A Job Monitoring System for the LCG Computing Grid

Ahmad Hammad[1], Torsten Harenberg[2], Dimitri Igdalov[1], Peter Mättig[2],
David Meder-Marouelli[2], Peer Ueberholz[1]

[1]Niederrhein University of Applied Sciences
Dep. of Computer Science
47805 Krefeld, Germany
{haah0001, igdi0001,peer.ueberholz}@hsnr.de

[2]University of Wuppertal
Department of Physics
42097 Wuppertal, Germany
{harenberg, meder, maettig}@physik.uni-wuppertal.de

Abstract

Experience with generating simulation data of high energy physics experiments has shown that a job monitoring system (JMS) is essential to understand failures of jobs within the Grid. Such a system can give information about the status of the user job as well as the worker node in parallel while a user job is running. It should support the user directly by allowing the user to interact with the running job and should be able to make an automatic error correction. Furthermore, such a system can be extended for an automatic classification of errors which can improve the stability and performance of the Grid environment. To increase the acceptance of the Grid, a graphical user interface (GUI) has been developed and integrated with the job monitoring system. Both components are currently integrated in the computing environment for generating data for the DØ Experiment. In this paper we want to describe the basic components of the job monitoring software.

1 Introduction

The increasing demand for computing power and storage is one of the major problems in many fields of science. Grid computing offers a possibility to solve this problem as long as the Grid software is stable and easy to use, jobs are running efficiently and reliably and users get good support in the case of errors. For experiments at the Large Hadron Collider (LHC) at CERN, the LHC Computing Grid (LCG) [1] is being built up, becoming the world's largest Grid installation with around 20000 computers at 180 participating institutes [2].

First experience with this software was gained by reprocessing data of the DØ Experiment [3, 4]. It was found that around 30% of the jobs failed. Therefore, there is a need for job monitoring in the worldwide heterogenous structure of LCG. This monitoring structure has to provide information about the current status of the jobs and – in case of failures – error messages as well as systematic information of the worker node acquired in parallel while the job is running. The current LCG monitoring systems works well until the job is being started. But afterwards it only distinguishes between "DONE (ok)" and "DONE (failed)" (see fig. 1) for finished jobs. If - even worse - the job was for any reason not considered being finished ("FAILED" or "ABORTED" status), the user has no access to any diagnostic job output (not even standard input and output).

There are many reasons for a job to fail, e.g. the network breaks down, not enough disk space, inadequate system library versions, time skews, or a simple script error made by the user. Unfortunately, in LCG a user doesn't have the opportunity to check the worker node on which his job is running interactively. Furthermore, LCG doesn't offer any central and systematic monitoring of the worker nodes.
Therefore, we decided to build up a monitoring system startable by the user by simply replacing a command line argument, with no need to change the LCG middleware. The system can

- provide information on the status of the job
- give information of the system resources on the worker node
- and give direct access to the worker node

1-4244-0054-6/06/$20.00 ©2006 IEEE

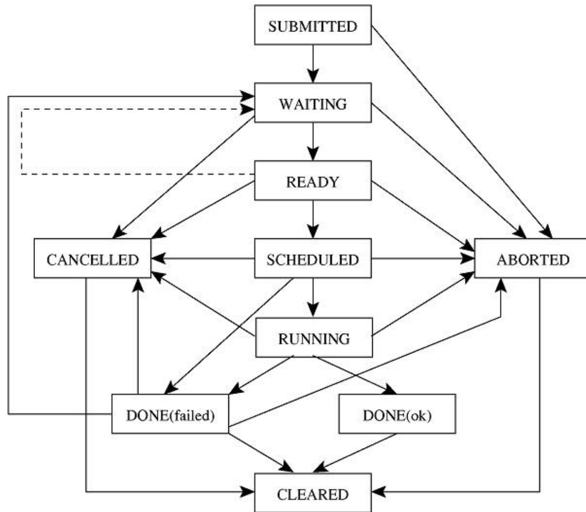

Figure 1. Job states in LCG-2. [5]

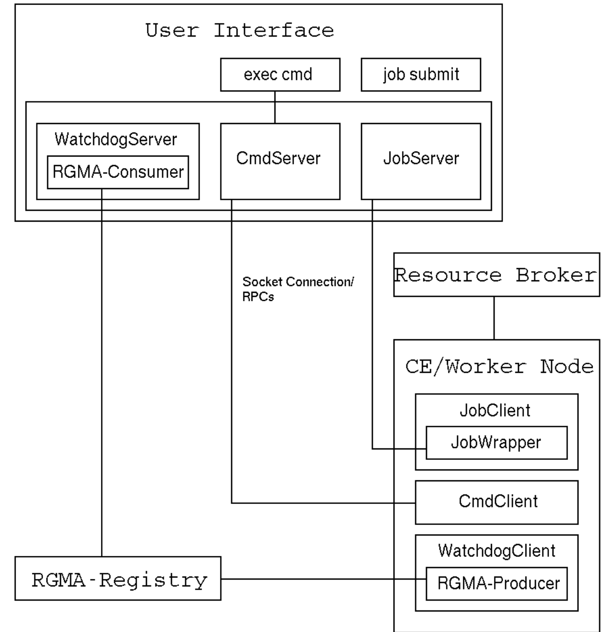

Figure 2. Communication structure of JMS

The JMS was designed in two parts; one part monitoring the worker node parallel to the job execution, and another which controls the execution of the user job step by step[1]. It is written in Python, which is part of the standard LCG-system, while GUI was developed in JAVA^{TM} Servlet Technology. It usually runs on a LCG user interface.

2 The Job Monitoring System

2.1 Worker Node Monitoring

The major aims of this main component of JMS on the worker node are to build up an information interchange channel between the worker node and the user interface, as well as to find system errors on the worker node (like not enough disk space, bad network connection, and many others [6]). These errors are unavoidable since there is always a delay between the decision of the resource broker of where to start the job, and the real starting time of the job. This delay can be – depending on the occupancy of the target's batch system – up to several minutes. In the meantime the system resources might have changed. The worker node is analysed frequently by JMS and both the system information as well as the output of the controlled job execution is forwarded to the user. Secondly, this component provides an interface for the user to execute commands directly on the worker node, so that he or she can get an impression of the runtime environment and the status of his or her job at any time.
The system is organized as a client server system with an asynchronous socket connection and remote procedure calls (RPC) between the worker node as client and the user interface as server (Fig. 2). The server is based on the *medusa* project [7], which is an architecture for building long-running, high-performance network servers in Python. Medusa runs as a single process, multiplexing I/O with its various client and server connections within a single process/thread.

The JMS has three client/server components running, one for executing commands on the worker node (CmdClient/CmdServer), one for controlling the job execution (JobClient/JobServer), and one for monitoring the system resources on the worker node parallel to the job execution (WatchdogClient/WatchdogServer).

- The JobClient/JobServer component: On the worker node the JobClient is responsible for the job execution monitor. It starts a job wrapper and collects the output of the job execution monitor during the execution of the job (see section 2.2). All output of the user job execution monitor is transfered by a socket connection via RPCs to the JobServer on the user interface while the user job is running. To distinguish between multiple jobs, every job has its own directory on the user

[1] only `bash` scripts at the moment

interface where all the monitoring data are stored. For future releases, these data should be handled by a data management system (for example, R-GMA).

- The CmdClient/CmdServer component: The Cmd-System, realized by a socket connection, allows the user to interact with the worker node running his user job directly. Using it, he can gain information of the status of the system and may influence the execution of the job, e.g., by changing runtime parameters or transferring data by hand. To start interaction with a worker node, the user has to start a simple command-line driven server side process which establishes the appropriate connection (marked `exec_cmd` in figure 2).
- The WatchdogClient/WatchdogServer component: Two ways for transferring the data of the WatchdogClient are realized: directly by the socket connection via RPCs, or over the R-GMA [8] database of LCG. The advantage of using R-GMA is that no information gets lost if the network connection to the user interface breaks down. On the other hand, two communication steps are needed and R-GMA is still under development. The history of the system resources on the worker node can be visualized by a RRDTOOL (Fig. 3) on the user interface and can be shown by a common browser like mozilla.

2.2 Job Execution Monitoring

The second component of the job monitoring system controls the execution of a user job line by line. It collects all information (command, input, output, return value, error information), which offers a simple possibility to analyze the reason for the failures [9]. To achieve this goal, a complete syntactic and lexical analysis of the job script has to be done. We wrote a shell debugger ourselves. The lexical analyses is realized with the help of PLY Python Lex, the syntactic analyses was done with a self written parser.

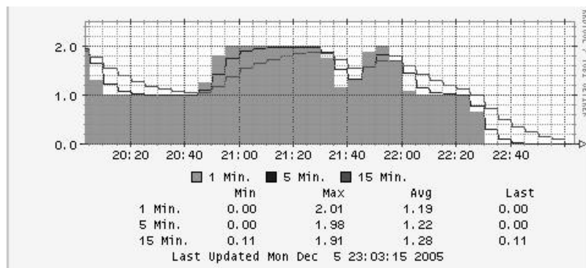

Figure 3. Monitoring the CPU usage

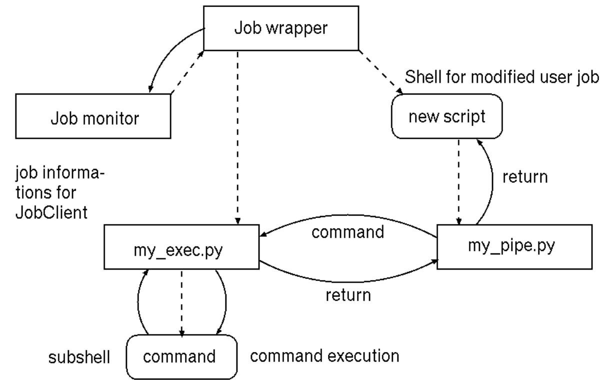

Figure 4. Script monitoring system, solid lines are communications, dashed lines are command calls

All commands of the job script are extracted and replaced by a python script called `my_pipe` taking a command including its parameters as argument (`''job wrapper''` in figure 4). This script (`''new script''`) is now being started by the `''job wrapper''` instead of the original one. Additionally, a python program called `my_exec.py` is started by `''job wrapper''`. This python tool keeps a persistent shell (marked "subshell") open. This ensures that the environment variables are kept during execution exactly as would be the case when running the original user's script. Similarly, the output of the executed commands are returned to modified user job.

The following information of every step of the job script is processed:

- **before execution:** line number, command and arguments, PID
- **after execution:** line number, command, standard output and error, exit code.

This information is collection in a Python dictionary and passed during runtime to the JobClient component of the worker node monitoring system, which is forwarding it to the user. If an error occurs, e.g. the network doesn't work correctly, a direct reaction can be integrated by changing the original job script so that the job can continue. A simple example would be to repeat a gridFtp-command until it is successful or a maximum number or retries is reached. Another possibility would be to stop the job and to wait for a user in-

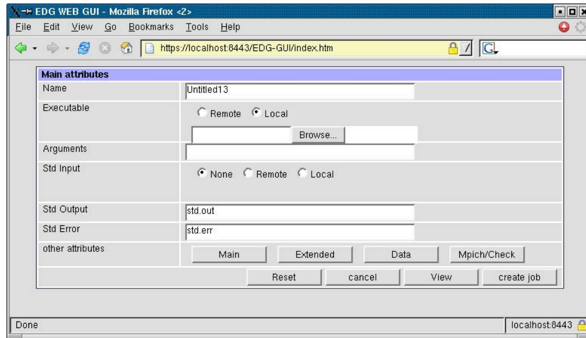

Figure 5. Graphical User Interface for LCG

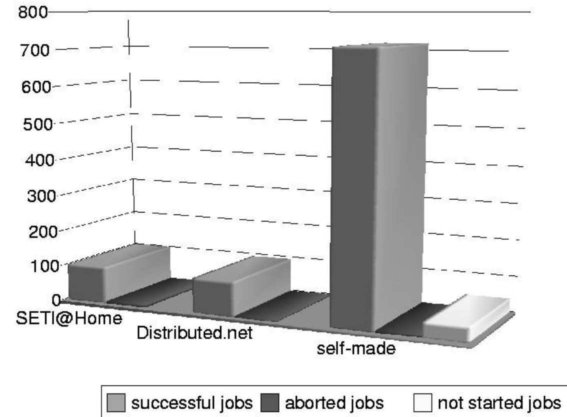

Figure 6. JMS-Tests

put, which can be done by the CmdServer/CmdClient component of JMS.

In case of an unknown script command or a script problem, the job wrapper is stopped and the script is executed without monitoring.

2.3 Graphical User Interface (GUI)

Currently we are integrating the JMS into the graphical user interface (Fig. 5) we have developed. The aim of the GUI was to make LCG more user-friendly and to offer a comfortable working environment to manage many user Jobs [10], a use case which occurs, for example, while generating simulation data for physics experiments.

The GUI allows the user to submit jobs to LCG from any computer without the need of installing a user interface. Nearly the complete functionality is offered by the GUI. The users are lead in a few steps to a complete job description or can include their own JDL files[2]. An integrated user administration allows the user to stop and continue his work on any computer. All data operations (upload and download, replicatation, deletion, ...) can be managed within the Grid. Currently we are integrating the JMS so that one can choose different monitoring levels depending on the amount of information a user would like to get.

3 Preliminary Test Results

To test the JMS, nearly 1,000 jobs were started in the LCG-2 environment. Different job types were tested with these results (Fig. 6):

- **110 `SETI@HOME` jobs**
  103 run successfully,
  7 failed with "ABORTED". The Job Output could be saved, which would have been lost otherwise.
- **100 Distributed.Net jobs**
  97 run successfully,
  3 failed and the errors were successfully recognized by the JMS.
- **765 self-made test jobs**
  These jobs have produced well defined errors, which had to be caught by the JMS.
  718 ran successfully, all errors were caught,
  47 failed on LCG with "ABORTED" status, but the output from 6 of them was recovered. 41 were not started at all.

These tests show that the overhead of the JMS is less than 0.1 sec per `bash` script line. Although this is already acceptable we're working on improving this performancs.

The LCG user is now able to get real time information about his running jobs. If jobs have been started by the LCG, the output of aborted jobs is recovered by the JMS. The parser worked fine in all cases and reported the included errors in our self-made tests jobs correctly. We could already classify several problem, which appeared frequently:

- Misconfigurations in local batch systems,
- several jobs were marked as failed, although exit 0 status has been thrown at the end of the job,
- jobs marked been successful, although exit status was not zero (happened with `SETI@HOME` jobs, when the central server in Berkeley was down).

[2] JDL = Job Description Language. The language to describe job parameters in LCG-2

Another advantage of the JMS is the fact that errors are reported directly during runtime. This opens the opportunity to recover certain classes of failures (with the `CmdClient/CmdServer` component), thus saving a lot of computing time.

4 Outlook and Conclusions

The presented Job Monitoring System is work in progress. Several technical enhancements are planned or being developed. Especially the TCP driven socket connection has to be replaced by secure, reliable, Grid conforming methods and protocols. The components affected are discussed now in detail:

- **JobClient/JobServer**
  For future releases, we plan to migrate the communication completely to R-GMA. This would solve possible problems using socket based connection to a single-side server system, which may be limited and/or overloaded by the number of concurrent connections. Furthermore, all the information transfer would be encrypted and authenticated by the user's certificate automatically. Another advantage would be that no persistent network connection is established.

- **CmdClient/CmdServer**
  This component is critical, since in the current state no encryption and authentication is used. Anyhow we expect that in near future all Grid systems will block insecure network transfer like this. R-GMA seems to be unreliable for a real-time application like this. So we are going to use gsissh [11] for the interactive component.

- **job failure classification**
  In the future we plan to implement an automatic analysis of the JMS output identifying and classifying the different job failures. An expert system should be developed to recover certain classes of these errors during runtime, thus improve performance and stability of the Grid system and efficiency of the CPU usage.

The current version of the job monitoring system is running stable and the integration in the GUI is nearly finished. We already get an impression of the errors occurring during job execution by using the system with real data from the DØ experiment. A systematic analysis is in progress.

References

[1] http://lcg.web.cern.ch/LCG

[2] http://goc.grid-support.ac.uk/gridsite/monitoring/

[3] T. Harenberg *et al.*,"DØ data processing within EDG/LCG,"; FERMILAB-CONF-04-477-CD; *Prepared for Computing in High-Energy Physics (CHEP '04), Interlaken, Switzerland, 27 Sep - 1 Oct 2004*

[4] T. Harenberg, K. H. Becker, W. Rhode and C. Schmitt, "AMANDA - first running experiment to use GRID in production," eConf **C0303241**, MOAT010 (2003)

[5] A. Delgado Peris et. al, LHC Computing Grid: LCG-2 User Guide, Manuals Series, CERN-LCG-GDEIS-454439

[6] A. Hammad, master thesis, http://www.grid.uni-wuppertal.de/jms

[7] http://www.nightmare.com/medusa

[8] The EGEE project, Information and Monitoring Service (R-GMA) – System Specification, EGEE-JRA1-TEC-490223-R_GMA_SPECIFICATION-v2-0

[9] D. Igdalov, diploma thesis, http://www.grid.uni-wuppertal.de/jms

[10] D. Vogel, diploma thesis, http://www.grid.uni-wuppertal.de/webgui

[11] http://www.globus.org/toolkit/docs/4.0/security/openssh/