GES 2007 German e-Science

Available online at http://www.ges2007.de

User-Centric Monitoring and Steering of the Execution of Large Job Sets

Ralph Müller-Pfefferkorn[1], Reinhard Neumann[1], Thomas William[1], Stefan Borovac[2], Torsten Harenberg[2], Matthias Hüsken[2], Peter Mättig[2], Markus Mechtel[2], David Meder-Marouelli[2], Peer Ueberholz[3], Peter Buchholz[4], Daniel Lorenz[4], Christian Uebing[4], Wolfgang Walkowiak[4], Roland Wismüller[4]

[1] Center for Information Services and High Performance Computing
Technische Universität Dresden, D-01062 Dresden, Germany
[2] Bergische Universität Wuppertal, Gaußstraße 20
D-42119 Wuppertal, Germany
[3] Hochschule Niederrhein, Reinarzstraße 49
D-47805 Krefeld, Germany
[4] Universität Siegen, D-57068 Siegen, Germany

Abstract

Processing of large data sets with high through put is one of the major focus of Grid computing today. If possible, data are split up into small chunks that are processed independently. Thus, job sets of hundreds > or even thousands of individual jobs are possible. For the job submitter or the resource providers such a scenario is a nightmare currently, as it is hard to keep track of such an amount of jobs or to identify failure reasons.
We present a system that will support gLite users to track and monitor their jobs and their resource usage, to find and identify failure reasons and even to steer running applications.

1 Introduction

Today, one of the major challenges in science is the processing of large datasets. Experiments or simulations can produce an enormous amount of results that are stored in databases or files. Processing these data is usually done by splitting the analysis process into a large number of small jobs that read only chunks of the data. By running these jobs in parallel on a Grid the processing time can be decreased significantly. Examples of such a scenario are the processing of images in medicine or the analysis of high energy physics data. The Large Hadron Collider (LHC) at CERN will provide particle physicists with several Petabytes of experiment data every year. Additionally, simulations of the physics processes and the detector response are needed to understand the experiment.

With the LHC Computing Grid (LCG) the High Energy Physics community wants to provide a Grid environment to enable such data processing capabilities. The gLite middleware stack is the base of this Grid effort.

To monitor the hundreds or thousands of jobs a physicist usually submits for an analysis intelligent tools are needed to support the user. The existing monitoring tools of the LCG/gLite environment currently provide only limited functionality. Either they focus on the underlying fabric, i.e. hardware, infrastructure (e.g. the LCG Real Time Monitor or GridIce) or are only simple command line tools flooding the user with textual information.

The High Energy Particle Physics Community Grid project[1] (HEPCG) [1] of the German D-Grid Initiative [2] wants to contribute to the functionality the LCG provides to the users. In this paper the user monitoring tools that have been developed in HEPCG are presented. Section 2 describes the job and resource usage monitoring system, that supports users in handling the large number of jobs. Section 3 presents the job execution monitor to track down problems in single steps of a job. Finally, a tool for online steering of jobs in LCG is introduced in section 4.

2 User-centric monitoring of jobs and their resources

AMon, the user-centric monitoring tool delivers the user information about the status and the resource usage of the submitted jobs. The information are answers to questions like:

- Is the job still running, successfully done or did it fail?
- What is the CPU usage over time? What is the memory consumption on the machine where the job is running? What about the I/O of my program?
- Are there any critical usage parameters of my jobs like a directory filling up or a job hanging ?

Such information can indicate a untroubled run or problems of the users application. In this context, the term "user" comprises both the submitters of the jobs, who want to know what is going on with their jobs, and the resource providers who want information about the usage of their resources.

Due to the large number of jobs and thus the large amount of monitoring data there are several constraints for the monitoring to be beneficial for the scientists:

1. Easy access and handling - only limited knowledge about monitoring should be needed by the user
2. Support the users with graphical representations of the pre-analysed information, that allow interaction to get further and detailed information
3. Authentication, authorisation and secure data transmission for data privacy reasons

The first constraint led to the decision to realise a web browser based interface. Due to the usage of web browsers in daily work people know how to handle and to work with them. A new tool would need new and additional knowledge. Therefore, the monitoring is integrated into a Web portal using the GridSphere

[1]funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01AK802C

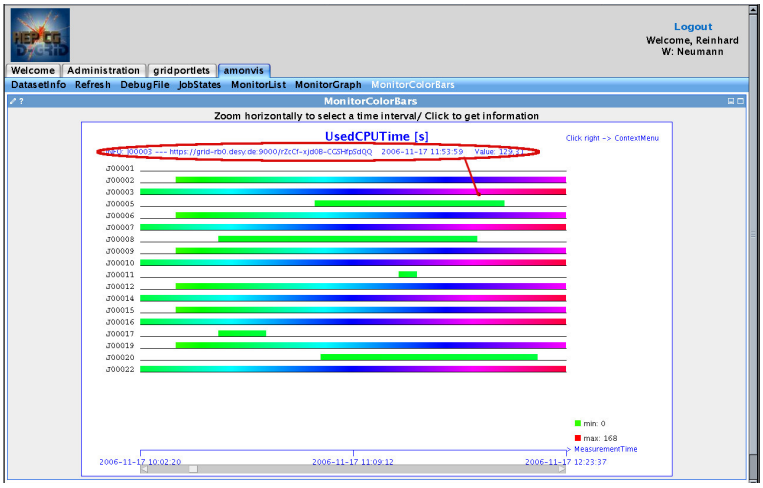

Figure 1: The monitoring is integrated into GridSphere. Here an example graph of the temporal development of an information (the CPU usage of the jobs). data is denoted by the scrollbars of the display.

portal technology [3]. It already provides features like user management and user login (with password or credentials). After login the user obtains the monitoring data by a click. Java applets - running inside the portlets - handle the visualisation of the data. The data are provided in a diversity of displays. Examples for displays are statistical summaries (e.g. pie charts) that give a general overview. Timelines (the temporal development of a information) inform about the dynamic behaviour of the jobs regarding a metric (a measurement, e.g. the used CPU time). An example screenshot is given in figure 1. Clicking into the charts will display more detailed information or allow the user to zoom into the data.

The distributed monitoring data are gathered and pre-analysed by a Web Service that the GridSphere portlet is contacting. Currently, it collects information from tables in R-GMA, the Relational Grid Monitoring Architecture [4]. R-GMA is a kind of a distributed relational data base that is used in LCG to store monitoring data. As the Web Service provides a generic interface access to any monitoring system could be plugged in.

The job monitoring information are measured and collected on the worker nodes (computers) where the jobs are running. The existing LCG worker node monitoring [5] was extended to collect a variety of useful information (see table 1). These data are stored in variable and configurable time intervals into R-GMA. The default, e.g. collects information every 3 minutes for the first 20 minutes of the runtime of the job, every 10 minutes for the next 40 minutes and then every half an our for the rest of the runtime.

The overall architecture with the described four components - information collection on the nodes where the jobs run, information storage, information analysis and the user interface - is sketched in figure 2.

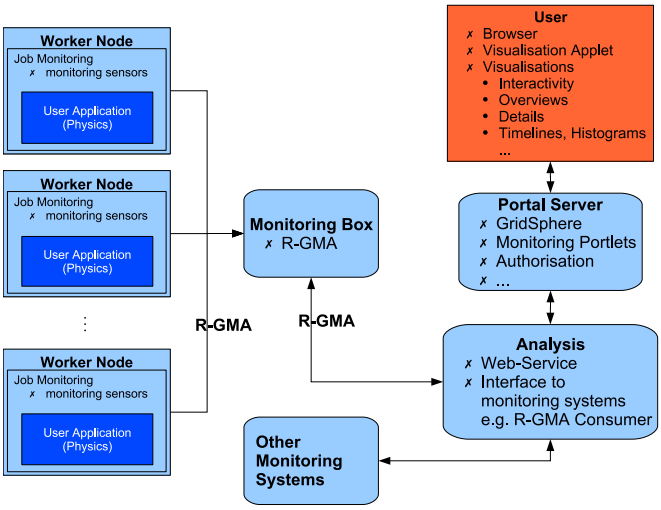

Figure 2: Architecture of the Job and Resource Usage Monitoring System

Future versions will extend the user interface with an authorisation framework using VOMS [6]. This will allow different levels of access rights to the monitoring data (a user, a site administrator, a VO manager). Furthermore, filters will pre-analyse the data to give the users direct hints and help in finding problems.

Category	**Metrics**
General	job ID; user name; the names of the resource broker, the computing element and the worker node (WN); job ID on the WN
CPU	WallClockTime; UsedCPUTime; load averages
Memory Storage	real, virtual, total, and free memory; free and total swap space free space on home, temporary and work directory; summary of file system properties
File I/O	I/O rates for every file access by the application
Network	received and transmitted network

Table 1: Available metrics of the extended LCG worker node monitoring

3 Monitoring the execution of jobs

In contrast to the existing monitoring concepts in gLite/LCG which focus on infrastructure, we developed the ”Job Execution Monitor” (JEM), a Python

based software which monitors the execution of script files within the user jobs.

Its purpose is to watch the execution of every command the user job wants to execute on a compute node and to provide information about its success or failure to the user on the user interface.

3.1 Structure of the Job Execution Monitor

There are two main components of the JEM, which both run on the compute node: the Job Wrapper and the Watchdog. Furthermore, the Wrapper makes use of the script parser, which is a third independent module.

The Script Wrapper can be thought of as an interpreter for the supported script languages. Currently, shell scripts for the *sh* and *bash* shell and the python programming language are supported, but it is planed to provide a generic framework for other languages. With this it will be possible to write separate modules/plugins, so that more languages may be interpreted as well.

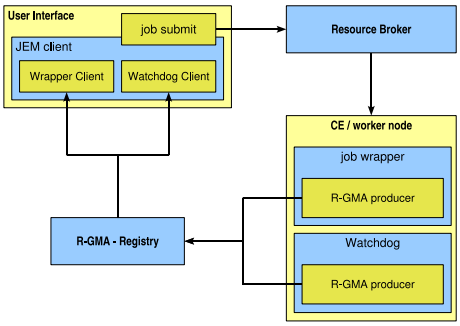

Figure 3: Job Execution Monitor, global principle

With this kind of generic interpreter for script files, it is possible to do other things in between the execution of the actual commands. Especially, the successful execution of every command can be checked, monitored and logged, so that the user is informed about the current state of his program, at all time. But as the most important (and most difficult) requirement, such a system must handle a given script file as if it were executed without this monitoring framework.

3.2 Script Wrapper

The Script Wrapper is the main component for monitoring the execution and the current state of the user job. It has a modular structure, which makes it easy to add support for other script languages.

3.2.1 Bash Wrapper

Stepwise execution of bash scripts takes place in two steps. First, a given bash script is parsed and a modified script file is then written to disk.

As a second step, the Bash Wrapper launches the modified script file created by the Bash Parser, which then runs in parallel to the wrapper. With this concept it is possible to track all input and output of all commands and the success of every single command can be checked. This mechanism is described in more detail below. Even more detailed information can be found in [7] and [8].

Bash Parser The purpose of this component is to analyse a given shell script, find every single command and put an escape sequence in front of it. This escape sequence is a call to a python module, which gets the actual command as a shell parameter. This intermediate python program can then do anything it likes with the command string. Of course, in the end, the command should be executed. But with this technique, it is possible to do other things between the call of a command and its execution.

The *bash* parser does his work in three steps: it starts with a lexical analysis of the script, then parses the script to identify where the commands are and finally it inserts the escape sequences to insert the monitoring code into the script.

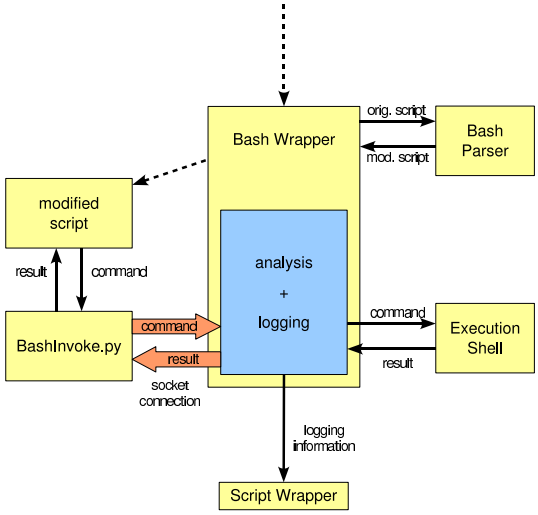

Figure 4: Bash Wrapper

Execution Shell The execution shell is the place, where all commands are actually processed. After the script wrapper has gotten the information, it writes an entry to R-GMA, that it is going to start the next command. After that, the wrapper executes the command in the execution shell and waits for its completion.

It is not possible to execute each command in an independent shell, that is closed after the command has finished. There has to be a shell kept open and running as long as the modified script is running. All commands have to be executed in the same shell. This is necessary, because the script may modify the environment, on which subsequent commands depend. If the shell is closed, the values of these variables are lost.

3.2.2 Python Wrapper

Supporting the Python programming language is quite important for users in high energy physics, as most software environments of the LHC experiments are Python based.

The Python Wrapper offers the same functionality as the bash wrapper. However, its implementation is much easier, as Python already offers debug methods, which are used by the Python Wrapper to examine the result of every command line. The success or failure of each command line is determined like in the *bash* and *sh* case.

3.3 Watchdogs and Pre-execution Tests

The watchdog component is responsible for monitoring the system resources, while the user script is being executed by the script wrapper. In a configurable regular time interval the watchdog publishes the current values of these resources to a table in R-GMA. Currently, the status of the file systems and the current memory usage as well as network statistics are monitored.

Before the script wrapper is started, several tests are performed on the compute node to check whether some needed services are available. Among these are tests of the R-GMA interface or accessability of some directories. Furthermore, several tests on the numerical quality [9, 10] of the results are performed.

4 Interactive steering of jobs

While the job monitoring supports the user with information about the environment of the job, this section describes the support given to the user for interactive monitoring and steering of the running job itself.

Once the job has started, it may produce useless results due to configuration faults of the job, or because the software installation has bugs. In other cases, the user may want to change parameters in order to improve performance or to explore a parameter realm. Without interactive steering, the user has to wait until the job has finished and then evaluate the results. Interactive steering can accelerate the research process, by providing earlier access to intermediate results and the possibility to modify parameters of running jobs.

The Result Monitoring and Online Steering Tool (RMOST) [11, 12] is build upon a model of distributed shared memory, to reduce modification requirements of existing applications and visualisation software. The goal is to provide a middleware which connects a local visualisation tool to remote Grid jobs. A general solution requires that the steering system does not need to know the type of the data it transports and synchronises. The marshaling problem is exported from the steering system to the application, or to a special data access layer. Most applications have already a serialisation method for their data for storing them on disk.

4.1 The Functionality

The functionality of RMOST is demonstrated by steering Grid jobs of the High Energy Physics (HEP) experiment ATLAS. In the ATLAS experiment a huge amount of data has to be evaluated and computed, which requires the use of a Grid. There exists an experiment software framework Athena [13] for computing the results. Athena is a modular framework which contains different kinds of components. The user creates a job description file (job options) for his job, where he specifies the components and parameters.

Numerous scientists around the world developed Athena components, thus modifications of core components for steering are hardly accepted by the HEP

community. A solution for the integration of steering into the Athena framework is to add additional components to the framework, that are sent with the job.

The intermediate results for visualisation are stored in so called ROOT files, which can reach a size of some GB. For the steering, these files must be accessed remotely. The user adjustable parameters are contained in the job options file. Thus, RMOST allows the modification of the job options file.

The ROOT toolkit [14] is used for visualisation of the physics results. Many scientists create own tools and components with ROOT, again changes to core components will be hardly accepted by the HEP community. To integrate steering into ROOT, the possibility of ROOT to extend its functionality with dynamically loaded libraries, was used.

For the steering the user must be able to create an interactive communication channel to the remote Grid job. This means firstly to find the job in the Grid. Then an interactive connection to the job has to be established, and finally, this connection must be secured. Hereby, some problems may occur:

After job submission the user retrieves a string, which identifies his job, but he does not know the target host, where the job runs. The host that executes the job, may be protected by firewalls or may be located in a private IP network, which inhibits a direct connection to the job. In addition the communication system must not compromise the target site's security nor allow unauthorised persons access to the site, and must ensure that only the submitter of the job can steer it.

In the ATLAS experiment the user typically submits hundreds of jobs in parallel. With this large number of jobs it is impossible to inspect each single job to control the execution of all jobs. Thus, a notification mechanism is provided, which automatically evaluates a condition and informs the user about suspicious jobs or on remarkable events. Furthermore, when a failure occurs and Athena terminates smoothly, the termination can be suspended for some time, the user is notified, and has the chance to inspect intermediate results or restart the computation with modified parameters.

4.2 RMOST Components

RMOST connects the user interface and the Grid job at runtime. Thus, RMOST provides functionality to access the data from the Athena framework and synchronises it with local data copies used by the user interface. To fulfil this task, a set of libraries and services are provided. In Fig. 5 an overview of the different components is shown.

On the side of the remote Grid job, the integration is realized by additional Athena components. The first component is called RM_Spy, which is applied by modifying the job options. With RM_Spy the following functionality is provided:

- Intermediate results stored in ROOT files can be accessed for visualisation.
- Online access to the log files.
- The number of executed events can be monitored.
- The job options file can be modified or replaced. The changes in the job options are applied by a restart of the job without resubmission.

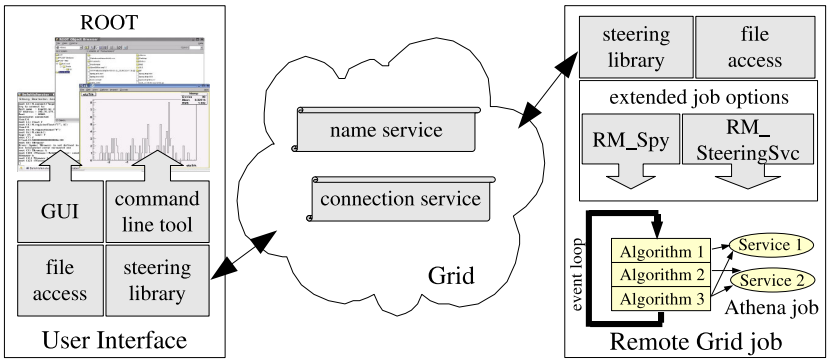

Figure 5: The components of RMOST

- The execution of the job can be suspended, terminated, or continued.
- The job can be executed stepwise.
- The job can be restarted without changing the job options.
- Optional notification of start and end of the execution.
- Optional notification on failure with delayed termination of the job for user interaction during the waiting period, enabling in-situ inspection.

The second component is RM_SteeringSvc, which supports steering functionality for customised components, including modification and monitoring of internal data, and setting of custom notification conditions.

Two additional components exist on both sides: the steering library and the file access library. The steering library manages the data exchange and the data consistency on user interface's and on the job's side. The data access library allows the synchronisation of data files on both sides, which the user interface and the remote job access like local files. The library catches local file accesses transparently to the applications and the data is accessed through the steering system. To use the file access utility, a shared library must be preloaded. Furthermore, the names of the files, of which the access should be caught, must be given to the file access library.

For the integration into ROOT, a graphical user interface is provided, which allows to connect to the job using the job identifier as address. A list of available data is presented along with the type of the data. For basic data types, also the values are shown and can be modified. Furthermore, the job execution can be controlled via the steering mechanism. Secondly, a command line API for ROOT has been developed, which allows steering from CINT, the C++ interpreter of ROOT, or from ROOT scripts or ROOT applications. A plugin of RMOST for the Grid interface GANGA is in preparation.

For establishing the communication channel between the user interface and the remote Grid job via the Grid, two additional Grid services are necessary. Firstly, a naming service, which maps a job identifier to its network contact information. For the naming service RMOST uses R-GMA [4]. Secondly, the

connection service, which is needed to connect to Grid jobs in spite of firewalls and private IP networks [11, 15].

5 Summary

With the tools presented we hope to give the scientists in LCG and - as gLite gets more and more widespread - also in other communities helpful and valuable support in their daily use of the Grid.

References

1. HEPCG. High Energy Physics Community Grid, 2005. http://www.hepcg.org.
2. D-Grid. The D-Grid Initiative, 2005. http://www.dgrid.de.
3. J. Novotny, M. Russell, and O. Wehrens. Gridsphere: A portal framework for building collaborations, 2005. http://www.gridsphere.org:80/gridsphere/gridsphere?cid=publications.
4. A. J. Wilson et al. Information and monitoring services within a Grid environment. In *CHEP 2004*, September 2004.
5. L. Field, F. Naz, et al. User level tools documentation, 2006. http://goc.grid.sinica.edu.tw/gocwiki/User_tools.
6. R. Alfieri, R. Cecchini, V. Ciaschini, Luca dell'Agnello, A. Frohner, K. Lörentey, and F. Spataro. From gridmap-file to voms: managing authorization in a grid environment. In *Future Generation Computer Systems*, volume 21-4, pages 549–558, April 2005.
7. JEM Project Webpage. http://www.grid.uni-wuppertal.de/jms.
8. A. Hammad, T. Harenberg, D. Igdalov, P. Mättig, D. Meder-Marouelli, and P. Ueberholz. A job monitoring system for the lcg computing grid. In *Proceedings of the 2006 20th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2006.
9. A. Frommer and M. Hüsken. Ensuring numerical quality in grid computing. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, Paris, June 2006.
10. M. Hüsken. Ensuring numerical quality in grid computing. In *12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics*, Duisburg, September 2006.
11. D. Lorenz, P. Buchholz, Chr. Uebing, W. Walkowiak, and R. Wismüller. Online steering of HEP Grid applications. In *Proceedings of the Cracow Grid Workshop '06*, Cracow, Poland, October 2006.
12. http://www.hep.physik.uni-siegen.de/grid/rmost.
13. European Laboratory for Particle Physics. *Athena Developer Guide*. http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Tech.Doc/Manual/2.0.0-DRAFT/AthenaDeveloperGuide.pdf.
14. R. Brun and F. Rademakers. ROOT - an object oriented data analysis framework. In *Proceedings of AIHENP'96 Workshop*, number A 389 in Nuclear Instruments and Methods in Physics research (1997), pages 81–86, September 1996.
15. D. Lorenz, R. Wismüller, P. Buchholz, Chr. Uebing, and W. Walkowiak. Secure connections for computational steering of Grid jobs. Submitted to the 8th IEEE/ACM International Conference on Grid Computing, September 2007.