Journal of Physics: Conference Series **119** (2008) 062034 doi:10.1088/1742-6596/119/6/062034

Experimental Evaluation of Job Provenance in ATLAS Environment

Aleš Křenek[1], Jiří Sitera[1], Jiří Chudoba[1,2], František Dvořák[1], Jiří Filipovič[1], Jan Kmuníček[1], Luděk Matyska[1], Miloš Mulač[1], Miroslav Ruda[1], Zdeněk Šustr[1], Simone Campana[3], Elisabetta Molinari[4], David Rebatto[4]

[1] CESNET, z. s. p. o., Prague, Czech Republic
[2] Institute of Physics of the Academy of Sciences, Prague, Czech Republic
[3] CERN, Geneva, Switzerland
[4] INFN, Milano, Italy

E-mail: ljocha@ics.muni.cz

Abstract. Grid middleware stacks, including gLite, matured into the state of being able to process up to millions of jobs per day. Logging and Bookkeeping, the gLite job-tracking service, keeps pace with this rate; however, it is not designed to provide a long-term archive of information on executed jobs.

ATLAS — representative of a large user community — addresses this issue with its own job catalogue (ProdDB). Development of such a customized service, not easily reusable, took considerable effort which is not affordable by smaller communities. On the contrary, Job Provenance (JP), a generic gLite service designed for long-term archiving of information on executed jobs focusing on scalability, extensibility, uniform data view, and configurability, allows more specialized catalogues to be easily built.

We present the first results of an experimental JP deployment for the ATLAS production infrastructure where a JP installation was fed with a part of ATLAS jobs, and also stress tested with real production data.

The main outcome of this work is a demonstration that JP can complement large-scale application-specific job catalogue services, while serving a similar purpose where there are none available.

1. gLite Job Provenance

The need for a grid middleware service that would help users tracking their jobs, store the information for long term, allow adding further annotations, and provide efficient querying capabilities finally, was the primary motivation for developing *Job Provenance* (JP).

Pragmatic implementational requirements on JP emerging from its main purpose are rather contradictory. Information on each job should be sufficiently detailed, while the gathered data should be stored for long time. This implies ever growing requirements on storage space that must be kept reasonable by making job records as compact as possible. The EGEE project aims at 1 million jobs per day; quantitative assessments of implications in JP are given in [1], as well as deployment considerations. At the same time efficient queries are required, which is virtually impossible on huge number of compact records. Finally, JP has to be able to cope with long-term evolution of various data formats consistently.

The overall JP design tries to keep these requirements in a reasonable balance. In this section we provide an overview. Further details can be found in [1, 2].

1.1. Data in JP and Their Organization

The primary data organization in JP is on a per job basis, a concept following the model of Logging and Bookkeeping service (L&B) [1]. Every data item stored in JP is associated to a concrete grid job. The following data are gathered from the grid middleware:

- *Job execution trace*, witnessing the environment of job execution — complete L&B data, that is when and where the job was planned and executed, how many times and for what reasons it was resubmitted etc.
- *Job annotations* — the JP service allows the user to add arbitrary annotations to a job in the form of "name = value" pairs. These annotations can be recorded upon job submission (via JDL[1].), during the job execution or at any time afterward. Besides providing information on the job (for example that it was a production-phase job of a particular experiment) these annotations may carry information on relationships between the job and other entities like external datasets.

In order to overcome the diversity of various data formats as well as their long-term evolution, to provide further extensibility, and to unify the handling of different data, the JP concept distinguishes between the following views on the data:

- *Raw representation* — the physical data received and stored in JP. There are two input and storage modes in JP:
  - Small size *tags*, expressed as "name = value" pairs, enter the system via its primary interface (a web service operation in the current implementation). "Value" is assumed to be a literal, without any structure that JP should be aware of.
  - *Bulk files*, typical example is the complete dump of L&B data, are uploaded via a suitable transfer protocol. Files are supposed to be structured. However, they are stored "as is", and upon upload they are annotated with format identification, including version of the format. To extract required information, specific plugins that understand the file structure and are able to handle particular file formats can be installed in JP.
- *Logical view* is an abstract level used for manipulation of JP data, and it is the preferred way for the most of JP operations (queries are specified in terms of attributes of the logical view). The following list summarizes the basic ideas:
  - All data are expressed by *attributes* at the logical level. A job attribute has a unique name and may have multiple values for a single job. The attribute name must be fully qualified with a namespace (its schema can be specified and enforced) in order to ensure extensibility and uniqueness.
  - Explicitly recorded tags (user annotations) map to attributes in a straightforward way, the name and value of a tag becoming the name and value of an attribute, resp.
  - An uploaded file is usually a source of multiple attributes, which are "digested" from the file based on knowledge of a particular file structure and semantics. For example, L&B dump file provides attributes like job submission and completion time, number of resubmits, computing element where the job ran etc.
  - The "digest" process extracting attributes from raw data files is implemented with *JP plugins*. The task of a plugin is parsing a particular file type and providing calls to retrieve attribute values. JP defines a fixed plugin interface API, supporting thus even future file formats.

[1] Job Description Language, self-contained specification of the job (executable, arguments, inputs, etc.) in the gLite workload management, see [3]

1.2. JP Components

JP provides two classes of services: a permanent *Primary Storage* (JPPS) and possibly volatile and configurable *Index Servers* (JPIS). Primary Storage accepts and stores job data, covering the first set of requirements specified for a Job Provenance — storing a compact job record, allowing the user to add annotations, and providing elementary access to the data. Index Servers provide an optimized querying and data-mining interface for the end-users. Relationship of JPPS and JPIS is a many-to-many — a single JPIS can query multiple JPPS's and vice versa, a single JPPS is ready to feed multiple JPIS's. Typically, the end users query JPIS's only.

The query language is intentionally restricted in order to allow efficient implementation of the query engine. The current format of the query is a list of lists of conditions. A condition is a comparison (less, greater, equal) of an attribute value to a constant. Items of an inner list must refer to the same attribute and they are logically OR-ed. Finally the inner lists are logically AND-ed. According to our experience with the L&B service, this query language is powerful enough to satisfy user needs while simple enough to allow efficient processing.

Index Servers are created, configured, and populated semi-dynamically according to particular user community needs. The configuration is formed by:

- one or more Primary Storages to contact,
- conditions (expressed in terms of JP attributes) on jobs that should be retrieved,
- list of attributes to be retrieved,
- list of attributes to be indexed — a user query must refer to at least one of these for performance reasons.

The set of attributes and the conditions specify the set of data that is retrieved from JPPS, and they reflect the assumed pattern of user queries. The amount of data fed into a single JPIS instance is assumed to be only a fraction of data in JPPS, both regarding the number of jobs, and the number of distinct attributes.

2. ATLAS Job Processing

The ATLAS experiment uses 3 grid flavours (EGEE, OSG and NorduGrid) for event simulation and reconstruction jobs. A custom database, a supervisor, and executors are the main parts of the ATLAS production system (ProdSys). Figure 1 shows a graphical representation of relations between components of the production system. The database (ProdDB) stores definitions of tasks and jobs and their current status. Supervisor is a common interface for all 3 grids for a communication with the ProdDB. Executors — Lexor for EGEE — translate job definitions to a specific format according a grid system and via the supervisor store states of jobs in the ProdDB.

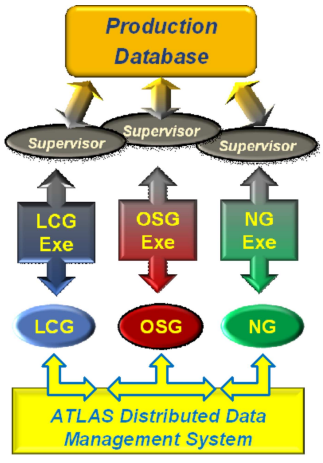

Figure 1. ATLAS production system high-level overview

A *task* is a high level definition of a set of jobs. It defines input and output datasets and a transformation to be used. Jobs are created from a task in the ProdDB and then passed via the supervisor to a pre-defined executor. Jobs are being resubmitted until successful completion or reaching the maximum number of attempts. A schematic overview of main tables in the ProdDB and their dependencies is shown in Figure 2. See [4] for a detailed description of the production system and [5] for recent updates of the EGEE part.

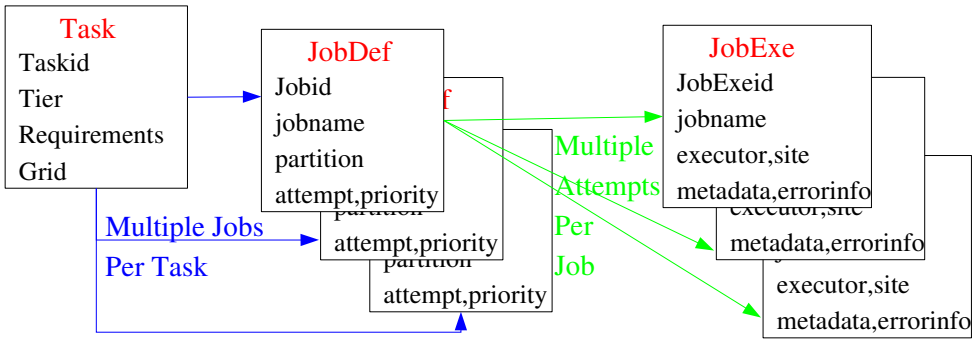

Figure 2. Main entities in ProdDB

3. JP deployment for ATLAS

3.1. Data Model

Following the JP data model, all detailed information on the jobs, including ATLAS-specific data, is expressed in terms of the job attributes (section 1.1). The ATLAS-specific attributes used in our experiments are summarized in Table 1.

Attributes taskId, jobDefId, and JobExeId are unique identifiers from tables in ATLAS ProdDB and they form the relationship of the JP job record with ProdDB. The others carry additional details (this set can be extended arbitrarily).

It can be seen clearly that the data model is flat w. r. t. the original hierarchical one (section 2). Despite of its well-known limitations (namely breaking normal forms of relational data model), the JP flat model is generic, allowing any hierarchy to be mapped into it, without any changes in JP. We can support this view on the example of mappping of ATLAS hierarchy:

- The hierarchy "task — job definition — job execution" is still present at the logical view.
- With appropriate JP plugins, the higher level entities (task and jobdef) can be stored in JP as "virtual jobs", therefore avoiding multiple storage of the task- and jobdef-level attributes with all jobs.

3.2. Service Setup

For the presented experiment we deployed a L&B and JP service chain on the EGEE JRA1 Preview testbed. This is supposed to be a place where developers expose new versions of middleware to be available to users for early evaluation and feedback on new features. The

Table 1. ATLAS specific job attributes. All are from namespace `http://egee.cesnet.cz/en/Schema/JP/Atlas`

Attribute	Meaning
taskId	ATLAS task identifier
taskName	descriptive task name
jobDefId	job definition identifier
jobExeId	job execution identifier
inputDatasetName	name of dataset processed by the job
partNr	processed partition of the dataset

used hardware does not significantly differ from hardware used in the production. Deployed services are:

- *LB server* (`skurut68-1.cesnet.cz`, Intel Xeon 3 GHz, 1 GB RAM) — a dedicated LB server configured to provide all information about jobs to the JP primary storage.
- *JP Primary Storage* (`umbar.ics.muni.cz`, AMD Athlon64 2.4 GHz, 1 GB RAM)
- *JP Index Server* (`skurut68-2.cesnet.cz`, Intel Xeon 3 GHz, 1 GB RAM) — for the purpose of the tests, we deployed just one JPIS customized to answer queries of our experimental GUI (section 3.3.1). In particular, it means that the index server is configured to get all attributes from Table 1 together with additional system attributes sumarized in Table 2. This index server is configured to retreive all available[2] ATLAS jobs (identified by existence of taskid attribute) submitted in August 2007. A real world deployment scenario assumes multiple JPIS's for each JPPS. Each of these index servers is configured to meet a particular user group needs, e. g. to retreive jobs only for a given set of datasets/tasks, time window, or computing elements used.

The only change in the ATLAS production system is that its WMS (`egee-rb-01.mi.infn.it`) is configured to use L&B service dedicated to this experiment. In order to access ATLAS specific information, we instrumented the ATLAS job executor to record new job attributes as described in the previous section. The information known at submission time is recorded as a part of JDL while the information known only at run-time or after job completion is logged as L&B user tags.

Table 2. Additional system attributes. Namespace prefix is omitted.

Attribute	Meaning
user	Job owner
finalStatus	Final job status (Done, Aborted, Cancelled)
LRMSDoneStatus	Final job status as reported by LRMS (computing element)
LRMSStatusReason	Reason of the status

3.3. User Interfaces

We describe the design and implementation of the application used to display jobs' information during the tests. In order to demonstrate the flexibility of the approach we also discuss two other potential views on the data that can be implemented quite easily with appropriate configuration of JPIS and slightly adapted end-user application.

3.3.1. Implemented user interface. Besides allowing users to browse individual job attributes, the client also offers a comprehensive 2-dimensional overview of jobs stored in JP. Figure 3 shows the client displaying a dataset/CE matrix. Each cell of the matrix (left half of the screen) contains icons representing the states of jobs matching the given dataset/CE combination. There may be failed jobs (red), succesfully finished jobs (green), or no jobs at all (empty). Clicking any of the cells opens a list of matching jobs in the top right table. Any of these jobs may be selected to display actual attributes in the bottom right table.

The graphical user interface has been implemented as a native Linux application written in C using GTK. It accesses the given JP Index Server and retrieves information in a fairly simple

[2] It means, due to the overall testbed setup, those submitted via Lexor instance in Milan.

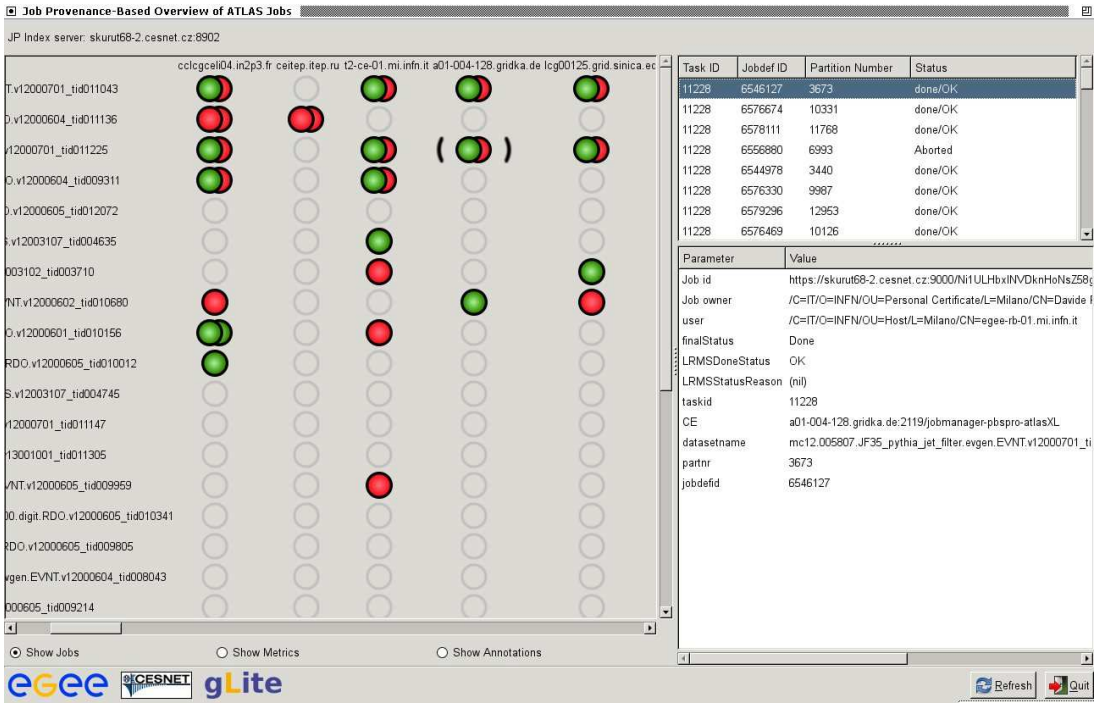

Figure 3. Typical view on the tested user interface

way. Queries get sent to the Index Server through gSOAP interface. The client reads available attributes for all ATLAS jobs at startup, and then decides on the way of representing job status overview in a matrix. It can either display information on the number of finished/failed jobs, or display color-scale representations comparing the values of various numeric attributes. The client supports switching between several views.

Modifying the program to display data in a different manner, to base the overveiw matrix on different attribute values, or making other adjustments is extremely simple.

3.3.2. High Priority Task Processing in Time. Some tasks in ATLAS production are assigned high priority as their completion is critical for various reasons. Jobs of high priority tasks are submitted first by executors, however, it may happen that some of the jobs keep failing, blocking completion of the entire task. Typically, human intervention is required.

The human operator can be alerted by only a brief look at a view similar to Figure 3 with only jobs of high-priority tasks being shown, arranged in rows according to tasks, and in columns according to days or weeks of their execution. In this view, tasks spanning over too long time or exhibiting too many job failures would be easily identified visually.

Implementation of the view requires adding the ATLAS task priority (`priority` column of EJOBDEF ProdDB table) among the attributes stored in JP. As it is known on job submission, it can be passed via JDL. Then, JPIS queries JPPS only for jobs exceeding certain task priority threshold, besides the usual timespan of their execution, and a detailed job record similar to the previous case is retrieved. Depending on actual number of job records, the user front-end queries for all jobs in this JPIS, or restricts the set further.

3.3.3. Executor vs. Computing Element. Some executor instances submit jobs to a certain administrative domain only (T1 cloud), others submit more widely but they should avoid areas dedicated to the specialized ones. An appropriate view arranges jobs in rows according to the

executor instance and columns by computing element. Unlike the previous views, the number of jobs in an array cell is mapped to color saturation of the icon (eventually two icons for successful and failed jobs). Then distribution of jobs to CEs is visible easily, as well as any anomalies, under- or over-loading of CEs by certain executors, etc.

The view requires adding the executor instance identifier (`executor` column of EJOBEXE table), passed via JDL again. JPIS would query for jobs run in certain time interval, and retrieve a tiny job record only (executor id, CE id, and final status).

This usecase becomes more challenging when the number of jobs that are retrieved from JPIS becomes too big (see section 3.4 for quantitative assessment) to be processed by the front-end. We consider extending the JPIS query interface with operations similar to `count()` and `group by` in SQL to cover similar usage.

3.4. Test Results

We have performed two classes of measurements — *on-line*, *stress* and partial *scalability* tests.

The on-line tests were done with the real ATLAS production traffic (all jobs handled by LCG executor in INFN Milano site) during two weeks in August 2007, using the testbed described in section 3.2. Job data flew through the entire chain of the services involved (L&B, JPPS, and JPIS), and we focused on observing the services' behaviour. Neither stability problems, nor any visible congestion were seen during the tests. Table 3 summarizes quantitative results.

Table 3. On-line tests summary

Number of jobs in the test	14,079
Test duration	12 days
Job arrival rate	1,110 jobs/day

On the other hand, the stress tests were run with the data captured in the on-line test, but feeding them into JP services at maximum possible speed. In addition, throughput of individual components was measured too. Table 4 shows the results.

Table 4. Stress tests results (throughput in jobs per day)

export job records from L&B	1,247,000
L&B → JPPS job registration	1,380,000
import job records into JPPS	147,000
JPPS → JPIS feed	1,556,000

An obvious bottleneck of the entire chain was identified in the import of job records into JPPS. This appears due to not reusing an available GSI-ftp connection in the current implementation and it can be removed quite easily.

The overall throughput (147 kjob/day) is higher than the current production throughput (1 kjob/day that we got in partial tests and approx. 20 kjobs/day reported as overall ATLAS production [5]). However, e. g., the target infrastructure throughput of 200 kjobs/day announced by CMS [6] is higher, therefore further performance improvements are necessary in the mid-term.

On the other hand, the end-user observes directly only the last element of the chain — the JPIS query. For the whole set of 14,000 jobs we measured approx. 7 s response time. (This

particular test case is not quite realisitic, we assume only hundreds of jobs being hit by a typical query; the test was overscaled to get measurable timing.) Detailed profiling shows that most of this time is spent in encoding and decoding SOAP XML messages of the communication protocol, therefore it is proportional to the amount of data sent to the client, not to the number of jobs in the JPIS database. Altogether, we consider throughput of this most critical part to be sufficient currently.

Finally, a partial scalability test was done by extracting data on approx. 560,000 former EGEE jobs available in ProdDB, and storing them in JPPS in the form of JP tags. The single job record obtained in this way is not comparable in size to those from the previous full-size tests as the L&B dump is not available anymore. Therefore, the measured throughput (approx. 5× higher) cannot be compared directly. However, the throughput sustained for the duration of the whole test (20 hours). Therefore we can conclude that the whole service is stable even when it is loaded with number of records corresponding to a long term real production deployment.

3.5. Non-gLite jobs

JP instalation described in this paper covers only ATLAS jobs submitted with gLite WMS onto the EGEE grid. However, as we mentioned in previous sections, ATLAS supports submissions not only to EGEE, but also to OSG and NorduGrid. L&B service, the source of events about job state on the grid, was already extended to support non-gLite jobs. In [7] we described L&B extension which supports jobs submitted directly to Condor and PBS. Similar extension of JP strongly reusing the L&B code is in a development phase currently, and it will provide virtually all necessary support for jobs submitted to OSG. Support for NorduGrid is not implemented. Both L&B and JP could be extended to support NorduGrid ARC job-submission system [8] analogically to Condor support. However, the required effort would be higher in this case.

4. Conclusion

Job Provenance service was developed in order to address user requirements for keeping information on grid jobs for long time, a task which was unfeasible to fulfill with the existing on-line job tracking service, Logging and Bookeeping.

The service was deployed experimentally at the EGEE JRA1 Preview Testbed, and it was fed with a part of the ATLAS production traffic (approx. 1,000 jobs/day). The services sustained this load without any observable problems. Consequent stress tests, performed with the gathered real data, show that the expected overall throughput of the available version is 147,000 jobs/day, matching the current requirements of the HEP experiments. On the other hand, the production-scale deployment identified a clear but addressable bottleneck. After fixing it we can expect the overall throughput of 1 million jobs per day.

Using the configurable JP services, a thin client delivering one possible view of executed jobs (both summary and details) was implemented. Two other different views were analyzed and found easily implementable. These end-user applications demonstrate that JP can be used to deliver many custom views according to particular and evolving user needs with high flexibility and little additional development effort. Our other work [9, 10] shows that this concept can be extended towards job submission and stearing, too. Therefore, JP is suitable for building systems similar to ATLAS ProdSys with much less development effort w. r. t. these custom solutions.

In the near future, JP will be provided for general use in gLite middleware release 3.1. We plan to provide extensive support to its deployment, as well as to development of application-specific customizations, gathering feedback for further improvements and extensions of the JP service.

Acknowledgement

This work was supported by the European Union within the EGEE-II project, INFSO-RI-031688.

References

[1] Matyska L *et al.* 2007 Job tracking on a grid—the Logging and Bookkeeping and Job Provenance services Tech. rep. CESNET `http://www.cesnet.cz/doc/techzpravy`
[2] Dvořák F *et al.* 2006 *Proc. IPAW'06, LNCS 4145* pp 246–253
[3] Pacini F *et al.* 2006 Job description language attributes specification `https://edms.cern.ch/file/590869/1/EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8.pdf`
[4] ATLAS Collaboration 2005 ATLAS computing technical design report Tech. Rep. ATLAS TDR–017, CERN_LHCC-2005-022 CERN
[5] Espinal X *et al.* 2007 *Proc. CHEP'07*
[6] Gutsche O and Hajdu C 2007 *Proc. CHEP'07*
[7] Ruda M *et al.* 2007 *HPDC 2007 Workshop on Grid Monitoring*
[8] Ellert M *et al.* 2007 *Future Generation Computer Systems* **23** 219–240
[9] Křenek A *et al.* 2007 Multiple ligand trajectory docking — a case study of complex grid job management, demo and poster at EGEE User Forum, Manchester, UK
[10] Křenek A *et al.* Multiple ligand trajectory docking study — semiautomatic analysis of molecular dynamics simulations using EGEE gLite services, submitted for PDP'08