A System for Monitoring and Management of Computational Grids

Warren Smith

Computer Sciences Corporation

NASA Ames Research Center

Mail Stop T27A-2

Moffett Field, CA 94035

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 the operation of resources and services to ensure that they are operating correctly and they must control the 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.

Key words: monitoring, management, event service, computational grids, distributed computing

ICPP Areas: Network-Based and Cluster Computing,

OS and Resource Management

1. Introduction

A recent trend in government and academic research is the development and deployment of computational grids [12, 18]. 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 [8, 9], and the NASA Information Power Grid [7, 23]. 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 [4, 17], Legion [6, 20], and Condor [2, 24].

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 [27] 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 [27]. 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 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 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. The next subsections describe these components in more detail.

2.1.1. Observer

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 that computer system. 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 a single 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.

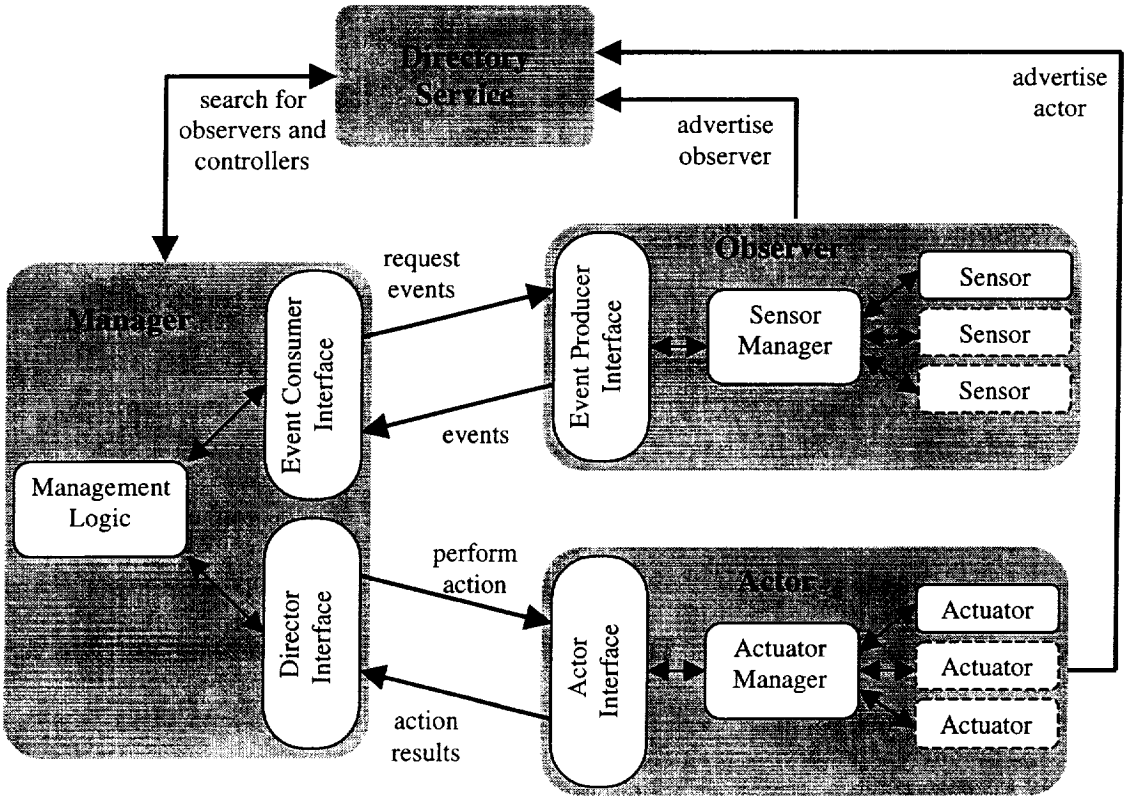

Figure 1. Architecture of the CODE framework.

An observer consists of the following components:

- **Sensor**. A sensor is a component that is used to sense or measure some property. For example, a CPU load sensor would measure the CPU load of a host, a network bandwidth sensor would measure the available bandwidth between two hosts, or a convergence sensor would measure the convergence rate of an application. 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 observer server interface, uses the appropriate sensor at 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.

2.1.2. Actor

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, submit a job to a scheduler, or change a variable in an application 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. The purpose of the distributed action service is to allow a director to request that an actor perform an action, and then transmit the results of the action back to the director.

2.1.3. Manager

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 an expert system to simplify the writing of managers. Without an expert system, the user must write a (potentially large) series of conditional statements to examine events, determine what they mean, and perform the appropriate actions. With an expert system, a user defines a (hopefully smaller) set of rules and the expert system uses these rules to reason on events and perform actions. We have incorporated the CLIPS expert system [1,

19] into our framework for this purpose. The management rules are written by the user and tell the expert system how to operate.

- **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.

2.1.4. Directory Service

A common component of computational grids is a directory service or grid information service [16, 31]. For our purposes, a directory service is searchable distributed database that is accessed using the Lightweight Directory Access Protocol (LDAP) [21, 22]. 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 that provide the information or actions they are interested in. The LDAP entries for our observers follow the schemas under development in the Grid Forum [28].

2.2. Implementation

We have implemented the CODE framework in C++ and in Java so that we can take advantage of the features provided by object-oriented languages when writing modular and extensible code. The C++ implementation allows users to use CODE from C and C++ codes while the Java implementation can be easily used by graphical user interfaces and web portals that are written in Java. In the future we may wrap our C++ code so that it can be called from scientific applications written in Fortran.

One of the main design goals of our code is modularity so that the code can easily be extended and modified. For example, the definition of Sensor and Actuator interfaces allows us to easily implement a variety of sensors and actuators while the sensor manager and actuator manager components simply understand the Sensor and Actuator interfaces but can manage any type of sensor or actuator. Similarly,

the event consumer, event producer, actor, and director interfaces in Figure 1 use Transport and Encoder C++ or Java interfaces for transmitting and encoding messages. This allows us to hide the implementation of various techniques for transmitting and encoding data. At this point, the CODE framework supports communication using TCP, UDP, and SSL. The SSL interface uses OpenSSL [10] and is compatible with the certificates uses with the Globus toolkit. The CODE framework supports encoding of communication messages with extension of the event protocol [29, 30] that is being defined in the Grid Forum Performance Working Group [5]. This protocol encodes data using the eXtensible Markup Language (XML) [14] 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 [28] being defined in the Grid Forum Performance Working Group.

As we mentioned previously, we are using the CLIPS expert system [19] in this project and we are initially targeting the Linux, Solaris, and IRIX operating systems. We expect to port our code to other flavors of Unix and to the Cygwin Unix-like environment for Microsoft Windows. The framework is implemented in a multi-threaded manner using pthreads.

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 3 shows the high-level architecture of this system.

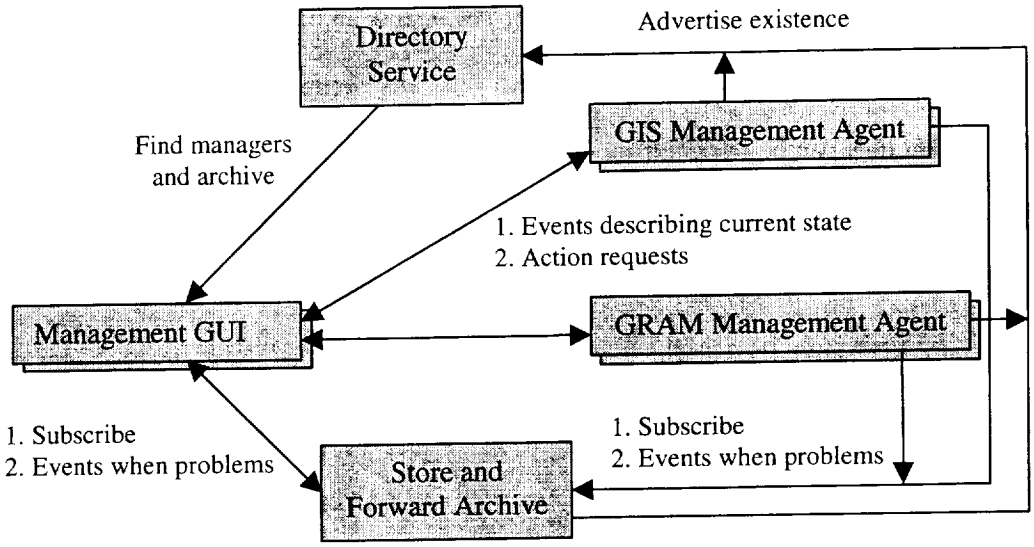

Figure 2. Architecture of our grid monitoring and management system.

This architecture shows the five major components of our system. The Globus toolkit includes a service called the Globus Resource Allocation Manager (GRAM) that allows remote users to execute applications on a computer system. Our monitoring and management architecture has an agent on each host that has a GRAM server. This agent observes the operation of the computer system it is executing on, observes the operation of the GRAM server, determines if there are any problems that should be managed, and takes action to manage these problems. This agent also provides observational data to remote agents and allows authorized remote agents to request actions.

A Globus-based computational grid also has a Grid Information Service (GIS) that is a distributed database, typically accessed using the Lightweight Directory Access Protocol, that contains information about the resources, users, services, and applications that are part of the computational grid. A GIS typically consists of multiple servers running on multiple hosts. Our architecture includes an agent to monitor the operation of the GIS hosts and the GIS agents, to determine if there are any problems with these hosts and agents, 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 third component is a store and forward archive. The agents on the GIS and GRAM hosts send events to this archive when there are problems. The archive forwards events to any interested manager and also stores them so that they can be retrieved later. The fourth component is a directory service that is used by the GRAM management agents, GIS management agents, and the archive to advertise their existence and how to contact them. The final component in our architecture is a transient management GUI that allows a grid administrator to observe the current state of the grid and be notified when any problems occur. Many of these GUIs can be active at any time. We will describe each of these components in further detail in the next subsections.

3.1. GRAM Management Agent

Our grid management system has a monitoring and management agent running on each host that allows remote users to execute applications through the Globus GRAM service. 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 monitoring and management agent contains a CODE observer, actor, and manager as shown in Figure 3. 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. This latency information is also useful to users to help them select which computer systems to execute an application on. 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 [33] 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 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 GRAM reporter. The IPG is currently running the Globus Metacomputing Directory Service in classic mode. In that mode, a GRAM reporter daemon is executing on each GRAM host and writing data into a remote LDAP server. A process sensor is used to observe the status of the GRAM reporter to ensure that it is operating.

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.

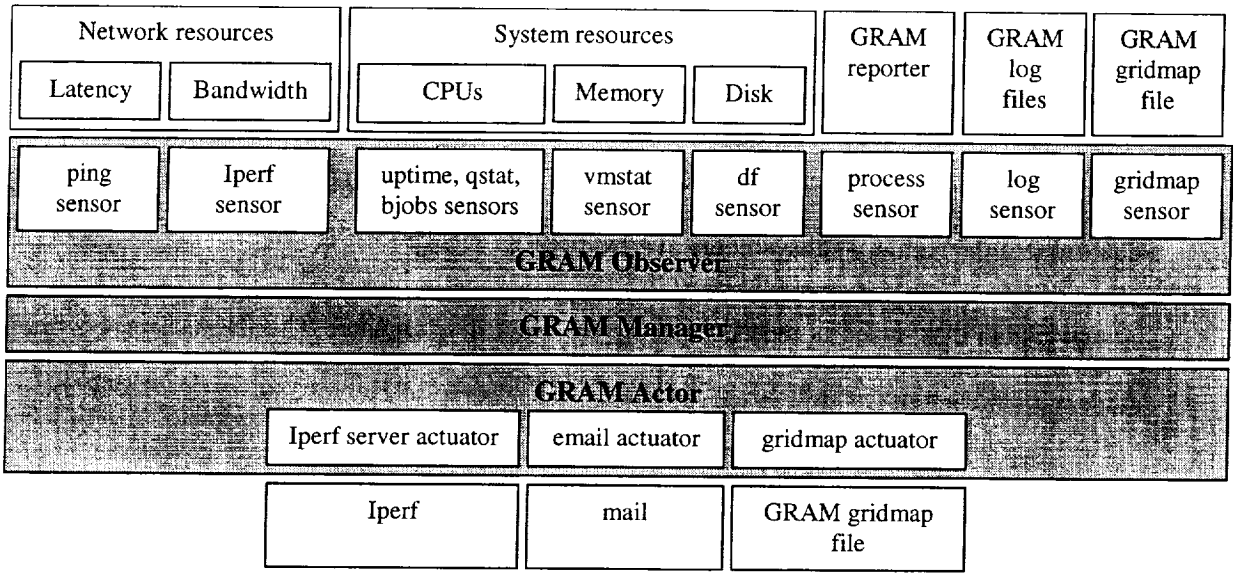

Figure 3. Monitoring and management components for a GRAM host.

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 store and forward 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

We also wish to monitor and manage the LDAP servers that make up a Grid Information Service and the hosts they execute on. This will allow us to ensure that a usable information service is available to our users a very high percentage of the time. The monitoring and management agent for each GIS server consists of an observer, an actor, and a manager as shown in Figure 4. 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. There must therefore be network connectivity between the servers. We use a ping sensor to determine if there is connectivity.
2. The CPU load of the host. If the CPU load becomes too high, the LDAP server will be slow in responding to requests.
3. The available memory of the host. Similarly, if the host is swapping memory pages to and from disk, the performance of the LDAP server will degrade.
4. The available disk space. The databases and the log files for LDAP servers can become quite large and we must ensure that there is enough disk space for them.

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.

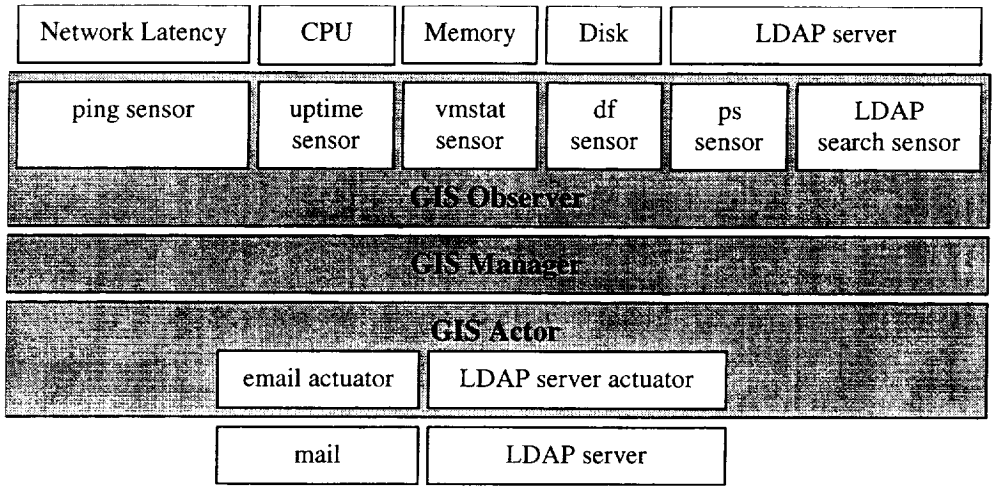

Figure 4. Monitoring and management components for a GIS host.

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. In the future this actor will most likely support more complex functionality such as compressing and moving log files to archival systems and querying the LDAP server for detailed usage information. Many commercial LDAP servers provide usage information such as average queries per second and average response time as LDAP entries that can be retrieved.

The GIS 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 GIS manager and that the system will be more scalable.

When this agent begins executing, it locates the store and forward 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. These problems include excessive CPU, memory, or disk use, lack of network connectivity to other GIS hosts, or failure of the LDAP daemon. At any time, management GUIs can contact this agent to receive information or request that actions be performed.

3.3. Store and Forward Archive

The store and forward archive has several similar purposes. First, it forwards events that GRAM and GIS monitoring agents generate when problems occur in real time to interested management GUIs. Second, these events are archived so that management GUIs can obtain information about problems that have occurred in the past. The architecture of this service is shown in Figure 5.

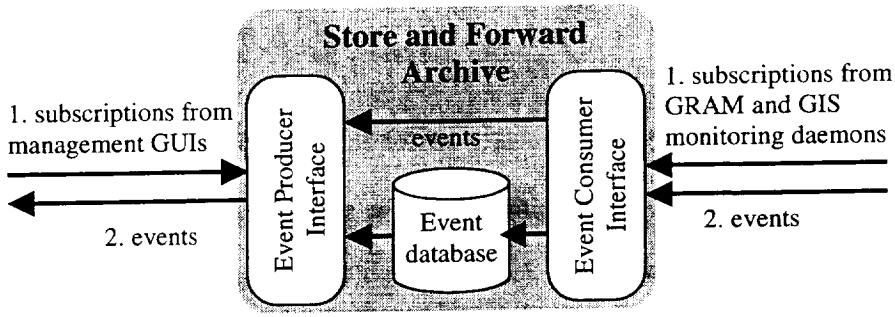

Figure 5. Architecture of the store and forward archive.

The store and forward archive acts as an event consumer for events generated by GRAM and GIS monitoring agents and acts as a producer of events for management GUIs. GRAM and GIS monitoring agents use the directory service to find the store and forward 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 subscribe for events from the archive. In the simple case, the subscription indicates that the GUIs want to receive future events that the archive receives. In the complex case, the subscription indicates that the GUI wants to receive events from the archive.

We have not seriously addressed what protocols are needed to interact with event archives so we have taken an ad-hoc approach for this prototype. When a subscription is initiated, an event filter can be specified. This filter is used to determine which events should be sent to the event consumer. Our approach is that if this filter contains a reference to an event time stamp in the past, then the event consumer must want events from the database. For example, a subscription that specified a filter of all events generated after January 1, 2001, when it is currently January of 2002, indicates that events from the event database should be sent. In this case, the archive will first send the appropriate events from the database to the management GUI. The archive will then forward any events that it receives in real time that satisfy the event filter. This forwarding of appropriate events will continue until the subscription is cancelled.

3.4. Directory Service

As described in Section 2.1.4, the observers and actors on the GRAM and GIS hosts register themselves in the directory service so that managers can find them. The store and forward 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 6 shows the management GUI being used to show the status of a subset of the resources on the NASA IPG. 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 a few expected events have not been received, and red when several 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.

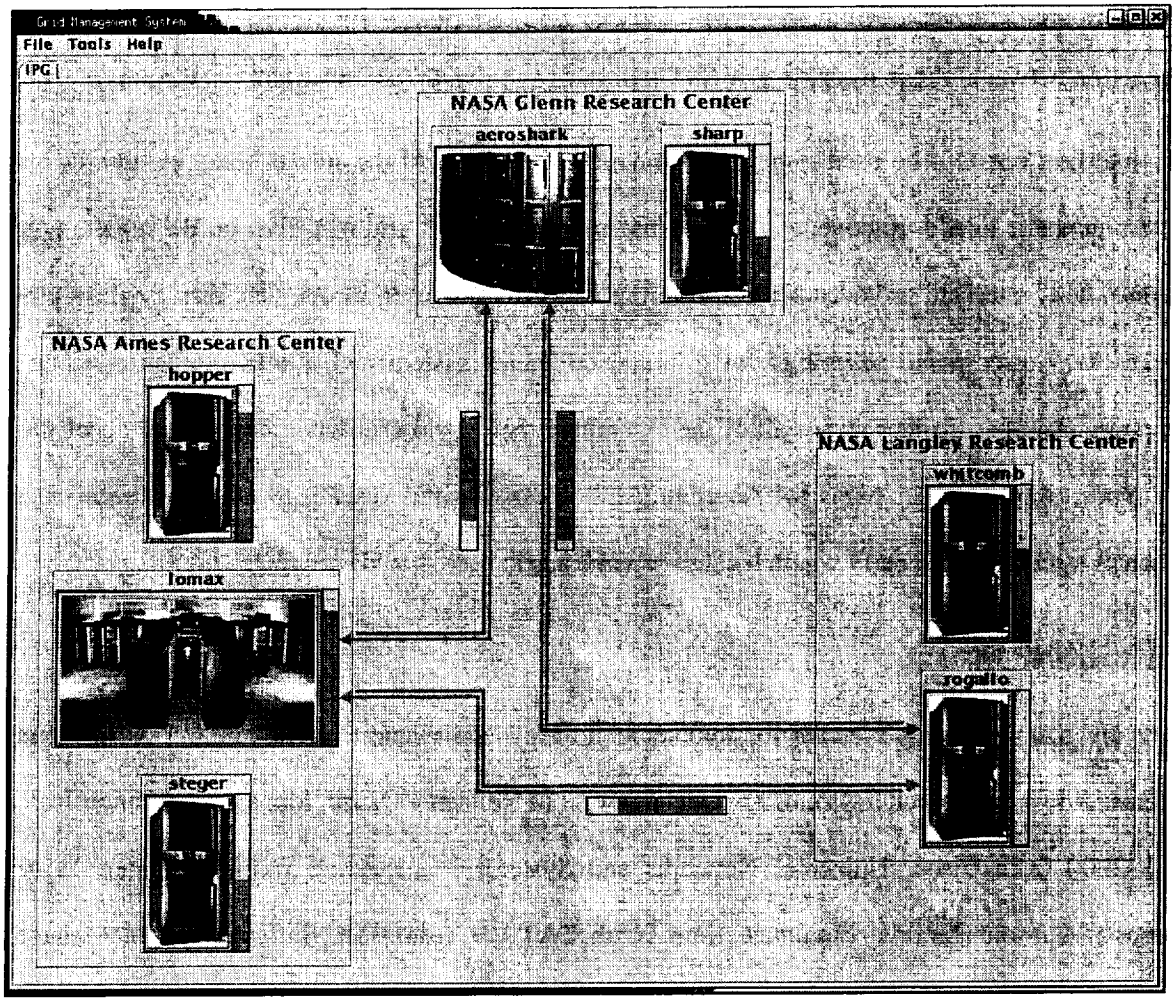

Figure 6. Management GUI displaying the status of a subset of the resources on the NASA IPG.

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 7.

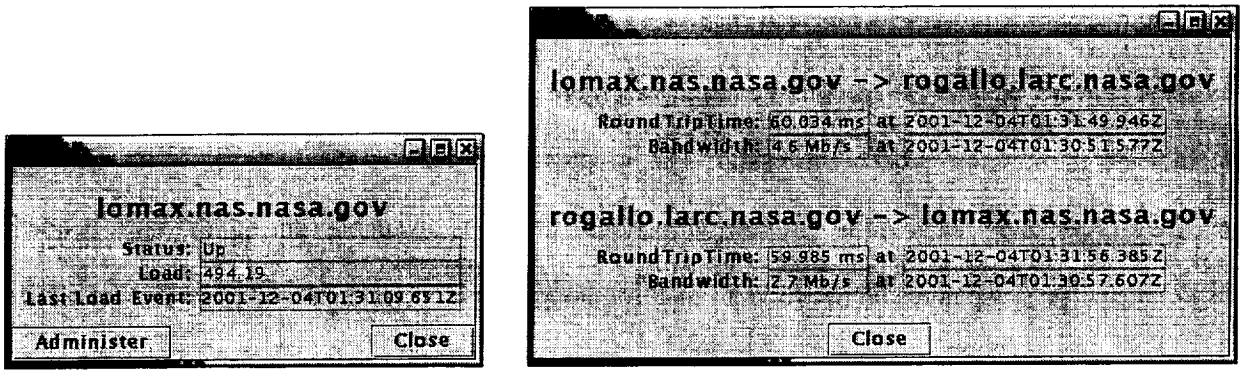

Figure 7. Detailed information about the load on the SGI Origin lomax.nas.nasa.gov and about the connection between lomax.nas.nasa.gov (in California) and rogallo.larc.nasa.gov (in Virginia).

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 from Hewlett Packard, ManagementCenter from Sun, Works2000 from Cisco, Unicenter Networks and Systems Management from Computer Associates, Enterprise Console and related products from Tivoli, PATROL from BMC Software, and SiteAssure 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. Two examples of such tools are Big Brother and Scotty. Many of these types of tools are based on the Simple Network Management Protocol [32] 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 and a notification service. The problem with these services is that they are part of CORBA, which is not commonly used in grid computing. The JINI and JXTA frameworks have support for distributed event services, but they only support the Java language. There are also research projects to develop distributed event services such as Sienna [11], Elvin [25], Echo [13], OIF [15], and XEvents [26]. 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 distributed event service components 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 our framework as it is used for new applications. For example, we will add new sensors and actuators, we may incorporate new security mechanisms such as Kerberos, and we will refine and develop functionality such as event archives and channels. 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] "CLIPS: A Tool for Building Expert Systems," http://www.ghg.net/clips/CLIPS.html.

[2] "Condor High Throughput Computing," http://www.cs.wisc.edu/condor/.

[3] "The DOE Science Grid," http://www-itg.lbl.gov/Grid.

[4] "The Globus Project," http://www.globus.org.

[5] "Grid Forum Performance Working Group," http://www-didc.lbl.gov/GridPerf/.

[6] "The Legion Project," http://www.cs.virginia.edu/~legion/.

[7] "The NASA Information Power Grid," http://www.ipg.nasa.gov.

[8] "The National Computational Science Alliance," http://www.ncsa.uiuc.edu/access/index.alliance.html.

[9] "The National Partnership for Advanced Computing Infrastructure," http://www.npaci.edu/.

[10] "The OpenSSL Project," http://www.openssl.org.

[11] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service." In Proceedings of the Ninteenth ACM Symposium on Principles of Distributed Computing, Portland, OR, 2000.

[12] C. Catlett and L. Smarr, "Metacomputing," in *Communications of the ACM*, vol. 35, 1992, pp. 44-52.

[13] 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.

[14] D. Fallside, "XML Schema Part 0: Primer," http://www.w3.org/TR/xmlschema-0/.

[15] 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.

[16] 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.

[17] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, pp. 115-128, 1997.

[18] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure,".: Morgan Kauffmann, 1999.

[19] J. Giarratano, "The CLIPS User's Guide,"., 1998.

[20] 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.

[21] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*: Macmillan Technical Publishing, 1997.

[22] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*: MacMillan Technical Publishing, 1999.

[23] 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.

[24] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System." In Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.

[25] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content Based Routing with Elvin4." In Proceedings of the AUUG2k, Canberra, Australia, 2000.

[26] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley, "SoapRMI Events: Design and Implementation," Computer Science Department, Indiana University TR549, May 2001.

[27] 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.

[28] W. Smith and D. Gunter, "Simple LDAP Schemas for Grid Monitoring," The Global Grid Forum GWD-Perf-13-1, 2001.

[29] W. Smith, D. Gunter, and D. Quesnel, "A Simple XML Producer-Consumer Protocol," The Global Grid Forum GWD-Perf-8-2, 2001.

[30] 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.

[31] W. Smith, A. Waheed, D. Meyers, and J. Yan, "An Evaluation of Alternative Designs for a Grid Information Service." In Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, 2000.

[32] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards*. Reading, Massachusetts: Addison-Wesley, 1993.

[33] A. Tirumala and J. Ferguson, "Iperf Version 1.2," http://dast.nlanr.net/Projects/Iperf/.