CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

A Portal for Visualizing Grid Usage*

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

[1] *Argonne National Laboratory, Argonne, IL 60439*
[2] *University of Chicago, Computation Institute, Chicago, IL 60637*

SUMMARY

We introduce a sensor 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. 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 ont the use of GridFTP services, the usage service has been expanded to report also on the use of other Grid services.

KEY WORDS: Grid, Grid Monitoring, Grid Usage, Java CoG Kit

1. Introduction

As the Grid evolves and is used as part of dynamically changing environments, it is important to be able to measure how Grid services are used within 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 access to such information enables the community to develop more sophisticated prediction algorithms, 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:

1. What data needs to be reported?
2. When and how often do we need to report?

*Correspondence to: Gregor von Laszewski, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440,U.S.A., E-mail: gregor@mcs.anl.gov

Received 9 December 2005
Revised

3. Where and how do we collect the information and archive it?
4. Who should have access to the data or a subset of the data?
5. How can we use this data in developing advanced Grid services?

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 [1]. The framework has been tested and is now integrated as part of the Globus Toolkit. The paper is structured as follows. First, we position our work in relationship to other Grid information services. Next, we introduce the architecture of our framework. Then we focus on the display of the data as part of a portal. Finally we present our summary.

2. Related Work

Monitoring has always been a major part of distributed computing, including Grid computing. In general, we find two types of systems: a) those designed for monitoring resource availability and status and b) those designed for monitoring resource usage. Most available Grid monitoring 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, MonALISA [4], NWS [2], and the Globus Toolkit MDS [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. Grid-based examples of such systems are discussed within the Global Grid Forum [7]. In addition, we find tools and frameworks that support the development of presentation components for monitoring systems such as the Round Robin Database tool (RRD) [5], and the Open Grid Computing Environment (OGCE) framework [6] to export the information as part of a portal. Our system falls into the class of monitoring resource usage and at the same time provides a framework for visualizing the results within a portal.

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 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 support the distributed nature of Grids. Figure 1 depicts the 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:

GUSS sensor: A sensor identifies the use of a component and forwards this information to a usage collector.

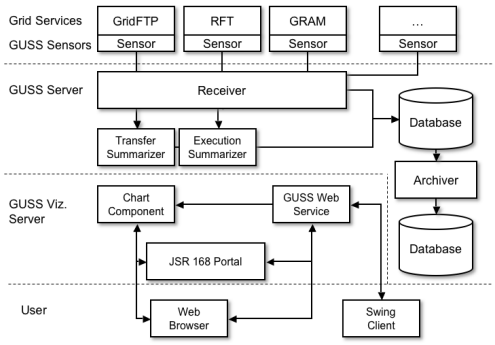

Figure 1. Architecture.

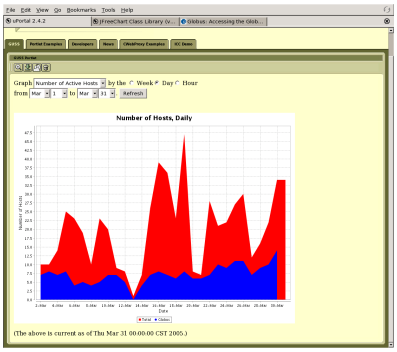

Figure 2. Portal.

GUSS server: A server contains a receiver that collects the information of a sensors. The data is than internally prepared for archiving to a database or for e-mail notifications to inform subscribed users about usage information.

GUSS visualization server: The visualization server contains a JSR168 portable portlet. Based on a query an authorized user fills out, it retrieves data from the database and displays it in a suitable diagram or table via a charting component.

GUSS thick client: In addition to the portlet, the user can also browse the data directly through a Java Swing component.

One of the architectural principles that we have employed is to design a language-independent implementation based on specialized protocols between the sensors and the service. However, to focus on the instrumentation of the Globus Toolkit, we have developed specialized sensor protocols and the associated sensors and clients for the GridFTP server [9], 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. Furthermore, we have integrated a number of parameters allowing us to control on a component basis what data is reported. For example, whenever the GridFTP server finishes sending or receiving a file, the GridFTP sensor sends out information containing metadata such as size and information about the transfer start and end times. In contrast, the Java Web Services core sensor [**?**] sends out information 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 information between the sensor and the clients. Although, one could adapt the architecture to use different protocols that provide more reliability, we found that using UDP provides the advantage to be independent from network

outages. In case of failures summary information could 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 periodically generated and stored in a historical database. A cron job is used to control the frequency of the statistical analysis process. Another cron job is used to initiate that the server forwards summary information to a mailing list to which 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 hosted on different machines. Furthermore, we have integrated a cron job that moves sensor data to a secondary archive 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 querying and displaying 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 [3]. 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 to include a component identifier in the first two bytes, a protocol version identifier in the next two bytes, and component specific information in the rest of the packet. The advantage of using this pattern is that when we add monitoring for a new Grid 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 like to monitor a new feature, we can assign a new version identifier and create a handler for it, while leaving the old handler to continue parsing packets. At present, we keep the length of all usage update packets under 1472 bytes to avoid packet fragmentation.

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, in order to give an idea about the kind of information reported, we look at the current implementation of the GridFTP GUSS packets in more detail. The data transmitted from the sensors to the server contains the following component specific data: IP address, a timestamp, and a number of name value pairs where the values store information for the name of the host logging the transfer; the time that transfer was initiated; the time that transfer was completed; the GridFTP server version; the size of the buffer used in the transfer; the block size used in the transfer; the number of bytes in the file transferred; the number of streams used in the transfer; the number of stripes used in the transfer; the types of transfer that identifies if the logging host is storing the file to disk, or retrieving it from disk. Of these fields, the ones of most interest are the start and finish times of the transfer, the host, the file size, and the operations store or retrieve. During our summarization process, many derived quantities are calculated from this basic information. Carful analysis is necessary, because a GridFTP transfer can involve up to three hosts, the sender, the receiver, and the host that commanded the transfer. Hence, the GridFTP server

sends update packets on both store operations (file received) and retrieve operations (file sent), which consequently results in the listener to potentially receive two update packets for the same transfer. To increase scalability this data will be first simply stored in the database, but the analysis process must take into account the possible duplication of information in order to avoid double counting. A purge process can eliminate duplicated data.

3.2. Scalability

We began data collection in March 2005 as part of an evaluation phase. By June 2005 the number of GridFTP packets reached over 6 million. Even if the storage space is not a problem, the need to search and analyze so much data may adversely affect the query performance. Therefore, we concluded to use a summarization and archival process 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 a visualization framework based on a portal that interfaces to a GUSS Web server and can access the data collected. It responds to two types of user requests: requests for plots of quantities over specified time periods and requests for tabular summaries of the 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 page with the summary information to the client.

Figure 2 shows a screenshot of the portal showing the result to a query to depict the number of active hosts participating in GridFTP transfers at a specified day.

These quantities that can be queried are as follows:

- 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; and
- Mean number of streams used in transfers, averaged over the time period;

Internally, the functionality of the GUSS Web service can be divided into the following tasks:

1. Get and parse queries coming from a GUSS client. Queries include information about 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 cached query result 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. Process the retrieved records to calculate averages, totals, and standard deviations, and store the results in a summary database.
5. Using the summaries for each time period, create a chart according to the users query, and save it as image in a file. Return to the client an html file containing a link to the image. Alternatively, return to the client an html fragment containing a table of numerical data calculated from the summaries.

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 I. 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 (626086 packets out of 1643596). GridFTP usage packets outnumbered packets from all other Globus components combined, and made up 72.33% of all packets received (see Table II).

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 potentially 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.

Table I. Known GridFTP hosts by top-level domain

Domain	Number	Comments
.gov	28	
.edu	71	
.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
.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	

Table II. Usage packets 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	

6. 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. We acknowledge the many Globus Toolkit developers who have provided customized protocols and handlers for the different Globus services reported on in this paper.

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. B. Gaidioz, R. Wolski, and B. Tourancheau. Synchronizing Network Probes to avoid Measurement Intrusiveness with the Network Weather Service. In *Proceedings of 9th IEEE High-Performance Distributed Computing Conference*, pages 147–154, August 2000. Available from: http://www.cs.ucsb.edu/~rich/publications/.
3. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
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. 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.
6. Open Grid Computing Environments. Web Page. Available from: http://www.ogce.org.
7. OGSA Resource Usage Service (RUS-WG). Available from: https://forge.gridforum.org/projects/rus-wg.
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. GT 4.0 GridFTP. Web, 2005. Available from: http://www-unix.globus.org/toolkit/docs/4.0/data/gridftp/.

Disclaimer