A Portal for Visualizing Grid Usage

Gregor von Laszewski[1,2], Jonathan DiCarlo[2], Bill Allcock[1]

[1] Argonne National Laboratory, Argonne National Laboratory,
9700 S. Cass Ave., Argonne, IL 60439
[2] University of Chicago, Computation Institute, Research Institutes Building
#402, 5640 S. Ellis Ave., Chicago, IL 60637

Abstract

We introduce a framework for measuring the use of Grid services and exposing simple summary data to an authorized set of Grid users through a JSR168-enabled portal[7, 1]. The sensor framework has been integrated into the Globus Toolkit and allows Grid administrators to have access to a mechanism helping with report and usage statistics. Although the original focus was the reporting of actions in relationship to GridFTP services, the usage service has been expanded to report also on use of other Grid services.

1 Introduction

As the Grid evolves [9] and is used in a dynamically changing environment, it is important to be able to measure how Grid services are used in an instantiation of a production Grid.

By enhancing Grid services with the ability to report its usage as part of a tightly integrated software solution, we allow production Grids to monitor which services are used when. Such usage data is essential for the development of mechanisms that deal with the ad hoc and sporadic nature of a Grid. Having such information enables the community to develop more sophisticated prediction algorithms and fault-tolerant Grid frameworks and to fulfill the need for reporting. With this information we can develop next-generation scheduling systems, quality-of-service guarantees, adaptive systems, and optimizations.

In the development of a usage service, we need to address the following elementary questions:

- What data needs to be reported?
- When do we need to report?
- Where do we collect the information and archive it?

- Who should have access to the data or a subset of the data?
- How can we use this data in developing advanced Grid services?

As with every information service in the Grid, special importance must be placed on privacy and security issues.

In this paper we introduce a framework that allows the monitoring of Grid services. The data is archived and can be visualized through a Grid portal using JSR168 compatible portlets. The framework has been tested and is now integrated as part of the Globus Toolkit.

The paper is structured as follows. First, we summarize some related work. Next, we introduce the architecture of our framework. Then we focus on the portlets that display the data as part of a portal. We conclude with a summary of the advantages of this new framework.

2 Related Work

Monitoring has always been a major part of distributed computing, including Grid computing. Hence, multiple frameworks are available, and others are under development. It is beyond the scope of this article to provide a comprehensive overview of all these efforts. We focus instead on particular features of two types of systems: those designed for monitoring resources availability and status and those designed for monitoring resource usage. Most available systems focus on resource availability and status. For example, they report which compute resources are available at a particular time, what disk space is available, and how the CPU is used. Ganglia [5], MonALISA [4], and the Globus Toolkit MDS [10, 8] are examples of general-purpose resource-monitoring systems.

Usage-monitoring systems answer the question: What are users doing with the resources? Such systems may be used as part of intrusion detection services or advanced Grid services that dynamically adapt based on use patterns. Examples of projects that deal with usage data are the GGF GAMA working group but can reach as far as the Round Robin Database tool (RRD) [6], which is similar to our framework's visualization component in that it generates charts from quantities in a database.

3 Design and Architecture

The design of our Grid Usage Sensor Service (GUSS) is independent of the Globus Toolkit and can, in principle, be reused by other frameworks and Grid services. To demonstrate its usefulness in a real Grid middleware, however, we are paying special attention to how we can use our framework in the Globus Toolkit. . The GUSS architecture is inherently distributed in order to foster the distributed nature of Grids. Figure 1 depicts a high-level architectural view of the GUSS framework, while demonstrating the integration of services of the Globus Toolkit that have been augmented with usage sensors. We distinguish the following components:

- Sensor: A sensor is a component that senses the use of a component and forwards this information to a usage collector.
- Server: A usage collector receives the information of a sensor and forwards the information to a server that internally prepares the data for archiving to a database or uses e-mail notifications to inform users about usage information.
- Visualization server: The visualization server contains a JSR168 portable portlet that, based on information filled out be a user, retrieves data from the database and prepares it in a suitable diagram or table via a charting component. A portlet displays a window in which the user can select and query the database. Through the portal the access to the portlet can be restricted.
- Thick client: In addition to the portlet, the user can also browse the data directly through a Java Swing component if the user is allowed to access the Web service.

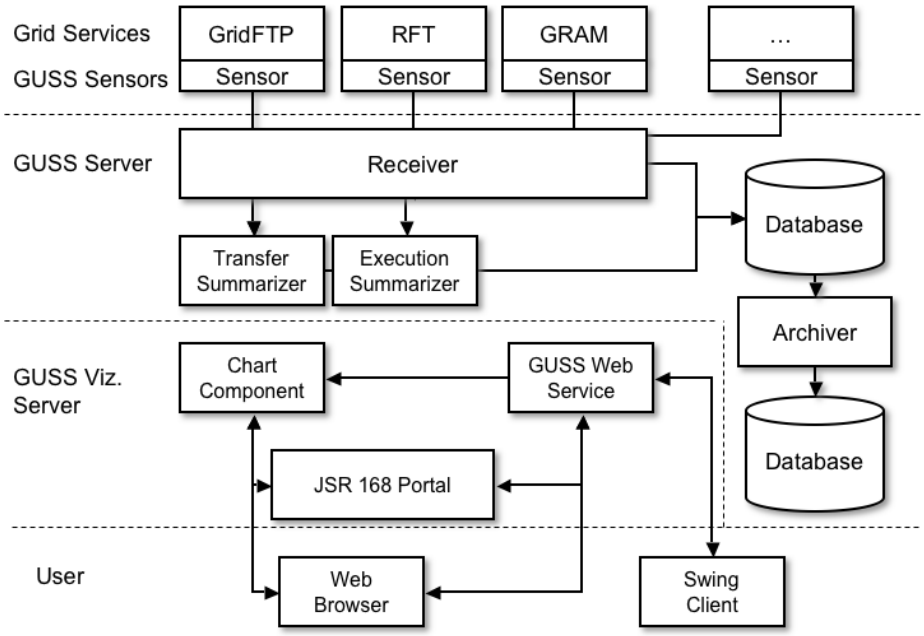

Figure 1: Grid usage sensor and service framework architecture.

We have designed our framework to be integrated into Grid middleware such as the Globus Toolkit. Specifically, we have provided a language-independent implementation that uses specialized protocols between the sensors and the

service. Hence it is possible to provide implementations, for example, in C and Java as they build the basis for the Globus Toolkit implementation. We have developed specialized sensor protocols and the associated sensors and clients for the GridFTP server, the Reliable File Transfer service, the Replication Location Services, the Java Web Services Core, the C Web Services Core, and the Grid Resource Allocation Manager. As each component can log its own data, one can customize what data is reported. For example, whenever the GridFTP server finishes sending or receiving a file, it sends out packets containing file metadata such as size and information about the transfer start and end times. In contrast, the Java Web Services core sensor [22] sends out packets each time the container is started or stopped, and includes a list of the Web services that are running in the container.

At present we use UDP to submit the data between the sensor and the clients (UDP is also used by the Condor project in their monitoring framework). However, one could adapt the architecture to use different protocols that provide more reliability. In our tests we found that use UDP provides several advantages, especially if parts of a network become unavailable. In such a case summary information can be send out at a later time. To improve performance under load, we have integrated in the server two listener threads for each sensor type: a high-priority thread catches packets and puts them unchanged into a ring buffer; a lower-priority thread takes packets out of the ring buffer and parses them. Once the server has parsed the data, it is analyzed and archived. Summary information is also generated periodically and stored in a historical database.

A cron job is used to control the frequency of the statistical analysis process. Another cron job allows the service to forward the summary information to a mailing list, to which authorized users may subscribe.

To improve the performance of the framework, we have taken care to ensure that the database and the GUSS server can be run on different machines. Furthermore we have integrated a cron job that moves sensor data to a secondary archive tables at a given time interval. Hence the data immediately accessible for processing in the server is kept small. The architecture supports two mechanisms to expose the data: a Java client that allows one to query and display the data in graphical form, and a JSR168-based portlet that allows the display in a portal.

3.1 Sensor Data Format and Components

To parse the packets, we use the "chain of responsibility" pattern [2]. For each Grid service class we customize handlers. Hence, one can develop new handlers and extend the framework with user-defined sensors and their analysis. For rapid parsing we have designed our packets according to the information provided in Table 1.

To decide whether to handle a packet, a handler examines the first two bytes. Our usage update packets always contain a Grid service class identification in the first 8 bytes and a packet version identification in the next 2 bytes. The advantage of using this pattern is that when we add monitoring for a new Grid

Table 1: Makeup of general usage packets

Bytes	Contents
0-1	Service class identification
2-3	Packet version identification
4-end	Component-specific information

service, one can easily write a new handler subclass and register it with the listener. Also, if we need to change the format of a packet, for instance when we start monitoring a new feature, we can assign a new packet version identification and create a handler for it, while leaving the old handler to continue parsing packets from the old version.

At present, we keep the length of all usage update packets under 1472 bytes to avoid packet fragmentation, since many routers discard fragmented UDP packets. We have assigned a service class identification for several components of the Globus Toolkit [3] according to the numbers 0 to 7. This includes GridFTP, GRAM, Java Core, C WS Core, RFT, and RLS.

It is beyond the scope of this paper to present a detailed technical description of all the different sensor packets designed for the Globus Toolkit; however, we select the current implementation of the GridFTP packets as an example. The data transmitted from the sensors to the server is organized as depicted in Table 2. We note that the contents starting at byte 24 contains a separate list of name value pairs. To reduce the size of the packets, one could assign a mapping between these value pairs and a number of fixed bytes. Currently, however, we use fully quantified names as depicted in Table 3.

Table 2: GridFTP usage packet contents, by bytes

Bytes	Contents
0-1	Component code (0 for GridFTP)
2-3	Packet version code
4-19	IP address (if IPv4, the first 12 bytes will be 0)
20-23	Timestamp (UNIX format)
24-end	Text containing fields in the NAME=VALUE format, separated by spaces

Of these fields, the ones of most interest are the start and finish times of the transfer, the host, the file size, and the operation (store or retrieve). During the summarization process, many derived quantities are calculated from this basic information. A GridFTP transfer can involve up to three hosts: the sender, the receiver, and the host that commanded the transfer, which may be distinct from the other two. Since the GridFTP server sends update packets on both store operations (file received) and retrieve operations (file sent), the listener

Table 3: GridFTP usage packet contents, by field name

Field Name	Contents
HOSTNAME	Name of the host logging the transfer
START	Time that transfer was initiated, in YYYYMMD-Dhhmmss.uuuuuu format (where u is microseconds)
END	Time that transfer was completed, in YYYYMMD-Dhhmmss.uuuuuuu format
VER	Quoted string describing GridFTP server version.
BUFFER	Size of the buffer used in the transfer.
BLOCK	Block size used in the transfer.
NBYTES	Number of bytes in the file transferred.
STREAMS	Number of streams used in the transfer.
STRIPES	Number of stripes used in the transfer.
TYPE	"STOR" if the logging host is storing the file to disk, "RETR" if the logging host is retrieving it from disk.
CODE	Completion status code. 226 is the code for a successful transfer.

sometimes receives two update packets for the same transfer. It stores both packets in the database; the GUSS service must handle such a situation and avoid double-counting the transfer.

3.2 Scalability

We began data collection in March 2005. By June 2005 the number of GridFTP packets alone was already over 6 million, and we expect the growth to accelerate as more sites adopt Globus Toolkit 4. Even if the storage space is not a problem, the need to search so many rows may adversely affect query performance. Therefore, we use summarization and archival to reduce the need for queries to search through all of these packet records. The data that we summarize includes obvious data points such as the total and average number of data entries as well as histograms showing the total number of hosts, transfer speed, transfer duration, and transfer size and their standard deviation.

3.3 Visualization

To provide adequate support for analyzing and monitoring the data, we have developed also a visualization strategy based on a portal that interfaces to a GUSS Web server. It responds to two types of user requests: requests for plots of quantities over specified time periods and requests for tabular summaries of overall usage. To answer these requests, it queries the database, compiles the individual usage packets into a summary for each time period, generates an image file of a plot if necessary, and returns an html fragment to the client. The

primary intended client is the GUSS portlet, but the service is general enough to be used by arbitrary clients.

An example of input to the portlet is the number of GridFTP transfers during a specific hour, day, or week. Eight statistical quantities can be requested from the service. Each of the quantities is calculated from the low-level data in the individual GridFTP transfer records that internally are aggregated over the time period specified by a granularity specification. The following quantities are observed :

- Number of unique hosts active during the time period (where "active" means that they sent or received at least one file)
- Number of files transferred (regardless of host) during the time period
- Total number of bytes in all files transferred during the time period
- Number of new hosts observed for the first time during the period
- Mean time taken for a single transfer, with standard deviation, averaged over the time period
- Mean size of a file transfer, with standard deviation, averaged over the time period
- Mean file transfer speed (size divided by time), with standard deviation, averaged over the time period
- Mean number of streams used in transfers, averaged over the time period

Internally, the functionality of the GUSS Web service can be divided into subtasks as follows:

1. Get and parse request coming from GUSSClient, including the time period (start and end dates) in question, the quantity to be plotted, and the time granularity (by hour, by day, etc.)
2. Compare new request to recently served requests to see whether an existing image file can be reused.
3. Check whether summaries exist for all of the time periods between the requested start and end dates and for all of the host categories of interest.
4. Get records from GridFTP transfer database table. (There have been on occasion hundreds of thousands of rows within a single hour, so to avoid running out of memory, we get these records a few at a time.)
5. Process all retrieved records to calculate averages, totals, and standard deviations, and store all of these as a summary in the database, keyed to the time period, for later use. Create one summary for each category of host.

6. Using the summaries for each time period in the range, create chart of the user's chosen quantity, and save this as an image file. Return to the client an html fragment containing a link to the image filed

Alternatively, return to the client an html fragment containing a table of numerical data calculated from the summaries. Figure 2 shows an example of the portal while depicting the number of active hosts participating in GridFTP transfers at a specified day.

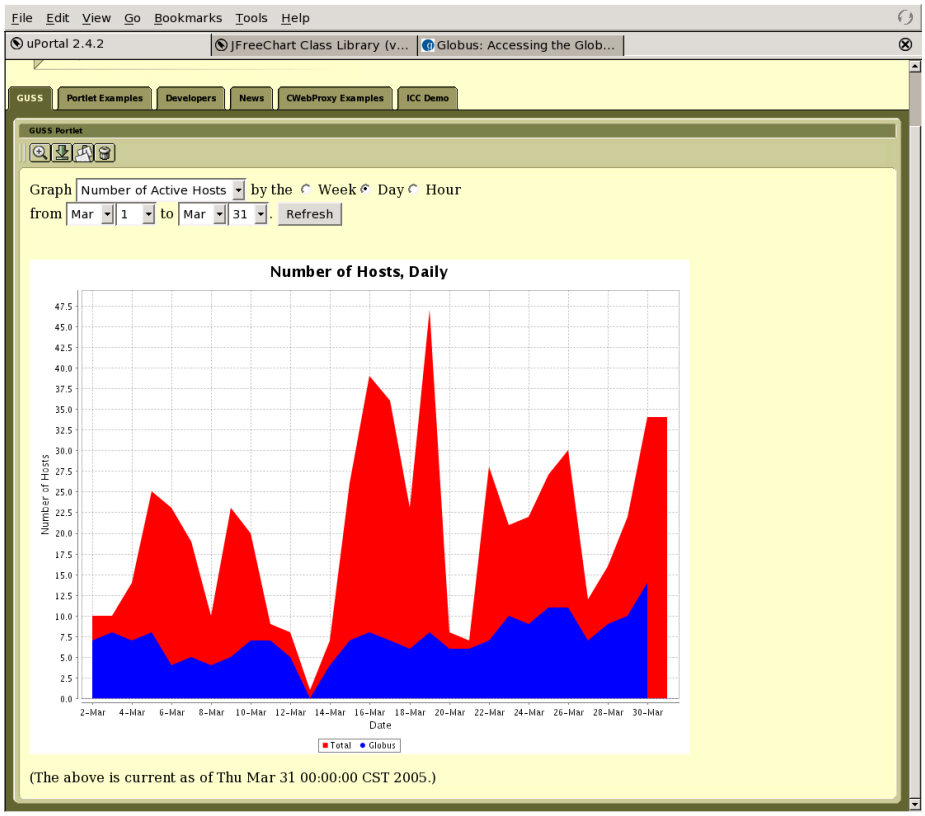

Figure 2: Query issued to the GUSS database from the GUSS portlet.

4 Results

Between June 18 and August 18, 2005, GridFTP usage packets were received from 428 unique hosts in 30 countries. The breakdown of these hosts by top-level domain is shown in Table 4. Packets received from hosts in the mcs.anl.gov and isi.edu domains are excluded because these domains are used as GridFTP testbeds and produce a very large volume of packets. Out of the 428 hosts 22

hosts were in the mcs.anl.gov and isi.edu domains, but these 22 hosts logged 38.1% of all usage packets (626,086 packets out of 1,643,596). GridFTP usage packets outnumbered packets from all other Globus components combined, and made up 72.33% of all packets received (see Table 5).

5 Conclusion

Development of the Grid requires us to think at higher levels of abstraction compared to traditional software development. For this purpose, a bird's-eye view of activity is invaluable. A usage sensor framework such as that introduced here may help advance the development of more sophisticated Grid services. It may also help Grid users and administrators in evaluating a snapshot in time and in identifying which Grid services are used. By presenting the information graphically through a portal, we enable users and administrators to search for use patterns that otherwise would be more difficult to find. This service distinguishes itself from other services such as MDS in that it reports on actual resource usage instead of resource status.

Acknowledgments

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. We acknowledge the many Globus Toolkit developers who have provided customized protocols and handlers for the different Globus services reported on in this paper. The GUSS project has been sponsored by the NSF NMI program.

References

[1] Alejandro Abdelnur and Stefan Hepper. Java specification request 168: Portlet specification. Web, October 2003. Available from: `http://www.jcp.org/en/jsr/detail?id=168`.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[3] Globus Release Documentation. Web Page, 2006. Available from: `http://www.globus.org/toolkit/docs/4.0/`.

[4] I.C. Legrand, H.B. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre. MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, 27 September - 1 October 2004. CERN. Available from: `http://monalisa.cacr.caltech.edu/documentation/monalisa_chep04.pdf`.

[5] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 2003. (submitted). Available from: `http://ganglia.sourceforge.net/talks/parallel_computing/ganglia-twocol.pdf`.

[6] Tobias Oetiker, Jake Brutlag, and Alex van den Bogaerdt. RRDtool: logging and graphing. Web, 2005. Available from: `http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/index.en.html`.

[7] Open Grid Computing Environments. Web Page. Available from: `http://www.ogce.org`.

[8] Jennifer M. Schopf, Mike D'Arcy, Neill Miller, Laura Pearlman, Ian Foster, and Carl Kesselman. Monitoring and discovery in a web services framework: Functionality and performance of the globus toolkit's mds4. Preprint ANL/MCS-P1248-0405, Argonne National Laboratory, Argonne, IL, 2005. Available from: `http://www-unix.mcs.anl.gov/~schopf/Pubs/mds-sc05.pdf`.

[9] Gregor von Laszewski. The Grid-Idea and Its Evolution. *Journal of Information Technology*, 47(6):319–329, June 2005. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-grid-idea.pdf`.

[10] Gregor von Laszewski, Steve Fitzgerald, Ian Foster, Carl Kesselman, Warren Smith, and Steve Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, OR, 5-8 August 1997. Available from: `http://www.mcs.anl.gov/~gregor/papers/fitzgerald--hpdc97-mds.pdf`.

Table 4: Known GridFTP hosts by top-level domain

Domain	Number	Comments
.gov	28	(of which 8 are mcs.anl.gov)
.edu	71	(of which 14 are isi.edu)
.com	5	
.org	126	(of which 97 are teragrid.org)
.mil	1	
.net	14	
.am	1	(Armenia)
.ar	2	(Argentina)
.at	9	(Austria)
.au	9	(Australia)
.br	3	(Brazil)
.ca	10	(Canada)
.ch	1	(Switzerland)
.cl	2	(Chile)
.cn	1	(China)
.cz	2	(Czech Republic)
.de	6	(Germany)
.es	21	(Spain)
.fi	3	(Finland)
.gr	4	(Greece)
.hr	4	(Croatia/Hrvatska)
.it	14	(Italy)
.ie	1	(Ireland)
.in	1	(India)
.jp	14	(Japan)
.kr	12	(South Korea)
.nl	1	(Netherlands)
.pl	3	(Poland)
.ru	9	(Russia)
.sg	5	(Singapore)
.sk	1	(Slovak Republic)
.th	1	(Thailand)
.tw	11	(Taiwan)
.ua	2	(Ukraine)
.uk	23	(United Kingdom)
.us	1	(United States)
Total	422	(Not counting 6 hosts whose top-level domains could not be identified.)

Table 5: Usage packets arranged by Globus component

Component	Number of Packets	Percentage of Total
GridFTP	1643596	72.33%
C WS Core	180454	7.94%
GRAM	173936	7.65%
Java WS Core	57956	2.55%
RLS	6398	0.28%
RFT	210063	9.24%
unparsable	256	0.01%
Total	2272403	