs actions only when the actuator manager requests them. We are providing a set of actuators as part of our framework, but users will most likely need to implement actuators for their own specific purposes.
- **Actuator Manager.** The actuator manager receives requests to perform actions from the actor interface, uses the appropriate actuator to perform the action, and sends the results of the action back to the actor interface.
- **Actor Interface.** The actor interface provides an interface to a distributed action service that transmits requests for actions and their results.

A manager is a process that asks observers for information, reasons upon that information, and asks actors to take actions when the observations indicate that actions need to be taken. A manager consists of the following components:

- **Management Logic.** The management logic receives events from the event consumer interface, reasons upon this information to determine if any actions need to be taken, and then takes any actions using the director interface. There are two ways to implement the management logic:
 - Write C++ or Java code that contains a series of if and case statements, a state machine, or whatever code is needed to decide what actions to perform.
 - Use an expert system and write management rules. We are experimenting with using the CLIPS expert system [25] to simplify the writing of managers.
- **Event Consumer Interface.** The event consumer interface is used to request events from observers and receive those events.
- **Director Interface.** The director interface is used to request that actors perform actions and to receive the results of those actions.

A common component of a computational grid is a directory service or grid information service [20]. For our purposes, a directory service is a distributed database that is accessed using the Lightweight Directory Access Protocol (LDAP) [28]. We use a directory service to store the locations of observers and actors, describe what types of observations and actions they provide, and allow managers to search for the observers and actors.

2.2. Implementation

We have implemented the CODE framework in C++ and in Java so that it can be used from a variety of programming languages. At this point, the CODE

framework supports communication using TCP, UDP, and SSL. The SSL communication is implemented using the Globus Grid Security Infrastructure [23]. The CODE framework is an implementation of the Grid Monitoring Architecture [40] defined in the Global Grid Forum and supports encoding of communication messages with extension of the event protocol [37, 38] that is being defined in the Global Grid Forum. This protocol encodes data using the eXtensible Markup Language (XML) and CODE uses the Xerces XML parsers to decode messages. Further, the format of the data CODE places in the directory service is compatible with the LDAP schemas [36] being defined in the Global Grid Forum.

3. Grid Management System

As computational grids grow, it becomes very difficult to ensure the correct operation of the large number of resources and services that make up a grid and to configure the services that are available on a grid. We have developed a prototype Grid Management System (GMS) to assist with these tasks in a Globus-based grid such as the NASA Information Power Grid. Figure 2 shows the high-level architecture of this system and we will describe the components of this architecture next.

3.1. GRAM Management Agent

The Globus toolkit includes a service called the Globus Resource Allocation Manager (GRAM) [17] that allows remote users to execute applications on a computer system. Our system has an agent on each host that has a GRAM server. The purpose of this agent is to ensure that the GRAM service is available to users, that the computer system it is associated with is operating correctly, and that there is network connectivity to other GRAM hosts. The GRAM management agent contains a CODE observer, actor, and manager. The observer is used to monitor the following properties:

1. The network latency between the GRAM host and other GRAM hosts. These latencies are used in this situation to detect any network problems. The ping sensor measures round trip times using the Unix ping command.
2. The available network bandwidth between the GRAM host and other GRAM hosts. These bandwidth measurements are also used to detect network problems and help users select resources. The IPerf [42] network measurement tool is used to make these measurements.
3. The CPU load is measured to determine if the computer system is overloaded and unusable. This measurement is made using three different sensors. One sensor uses the Unix uptime program, a second uses the PBS qstat command, and the third uses the

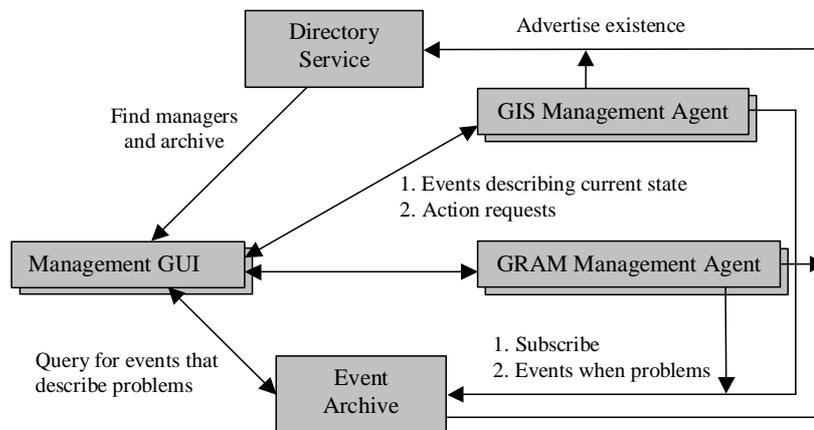


Figure 2. Architecture of our grid monitoring and management system.

LSF bjobs command. The sensor that is used depends on how access to the computer system is scheduled.

4. The memory statistics are measured using the Unix vmstat command, or similar commands, to determine if the memory subsystem is overloaded.
5. The available disk space is measured using the Unix df command. The GRAM servers require some minimal amount of disk space to operate.
6. The status of the GRAM reporter. The IPG is currently running the Globus MDS in classic mode. In that mode, a GRAM reporter daemon is executing on each GRAM host and writing data into a remote LDAP server.
7. The GRAM log files. These log files contain information about usage of the GRAM service and information about any problems that occur. These log files are observed for any problems.
8. The GRAM grid map file. This file specifies which grid users can execute applications through the GRAM service and also maps grid user identifiers to local Unix user identifiers. This information is provided so that remote administrators can determine and modify which grid users can use the GRAM service.

The actor has actuators to perform the following actions:

1. Start and stop the Iperf server. An Iperf server is needed so that Iperf clients can connect and perform Iperf experiments.
2. Send email. The email actuator is used to send email to administrators when a problem cannot be handled automatically, but must be corrected immediately.
3. Modify the GRAM gridmap file. This actuator is used by the remote management GUI so that access to the GRAM service on the host can be given to or taken away from grid users and the mapping of grid users to local user identities can be modified

4. Start, stop, or restart the GRAM reporter. If the GRAM reporter is not running or is not responding it can be stopped or started.

The GRAM management agent also includes a CODE manager. At this time, this manager does not receive any observations nor perform any actions. This approach assumes that the management of a grid takes place in the management GUIs. In the future, this manager will receive observations and perform actions so that management functionality will be offloaded to the GRAM manager and that the system will be more scalable.

When this agent begins executing, it locates the event archive (described further in Section 3.3) using the directory service and initiates subscriptions with the archive as the producer of events. These subscriptions indicate that the GRAM management agent will send events to the archive when problems occur. These problems include excessive CPU or memory use, failure of the GRAM reporter, or problems in the GRAM log files. At any time, management GUIs can contact this agent to receive information or request that actions be performed.

3.2. GIS Management Agent

A Globus-based computational grid also has a distributed database, called a Grid Information Service (GIS) that contains information about the resources, users, services, and applications that are part of the computational grid. The Globus GIS is called the Metacomputing Directory Service [16]. A GIS typically consists of multiple servers running on multiple hosts. Our architecture includes an agent to monitor the operation of GIS servers, to determine if there are any problems, and to take actions to attempt to address any problems. Similarly to the GRAM agent, this agent provides observational data to remote agents and allows authorized remote agents to request actions. The GIS management agent for each

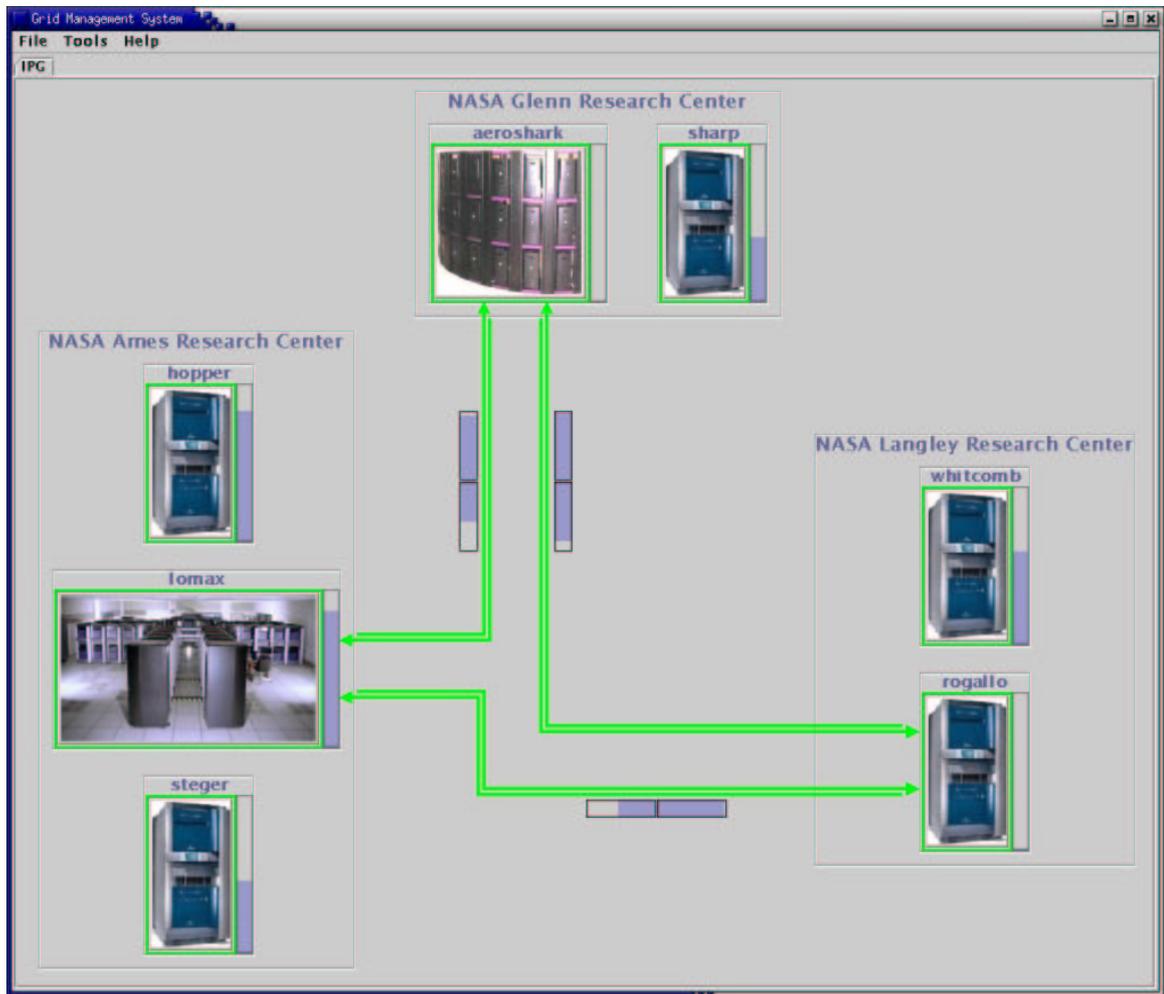


Figure 3. Management GUI displaying the status of a subset of the resources on the NASA IPG.

GIS server consists of an observer, an actor, and a manager. The observer monitors the following properties:

1. The network connectivity between the GIS hosts. LDAP servers typically refer searches for information to other LDAP servers.
2. The CPU load of the host.
3. The available memory of the host.
4. The available disk space.
5. The status of the LDAP server itself. This is measured in two ways. First, the existence of the LDAP server process is observed. Second, the time to request a search and receive a reply is measured.

The actor that is part of a GIS management agent is relatively simple: It only has two actuators at the current time. One actuator is used to send emails. The other actuator is used to start, stop, and restart the LDAP server. The GIS management agent also includes a CODE manager. At this time, this manager does not receive any observations nor perform any actions.

When the GIS management agent begins executing, it locates the event archive using the directory service and initiates subscriptions with the archive as the producer of events. These subscriptions indicate that the GIS management agent will send events to the archive when problems occur.

3.3. Event Archive

The event archive stores events so that management GUIs can obtain information about problems that have occurred in the past. The archive acts as an event consumer for events generated by GRAM and GIS management agents and acts as a producer of events for management GUIs. GRAM and GIS management agents use the directory service to find the archive and then they initiate a subscription to the archive. The agents then send events to the archive whenever problems occur. Management GUIs also find the archive using the directory service. They then query for events from the

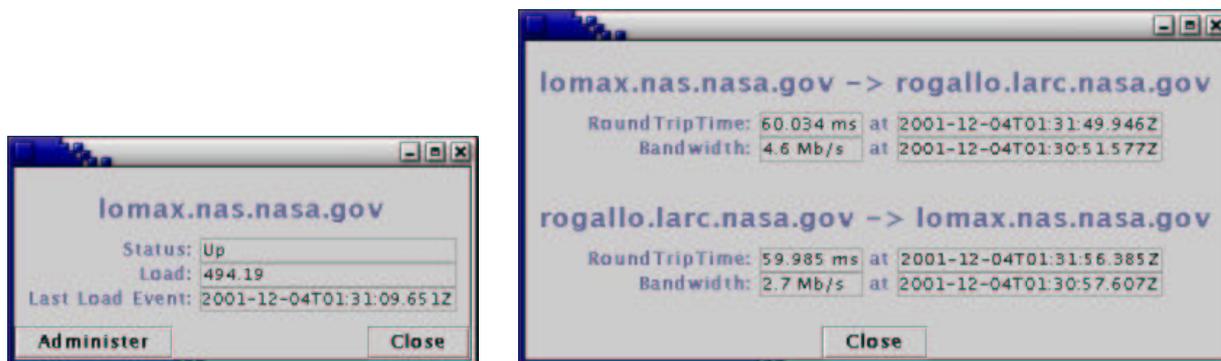


Figure 4. Detailed information about the load on the SGI Origin lomax.nas.nasa.gov and about the connection between lomax.nas.nasa.gov (in California) and rogallo.larc.nasa.gov (in Virginia).

archive. This query contains an event filter that is used to select which events to return. At the current time, we are using the Xpath [15] language as our filter language and the Xindice [12] XML database to store our events.

3.4. Directory Service

As described in Section 2.1, the observers and actors on the GRAM and GIS hosts register themselves in the directory service so that managers can find them. The event archive also registers itself so that management GUIs, GRAM managers, and GIS managers can find it.

3.5. Management GUI

The final component of our grid monitoring and management system is a graphical user interface that is used by grid administrators. Instances of this interface can be started and stopped at any time by multiple administrators. This interface allows grid administrators to view the current status of a grid, be notified when problems occur on the grid, examine problems that have occurred in the past, and perform grid administrative tasks. Figure 3 shows the management GUI being used to show the status of a subset of the resources on the NASA IPG. While the colors can't be seen here, the boxes around each computer system icon indicate whether status information from the machine has been received recently. The boxes are colored green when information has been received recently, yellow when an expected event has not been received, and red when two expected events have not been received. The vertical progress bars next to each computer system icon show what fraction of the CPUs in that computer system is being used. The lines between computer systems indicate whether the machines can ping each other. They are also colored green, yellow, and red to indicate if pings have been successful. The progress bars next to the lines show the fraction of maximum measured bandwidth, in both directions, that was available during the last bandwidth experiment. The computers and

networks to monitor along with icon selection and placement of all of the graphical components are stored in a configuration file that is loaded when the GUI starts.

As you can see from the figure, a user of the management GUI can quickly understand the status of some of the major IPG systems and the networks that connect them. Users can also click on any of the machines or network connections to display more detailed information such as that shown in Figure 4.

This interface can also be used to perform administrative tasks. At the current time, the interface allows an administrator to add, remove, and modify users in the GRAM grid map files on the remote computer systems. This interface provides several different ways to view user access. One display shows the grid user to local user name mappings for a single computer system and allows modifications to these mappings. Another display shows all of the computers that a grid user has access to and all of the local user names the grid user maps to. An administrator can use this display to add or remove access to computer systems and specify which local user name a grid user should map to.

4. Related Work

There are many existing systems for remote monitoring and management of networks and computer systems. A few commercial systems are OpenView [4] from Hewlett Packard, ManagementCenter [10] from Sun, Works2000 [2] from Cisco, Unicenter Networks and Systems Management [11] from Computer Associates, Tivoli Enterprise Console [5] and related products from IBM, PATROL [8] from BMC Software, and SiteAssure [9] from Platform Computing. These systems typically provide a wide range of monitoring and management services for a variety of resources. There are several problems with using these tools in our current grid environments. First, these products do have a cost associated with them, which may be difficult to afford for all of the participants in a multi-institution research grid.

Other problems are the lack of standards and compatibility between products and the lack of portability because of the unavailability of source code. Further, such tools do not support the grid security infrastructure. There are other systems that are free for noncommercial use or open source, but these tools tend to lack the functionality of the commercial tools previously mentioned and would need to be extended to manage grid services. Examples of such tools are NetSaint [24] and Big Brother [1]. Many of these types of tools are based on the Simple Network Management Protocol [39] that can provide information on networking and other types of resources, services, and so on.

Another area of related work is distributed event services, one of the core components of our framework. There has also been a large amount of work in this area. CORBA has defined an event service [31] and a notification service [32]. The problem with these services is that they are part of CORBA, which is not commonly used in grid computing. There Java Messaging Service [27] has support for distributed event services, but this only supports the Java language. There are also research projects to develop distributed event services such as Sienna [13], Elvin [33], Echo [18], OIF [19], and XEvents [34] and research projects to perform various types of monitoring such as NWS [43], JAMM [41], and MDS [16]. Many of these services are quite usable, the main advantage of the one that is part of our framework is that it will continue to be compatible with the standards defined in the Grid Forum. The benefit of this is that each implementation of an event service or monitoring system will have positives and negatives in terms of programming language, performance, and usability. Standards allow users to select the best implementations for their needs and still communicate with other implementations that are optimized for different purposes.

5. Summary and Future Work

Our efforts to deploy a computational grid at NASA have demonstrated the need for tools to observe and control the resources, services, and applications that make up grids. This has led to our development of CODE which provides a secure, scalable, and extensible framework for making observations from remote computer systems, transmitting this observational data to where it is needed, performing actions from remote computer systems, and analyzing observational data to determine what actions should be taken.

A prototype of our framework is complete and we are continuing to improve it. In addition to the core framework, we have implemented sensors for measuring various properties such as process status, file characteristics, disk space, CPU load, network interface characteristics, and LDAP search performance. We have

also implemented a few simple actuators. We have used this framework to develop a prototype grid management system that allows administrators to observe the current status of the resources and services that make up a grid, to correct problems when they appear, and to perform administrative tasks such as modifying which grid users can access which computer systems. The grid management system also records failures that occur so that they can be examined at a later time.

In the future we will continue to improve and extend the CODE framework and we will also improve our Grid Management System as we gain experience from its use. Further, we will track the standards for grid event services that are being developed in the Global Grid Forum and will strive to be compatible with those standards.

Acknowledgments

We gratefully acknowledge the help of Abdul Waheed who participated in the early phases of this project and of Jerry Yan who provided the initial motivation. We also wish to thank Dan Gunter, Ruth Aydt, Brian Tierney, Dennis Gannon, and Valerie Taylor for the many useful discussions we have had related to this work both inside and outside of the Grid Forum. This work has been supported by the NASA HPCC and CICT programs.

References

- [1] "Big Brother," <http://bb4.com/>.
- [2] "CiscoWorks2000," http://www.cisco.com/warp/public/44/jump/ciscowork_s.shtml.
- [3] "The DOE Science Grid," <http://www-itg.lbl.gov/Grid>.
- [4] "HP OpenView," <http://www.openview.hp.com>.
- [5] "IBM Tivoli Enterprise Console," <http://www.tivoli.com/products/index/enterprise-console/>.
- [6] "The National Computational Science Alliance," <http://www.ncsa.uiuc.edu/access/index.alliance.html>.
- [7] "The National Partnership for Advanced Computing Infrastructure," <http://www.npaci.edu/>.
- [8] "PATROL Console," <http://www.bmc.com/products/>.
- [9] "Platform SiteAssure," <http://www.platform.com/products/rm/SiteAssure/index.asp>.
- [10] "Sun Management Center," <http://www.sun.com/software/solaris/sunmanagement-center/>.
- [11] "Unicenter Network and Systems Management," <http://www3.ca.com/solutions/product.asp?id=2869>.
- [12] "Xindice," <http://xml.apache.org/xindice>.
- [13] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service." In Proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing, Portland, OR, 2000.

- [14] C. Catlett and L. Smarr, "Metacomputing," in *Communications of the ACM*, vol. 35, 1992, pp. 44-52.
- [15] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0," World Wide Web Consortium November 16 1999.
- [16] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing." In Proceedings of the The 10th IEEE International Symposium on High Performance Distributed Computing, 2001.
- [17] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metasystems," *Lecture Notes on Computer Science*, vol. 1459, 1998.
- [18] G. Eisenhauer, F. Bustamante, and K. Schwan, "Event Services for High Performance Computing." In Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, 2000.
- [19] R. E. Filman and D. D. Lee, "Managing Distributed Systems with Smart Subscriptions." In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 2000.
- [20] S. Fitzgerald, I. Foster, C. Kesselman, G. v. Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations." In Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, 1997.
- [21] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, pp. 115-128, 1997.
- [22] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure,".: Morgan Kaufmann, 1999.
- [23] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids." In Proceedings of the 5th ACM Conference on Computer and Communications Security, 1998.
- [24] E. Galstad, "NetSaint Network Monitor," <http://www.netsaint.org/>.
- [25] J. Giarratano, *Expert Systems: Principles and Programming*: Brooks and Cole Publishing, 1998.
- [26] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. R. Jr., "Legion: The Next Logical Step Toward A Nationwide Virtual Computer," Department of Computer Science, University of Virginia CS-94-21, June, 1994 1994.
- [27] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java Message Service Specification v1.1," Sun Microsystems June 2002.
- [28] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*: Macmillan Technical Publishing, 1997.
- [29] W. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid." In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999.
- [30] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System." In Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.
- [31] OMG, "Event Service Specification v1.1," 2001-03-01, March 2001.
- [32] OMG, "Notification Service Specification v1.0," 2000-06-20, June 2000.
- [33] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content Based Routing with Elvin4." In Proceedings of the AUUG2k, Canberra, Australia, 2000.
- [34] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley, "SoapRMI Events: Design and Implementation," Computer Science Department, Indiana University TR549, May 2001.
- [35] W. Smith, "A Framework for Control and Observation in Distributed Environments," NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA NAS-01-006, June 2001.
- [36] W. Smith and D. Gunter, "Simple LDAP Schemas for Grid Monitoring," The Global Grid Forum GWD-Perf-13-1, 2001.
- [37] W. Smith, D. Gunter, and D. Quesnel, "A Simple XML Producer-Consumer Protocol," The Global Grid Forum GWD-Perf-8-2, 2001.
- [38] W. Smith, D. Gunter, and D. Quesnel, "An XML-Based Protocol for Distributed Event Services." In Proceedings of the The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 2001.
- [39] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards*. Reading, Massachusetts: Addison-Wesley, 1993.
- [40] B. Tierney, R. Aydt, D. Gunter, W. Smith, Valerie Taylor, R. Wolski, and M. Swany, "A Grid Monitoring Service Architecture," Global Grid Forum Performance Working Group 2001.
- [41] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson, "A Monitoring Sensor Management System for Grid Environments." In Proceedings of the 9th IEEE Symposium on High Performance Distributed Computing, Pittsburgh, PA, 2000.
- [42] A. Tirumala and J. Ferguson, "Iperf Version 1.2," <http://dast.nlanr.net/Projects/Iperf/>.
- [43] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service." In Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing, 1997.