PROCEEDINGS OF SCIENCE

Overview over the High Energy Physics Community Grid in Germany's D-Grid Initiative

Stefan Borovac[a], Torsten Harenberg[*a], David Meder-Marouelli[a], Peter Mättig[a], Markus Mechtel[a], Ralph Müller-Pfefferkorn[b], Reinhard Neumann[b], Thomas William[b], Peter Buchholz[c], Daniel Lorenz[c], Christian Uebing[c], Wolfgang Walkowiak[c], Roland Wismüller[c], Peer Ueberholz[d], Anar Manafov[e], Robert Manteufel[e], Victor Penso[e], Carsten Preuss[e], Kilian Schwarz[e], Günter Duckeck[f], Johannes Elmsheuser[f], Tariq Mahmoud[f], Dorothee Schaile[f], Patrick Fuhrmann[g], Stefan Freitag[h], Lars Schley[h]

[a] *Bergische Universität Wuppertal, Gaußstraße 20, D-42119 Wuppertal, Germany*
[b] *Center for Information Services and High Performance Computing, Technische Universität Dresden, D-01062 Dresden, Germany*
[c] *Universität Siegen, Emmy-Noether-Campus, D-57068 Siegen, Germany*
[d] *Hochschule Niederrhein, Reinarzstraße 49, D-47805 Krefeld, Germany*
[e] *Gesellschaft für Schwerionenforschung mbH (GSI), Planckstraße 1, D-64291 Darmstadt, Germany*
[f] *Ludwig-Maximilians-Universität München, Sektion Physik, Am Coulombwall 1, D-85748 Garching, Germany*
[g] *Deutsches Elektronen-Synchrotron (DESY), Notkestraße 85, D-22607 Hamburg, Germany*
[h] *Institut für Roboterforschung (IRF), Otto-Hahn-Straße 8, D-44227 Dortmund, Germany*

Today, one of the major challenges in science is the processing of large data sets. Experiments or simulations can produce an enormous amount of intermation that is 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 limited fragments of the data and process them in parallel on a Grid. Examples of such a scenario are the processing of images in medicine, gene alignment in bioinformatics, or the analysis of high energy physics data. The Large Hadron Collider (LHC) at CERN will provide particle physicists with several petabytes of experimental 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. The High Energy Particle Physics Community Grid project (HEPCG) of the German D-Grid Initiative wants to contribute to the functionality that the LCG provides to users in the areas of distributed data management, job monitoring and distributed data analysis.

XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research
April 23-27 2007
Amsterdam, the Netherlands

[*] Speaker.

PoS(ACAT)025

The High Energy Particle Physics Community Grid project (HEPCG) (*funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01AK802C*) of the German D-Grid Initiative wants to contribute to the functionality that the LCG provides to the users.

It is seperated into three work packages which cover the following areas:

- Work Package 1: Distributed Data Management
  The most demanding experiment of this decade, concerning storage and distribution of data, is certainly the Large Hardron Collider at CERN, which will begin its production service in the midddle of 2008. The LHC will produce a sustained stream of data in the order of 300MB/sec, which needs to be distributed and persistently stored at several dozens of sites around the world, known as the LHC data grid. Knowing that the WLCG collaboration will have solved most of the issues related to such a challenge, other worldwide operating groups try to profit from middle-ware used in this area, such as the "International Lattice Data Grid". The purpose of Work Package 1 in HEPCG is to cover a wide range of topics in the data management area and to provide solutions ready to be used. This ranges from a flexible and customizable metadata catalog to manage worldwide distributed data, up to a highly scalable Storage Element, capable of storing data in the peta byte range. Moreover, we are offering a solution on how to use storage location and data retention information in order to optimize grid job distribution among the available grid compute systems worldwide.

- Work Package 2: Job Monitoring
  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, or are only simple command line tools flooding the user with textual information. One major goal is to improve the user-centric monitoring of jobs, their execution, and their resource usage.

- Work Package 3: Distributed Data Analysis
  Distributed data analysis using Grid resources is one of the fundamental applications in high energy physics to be addressed and realized before the start of LHC data taking. In every experiment up to a thousand physicist will be submitting analysis jobs into the Grid. Appropriate user interfaces and helper applications have to be made available to assure that all users can use the Grid without too much expertise in Grid technology. These tools enlarge the number of grid users from a few production administrators to potentially all participating physicists.

Some highlights of the work that has been done inside the workpackage:

WP 1: Distributed Data Management

The central component for storaging and providing data is the storage element. Though slightly differently defined for the various data grids it mainly has to be able to store data in the range from some Tbytes into the peta byte level. In order to provide the necessary management

and transfer services, a Storage Element has to implement a briefly defined set of protocols. This is a wide area transfer protocol, essentially gridFtp (gsiFtp), a data management protocol, which has been agreed upon to be the Storage Resource Manager Protocol (SRM) and information protocols which may differ between data grids. A major player in the set of these Storage Elements is the dCache technology, which has proven to be capable of storing and exchanging several hundreds of tera bytes of data, transparently distributed among dozens of disk storage nodes and connected tape system back ends. One of the key design features of dCache is that although the location and multiplicity of the data is autonomously determined by the system based on configuration, cpu load and disk space, the name space is uniquely represented within a single file system tree. Being an accepted Storage Element already implies that the basic protocols mentioned above are supported by dCache. In addition to these well established protocols we designed and implemented an enhanced mechanism to provide information to enable Work Load Management systems to query Storage Elements on access latency and retention policy of data. This should significantly improve the job throughput for data intensive applications. In addition to the required functionality of a storage element, this paper will report on extended features of dCache, making the technology unique. This includes the high performance name space system, Chimera, which provides full ACL control to the stored data as well as a subproject, adding full NFS 4.1 functionality to the dCache Storage Element.

1.1 Co-Scheduling of jobs and data

The effort for co-scheduling of jobs and data is based on the currently available gLite middleware stack. Herein a job is submitted by the user to a Work Load Management system which forwards the job in respect to its requirements to a Compute Element. Close to a Compute Element usually one or more Storage Elements reside which contain TBytes of data. To support long-term storage of data, Storage Elements do not only consist of disk storage nodes. They are extended by tape backends to Hierarchical Storage Management (HSM)-based systems.

The state information for data in such a HSM-based system can not always be used properly. For example data may appear "available" on the Storage Element, but is located on tertiary storage. Before it is accessible and therefore really available to a user it has to be staged from tape to disk storage nodes. This action can take several minutes up to a few hours. So, a job would greatly benefit if the data it needs is made available in advance.

To offer advancded planning and above mentioned co-scheduling the gLite Work Load Management system, Compute Element and the dCache Storage Element are enhanced [1]. On the one hand the Extended Information Service (EIS) is added to dCache. It enables queries for the state of data in terms of e.g. availability (on-line or near-line), file size or estimated staging time. On the other hand the decision making process of the Workload Management system is modified in the way that it uses information provided by EIS.

In doing so, a job with data requirements can be scheduled to a Compute Element where most of the data is on-line on close Storage Elements. Additionally, for near-line data a staging process can be initiated as the job passed the Work Load Management system. In previous approaches the staging process started after the job reached a Worker Node, resulting in a degraded performance, because the job had to wait for data access until the staging process completed.

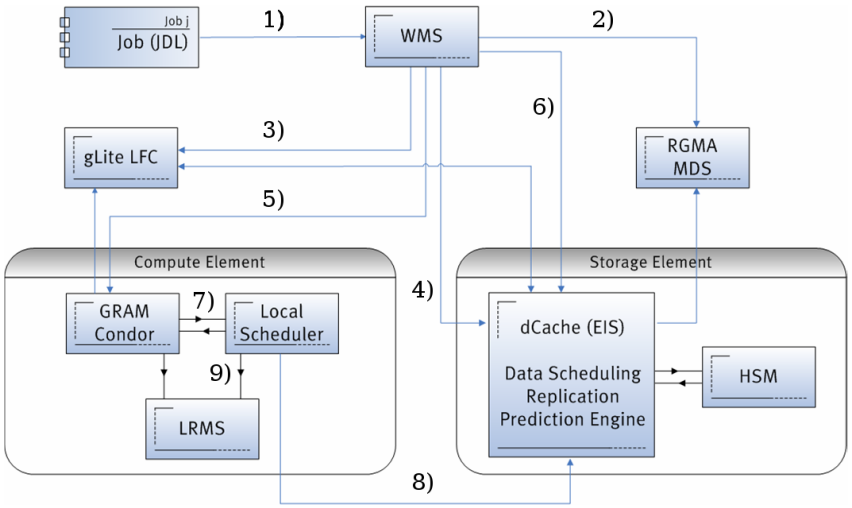

Figure 1: Architectural overview for co-scheduling of jobs and data

Figure 1 describes the job flow from the User Interface up to the Batch System. A job in form of a JDL is submitted by the user to the Work Load Management system. This system posseses information about all available Compute and Storage Elements. If the job contains data requirements specified by logical file names (LFNs), those are resolved to physical file names (PFNs) and GUIDs (Global Unique Identifiers) through the LCG File Catalog (LFC). The Workload Management system acts as EIS client and queries suitable (in respect to supported data protocols like gsiFtp or SRM) Storage Elements for the needed PFNs. Depending on the data availabilty and estimated staging time a Compute Element nearby the best fitted Storage Element will be choosen. After doing so, the job is forwared to the Compute Element where it is set in hold state. A local scheduler service on the Compute Element periodically checks for the different held jobs if the needed data is available or not. In parallel, if needed data is only near-line on the selected Storage Element a request will be submitted to bring the data online. After all data for a job is available online the job state is changed from hold to running by the local scheduler on the Compute Element. The enhancements in steps (4),(6), (7) and (8) help to avoid jobs idling on Worker Nodes and to increase better utilization of the Batch System.

WP 2: Job Monitoring

2.1 User-centric monitoring of jobs and their resource usage with AMon

The user-centric monitoring AMon program delivers the user with information about the status and the resource usage of submitted jobs. Such information includes: 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? Is there still enough space on the disks I use? This is information a user is accustomed to having to when running an application on a desktop computer. Furthermore, they can indicate problems or an untroubled run of the user application.

In this context we address two types of "users": the submitters of the jobs, who want to know what is going on with their jobs, and the (hardware) 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 of use 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 allows interaction to get further and more detailed information

3. Authentication, authorisation, and secure data transmission for data privacy reasons

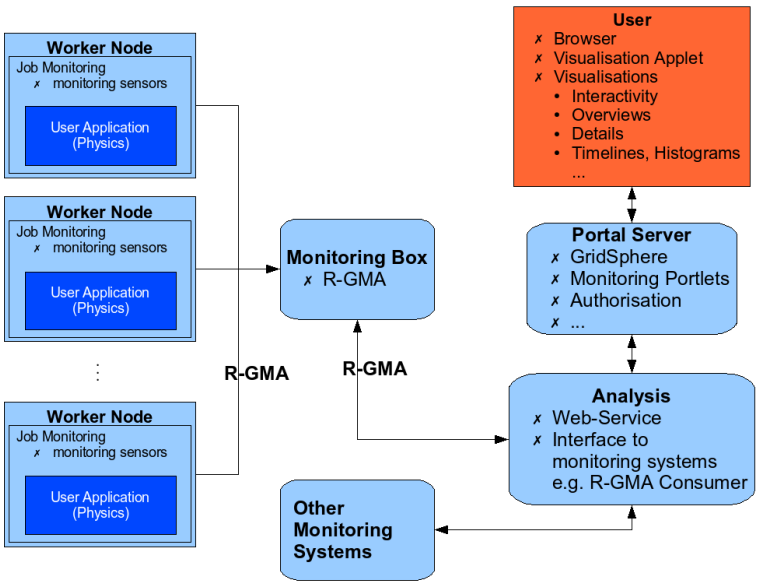

Figure 2: The AMon architecture

The AMon system consists of four components: information collection on the nodes where the jobs run, information storage, information analysis, and the user interface.

The component to collect the monitoring information is based on the LCG worker node monitoring [2]. It was extended to collect more useful metrics. The current set of monitoring metrics is listed in Table 1. On the LCG/gLite worker node a monitoring daemon is started in parallel to the execution of the job. The monitoring data are stored in R-GMA, the Relational Grid Monitoring Architecture [3], which is used in gLite to publish monitoring data. R-GMA is a relational distributed database. Data are managed and defined in tables, which are accessible via SQL query statements. Programs referred to as producers publish data to R-GMA. A consumer asks the R-GMA registry for the producers of a table, is directed to the (distributed) producers, and then gets the data directly from them. The analysis of the data is implemented as a web service. It gathers the distributed monitoring data from R-GMA, pre-analyses them and makes them available.

The user interface is the users favorite web browser. He/she accesses a web portal which provides access to the monitoring system. The portal is based on the GridSphere portal technology.

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	real, virtual, total, and free memory; free and total swap space
Storage	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
Output	last lines of user application output files

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

When the user wants to see the monitoring information about her/his jobs, a portlet contacts the Analysis Web Service, which gathers the data and transfers them back to the portlet. The visualisation is handled by Java applets which run within the portlet. The Java applets provide graphical displays to the user to summarise the status of the jobs, give detailed information about the resource usage of single jobs, hint at possible problems in the execution of jobs, and more. Interactivity allows the user to zoom into the displays, click on jobs to get more detailed information or to get other displays. Figure 3 shows some screenshots of currently avaliable displays within their portal environment.

Due to data privacy reasons, access to the data is restricted according to authorisation rules. This is done by contacting a VOMS system [4] which allows one to authorise users with their Grid credentials. Currently, three authorisation levels are established:

1. user: has access to his/her own data only
2. resource provider: can view the data of all jobs running at his/her site
3. VO manager: can access all data of his/her VO

In November 2006 the first prototype of the complete system was finished, and is now under intensive testing within the project. Ongoing and future work will cover algorithms to analyse the data to find problems in the job execution, new and improved visualisations, the usage of other information systems, and more.

We hope that with the AMon system users are enabled to better understand what is going on with their jobs in the Grid and to help them in finding possible problems or bottlenecks.

2.2 Monitoring the execution of jobs – the Job Execution Monitor

Existing job monitoring tools in the LCG/gLite environment provide limited functionality to the user, e.g., the LCG job monitoring tools. These are either command line tools which deliver simple text strings for every job or the information they provide is very limited (e.g., status only).

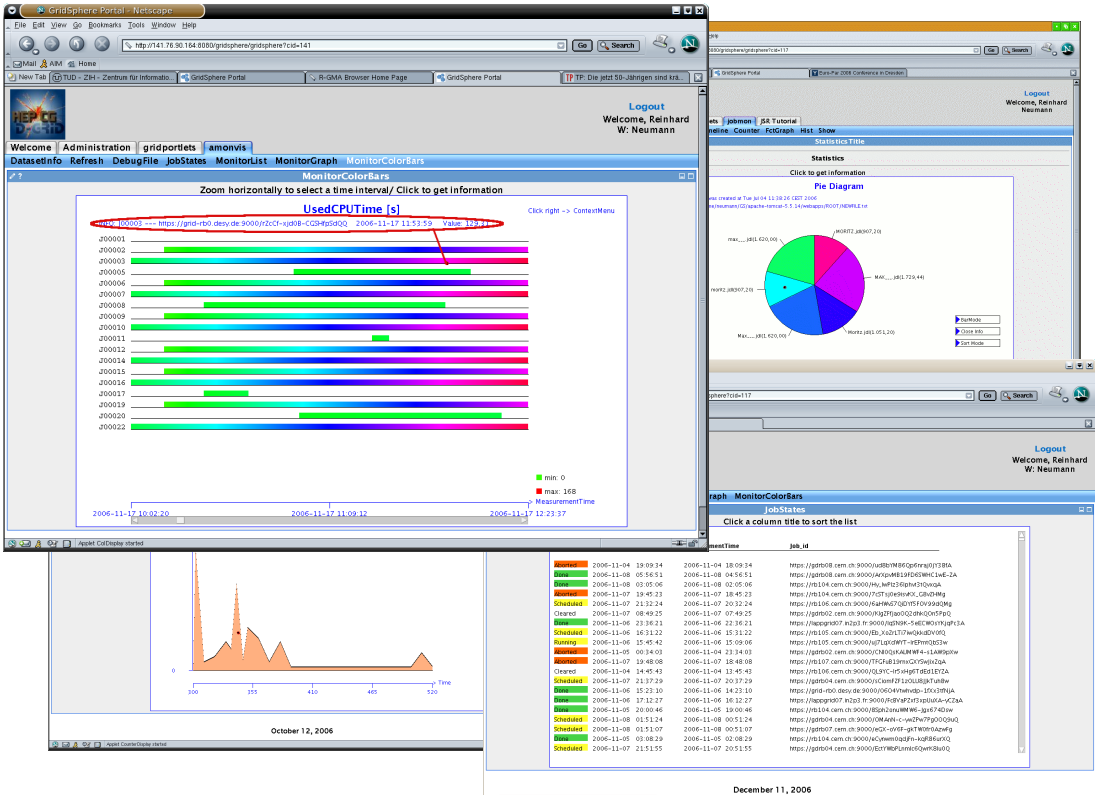

Figure 3: Screenshots of the graphical displays whithin the portal

For example, LCG/gLite distinguishes the status “ok” and “failed” for a finished job, although the status of a job is very well known until it is actually started.

Other existing monitoring tools (e.g., the LCG Real Time Monitor, or GridIce) focus on the monitoring of the infrastructure rather than the user application/job.

In contrast to these concept, we developed the "Job Execution Monitor", a Python based software project, which monitors the execution of script files within the user jobs. In addition, it inspects system parameters which could be important for classifying failures.

Our design goals are:

- connectivity realised via R-GMA (standard gLite component, no additional communication channel required).
  - runs on all Grid middleware systems using R-GMA.
  - information is passed to the user interface online.
  - user is informed nearly in real time.
  - debugging information is not lost even if the output sandbox cannot be retrieved by the Grid middleware.
- monitoring down to the source code level (several verbosity levels possible).
- other resource monitoring systems on the worker node may be used in addition.

- doesn't need additional setup by the site administrator.
- no changes to user code is neccessary.
- supported languages until now: bash and python.
- detailed log files available.
- support for the ATLAS software framework *Athena*.

Typically, the first thing to be executed on a worker node is not a binary executable, but a script file which sets up the environment first. This step includes, e.g., the setup of some environment variables or the download of data files from a storage element. The setup process often includes commands which are known to be the source of many job failures. It is the goal of the Job Execution Monitor to monitor the execution of such critical commands and report their success or failure to the user.

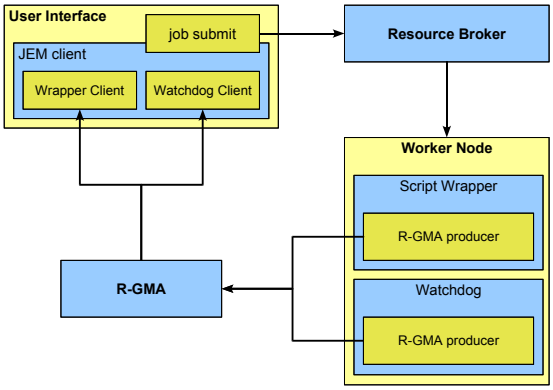

Figure 4: Basic design of the Job Execution Monitor

The basic design of the Job Execution Monitor is shown in Figure 4. The core module of the Job Execution Monitor is the script wrapper. It is responsible for loading the language wrapper modules for the supported languages. In order to gain detailed information about the job execution, a given script file is executed command by command. We are developing a new version, which makes changes to the bash script unneccessary. In case of bash scripts, we developed a parser which analyses and modifies the script, as bash does not provide this important function. This script is then executed line by line in a seperate shell, which is kept open during the execution. See Figure 5 for an overview over the application flow.

After a command has finished, its exit status is checked and logged. The script wrapper publishes the logging data to the Relational Grid Monitoring Architecture (R-GMA), which is used in LCG/gLite to store monitoring data. The current version of the Job Execution Monitor is able to monitor the execution of bash and python scripts.

While the monitoring part is rather complete, we are currently developing an expert system that tried to classify error conditions using the information gained by the monitoring software described above as well as external information, for example the Service Availability Monitoring

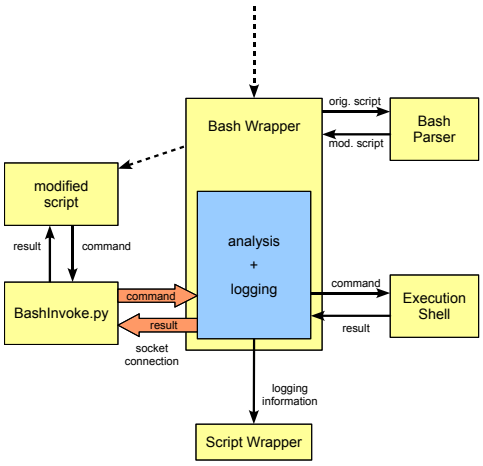

Figure 5: Application flow of the bash script monitor running on a worker node

system of LCG/gLite [5]. New developments are announced on the homepage of the Job Execution Monitor [6]. Integration into the Ganga job management system (see section 3.2) is in progress.

2.3 Interactive steering of jobs

Science based on Grid computations, like in the HEP experiment ATLAS, can be accelerated by shortening the delays between job submission and access to the first results. This is targeted by online steering, which provides for the monitoring of intermediate results of the running job, as well as online interaction.

RMOST (Result Monitoring and Online Steering Tool) [7],[8] is a computational steering tool which is designed to interactively steer Grid jobs of the ATLAS experiment. It is integrated into the ATLAS experiment software Athena [9] and the ROOT [10] toolkit which is used for visualization. The main functionality of RMOST can be applied to an ATLAS Grid job without modifying the source code of the ATLAS experiment software. Advanced possibilities, which require some source code instrumentation, exist for expert users. The most important RMOST features are:

- Establish a secure connection to the remote Grid job.
- Monitor intermediate results and standard output.
- Modify the job parameters.
- Send notifications on important job events.

2.3.1 Integration into ATLAS experiment software

Typically, a job in the ATLAS experiment processes a large number of events which are independent of each other. This class of jobs is supported by the Athena framework, which consists of a collection of algorithms and services. When preparing an Athena job, the user composes a list of Athena algorithms which is executed for each event. Athena services provide important functionality to these algorithms and other components. Thus, an Athena job is fully described by the

composition of algorithms and services, which is stored in a job options file. The output of the job is continuously written to a file in a format suitable for the ROOT visualization toolkit.

The Athena framework can be extended with additional components by using shared libraries. RMOST uses this mechanism to provide additional Athena components which integrate steering into the Athena framework (See Fig. 7).

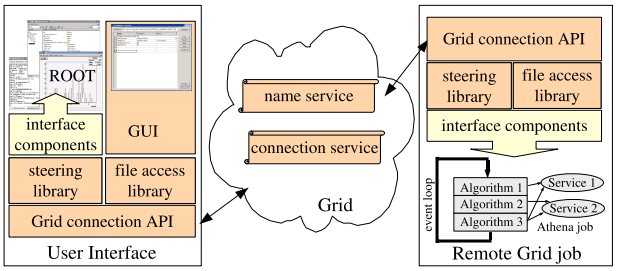

Figure 6: RMOST connects a user interface to a running job

The first component is the Athena algorithm `RM_Spy`, which contains the basic functionality. It can be used within a job by just changing the job options and supports the following actions:

- Online access to intermediate results of the Grid job, which are stored in ROOT files. `RM_Spy` ensures that all results generated so far are flushed to the ROOT file, and the file is then properly completed in such a way that it can be correctly visualized with the ROOT toolkit.
- Monitoring of the job's log files and standard output.
- Control of the execution of the job with the possibility to suspend execution, to execute the job stepwise, to restart the job without resubmission, and to terminate the job.
- Modification of the job parameters during runtime, or replacement of the job options file. The changes are applied via restarting the job on its local host without resubmitting it to the Grid.
- Optional notification on start and end of job execution. Notifications can be sent via the secure Grid connection established by RMOST and will then be displayed on the user interface. Alternatively, they can also be sent by email. In this case, `sendmail` must be configured on the host running the connection service which is required for establishing the Grid connection.
- Optional delay when the job terminates due to a failure in the Athena framework, typically caused by an incorrect configuration in the job options file. During this delay, the user can interact with the job in order to detect the cause of the failure.

The second Athena component is the Athena service `RM_SteeringSvc`. It supports the following features:

- Monitoring and modification of arbitrary data with user defined data types.
- Access to data in main memory.

- Application of modifications without a restart.
- User defined notifications on user defined conditions.

The `RM_SteeringSvc` can be used by any other component to customize the steering possibilities of this component. For using the `RM_SteeringSvc`, the source code of the component must be instrumented with calls to the steering service.

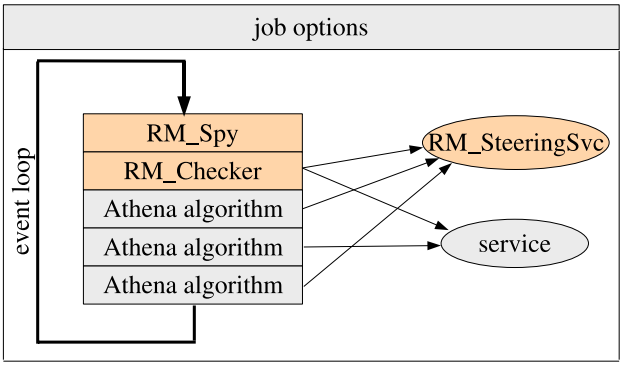

Figure 7: Integration into the Athena framework

2.3.2 User interface of RMOST

RMOST provides an interface to the ROOT toolkit [10], which provides for the steering of one or several jobs and the visualization of intermediate results from within ROOT (see Fig. 8). The Grid job can be accessed from the ROOT command line and via a graphical user interface. The GUI supports all actions of `RM_Spy`, while the ROOT command line extension also supports the whole functionality of the `RM_SteeringSvc` service. The RMOST interface can be dynamically loaded into ROOT. An interface to the Ganga [11] front end is currently being developed.

With these features of RMOST, scientists can use the ATLAS software on the Grid in a comfortable, interactive way, similar to its use on a local workstation.

WP 3: Distributed Analysis

3.1 The RGLite plugin for ROOT

A ROOT plugin, RGLite, for the gLite middleware has been developed. By using this plug-in, ROOT users are able to submit jobs to a gLite flavored Grid, query the job status, and perform a range of file catalogue operations. Currently the next stage of the project is on the way, during which a prototype of a data analysis schema and tools will be created. The schema will offer users the possibility to process a distributed data analysis using RGLite and PROOF on a gLite Grid.

3.2 Distributed data analysis in ATLAS/LHCb using Ganga

An analysis job in the computing environment, Athena, at the ATLAS experiment [12] will typically consist of a Python or shell script that configures and runs a user algorithm in the Athena framework, reading and writing event files, and/or filling histograms /n-tuples. More interactive

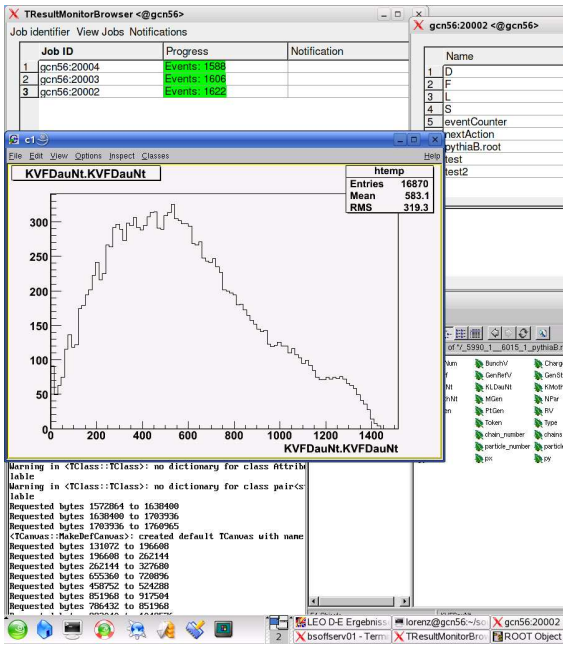

Figure 8: The graphical user interface of RMOST

analysis may be performed on large datasets stored as n-tuples. The distributed analysis system must be flexible enough to support all work models depending on the needs of a single user or an analysis team. A distributed analysis system should be robust and easy to use for all collaboration members. The look and feel of the system should be the same whether one sends a job to one's own machine, a local interactive cluster, the local batch system, or the Grid.

There are several scenarios relevant for a user analysis:

- Analysis with fast response time and a high level of user interaction.
- Analysis with intermediate response time and interaction.
- Analysis with long response times and a low level of user interaction.

The first point is well matched by the parallel ROOT facility PROOF [13] for interactive usage and fast turn around times on a local computing cluster. For the second and third point an automatic job manager and scheduler in an distributed analysis environment is the key feature for a robust system.

The Ganga job management system [14] [15], developed as a common project between the ATLAS and LHCb experiments, provides and integrates these kind of tools. Ganga provides a simple and consistent way of preparing, organizing, and executing analysis tasks within the experiment analysis framework, implemented as a plug-in system. It allows trivial switching between running test jobs on a local batch system and running largescale analyses on the Grid, hiding Grid technicalities. The integration and interaction with the ATLAS data management system DQ2/DDM [16]

into Ganga is a key functionality. In combination with the job splitting mechanism, large numbers of analysis jobs can be sent to the locations of the necessary data files, as required by the ATLAS computing model. Ganga supports tasks of user analysis with reconstructed data and small scale production of Monte Carlo data.

Figure 9 shows the work-flow of a job scheduler. A user provides an application and its configuration before job execution. The dataset to be processed is queried for its location and contents in the DQ2/DDM data management database. Depending on the size of the dataset the Grid job is divided into several sub-jobs that are executed in parallel and are only processing a subset of the input dataset. During execution jobs are monitored. At job completion the results and output files are stored as an output dataset with a reference in the DQ2/DDM data management system database. Afterwards, the user can retrieve the output by simple means.

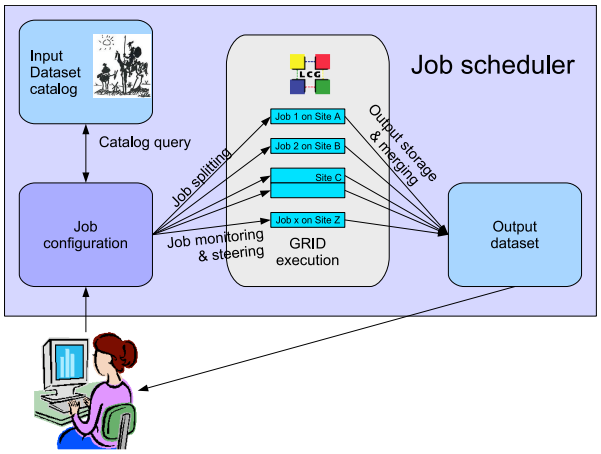

Figure 9: Outline of a job scheduler work-flow. A detailed explanation is given in the text.

We have successfully extended the functionality of Ganga in numerous areas. The most important are the extension of job splitting for ATLAS jobs and the integration of the ATLAS data management system DQ2/DDM with direct access to the input data files via POSIX I/O [17]. These items are crucial, since only with an intelligent job splitting and parallelization and a robust data management system can a successful distributed analysis system be built.

Ganga has performed well in all tests of configuring, submitting, monitoring, and output retrieval of a few hundred jobs. We have used Ganga on LCG/EGEE infrastructure at several Tier 1 and Tier 2 sites.

References

[1] `http://isaac.e-technik.uni-dortmund.de/trac/projects/hep_c g`.

[2] L. Field, F. Naz et al. User level tools documentation, 2006. http://goc.grid.sinica.edu.tw/gocwiki/User_tools.

[3] A. J. Wilson et al. Information and Monitoring Services within a Grid Environment. In *CHEP 2004*, September 2004.

[4] R. Alfieri, R. Cecchini, V. Ciaschini, Luca dell'Agnello, A. Frohner, K. Lörentey, 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.

[5] Production SAM portal:
`https://lcg-sam.cern.ch:8443/sam/sam.py`.

[6] Job Execution Monitor Homepage:
`http://www.grid.uni-wuppertal.de/jem`.

[7] Daniel Lorenz, Peter Buchholz, Christian Uebing, Wolfgang Walkowiak, , Roland Wismüller. Online steering of HEP Grid applications. In *Proceedings of the Cracow Grid Workshop '06*, Cracow, Poland, October 2006.

[8] `http://www.hep.physik.uni-siegen.de/grid/rmost`.

[9] CERN European Laboratory for Particle Physics. Athena Developer Guide.

[10] Rene Brun, Fons 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), September 1996.

[11] Ganga: a Grid user interface. In Sunanda Banerjee, editor, *Proceedings of XV International Conference on Computing in High Energy and Nuclear Physics (CHEP 06)*, pages 982–985, Mumbai, India, February 2006.

[12] ATLAS Computing TDR, CERN-LHCC-2005-022.

[13] ROOT and PROOF project web page:
`http://root.cern.ch/`.

[14] GANGA project web page:
`http://ganga.web.cern.ch/ganga/`.

[15] GANGA: a Grid user interface. In Sunanda Banerjee, editor, Proceedings of XV International Conference on Computing in High Energy and Nuclear Physics (CHEP 06), pages 982-985, Mumbai, India, February 2006.

[16] ATLAS DQ2/DDM project wiki page:
`https://twiki.cern.ch/twiki/bin/view/Atlas/DistributedDataManagement`.

[17] Portable Application Standards Committee project web page:
`http://www.pasc.org/plato/`.