gLite Job Provenance*

František Dvořák, Daniel Kouřil, Aleš Křenek, Luděk Matyska, Miloš Mulač, Jan Pospíšil, Miroslav Ruda, Zdeněk Salvet, Jiří Sitera, and Michal Voců

CESNET z.s.p.o., Zikova 4, 160 00 Praha 6, Czech Republic
First.Last@cesnet.cz

Abstract. The Job Provenance (JP) service is designed to automate keeping track of computations on large scale Grids, giving thus users a tool to correctly archive information about their jobs and to re-submit any job in a reconstructed environment. JP provides a permanent minimal record of job (and its environment) related information, to which free-form user annotations can be added. JP also offers the capability of configuring any number of indexed logical views on the large collections of raw data, allowing efficient processing of even complex user queries selecting on both system data and the annotations. The scalable architecture, capable to handle millions of jobs in a single JP installation, and integrated into the EGEE gLite middleware environment is presented.

1 Job Provenance

New methods, instruments, and sensors are producing extreme amount of raw experimental data. The data must be further processed to provide a novel scientific insight and new knowledge. This leads to increased importance of processed (computed) scientific data whose amount growth at least exponentially. In this context the computation gets into the position of a traditional scientific experiment, including the principle that any results, should they be accepted by the community, *must be verifiable by re-doing the experiment*, i. e. re-running the computation. Consequently, an exact description of the computation — the *job* that produced a piece of data — contributes to the data provenance.

Unfortunately, many users, despite being scientists who are used to keep thorough track of their "real" experiments, tend to underestimate the importance of their computational experiments rather frequently. A common practice is running a job, analyzing and keeping the output, modifying some input parameters to run another iteration, but not archiving the original parameters. Then, after the user forgets the original parameters, the results of the first job get orphaned in the provenance sense, despite being archived otherwise.

In addition, the environment (e. g. versions of used software or internal parameters) of the job may affect the result of the computation, hence contributing to its provenance too.

We propose the Job Provenance service to automate the important but tedious task of keeping track of computations in the case of Grid jobs, as well as to allow effective access to the gathered data.

* This work has been supported by the EU EGEE project INFSO-RI-508833.

L. Moreau and I. Foster (Eds.): IPAW 2006, LNCS 4145, pp. 246–253, 2006.

1.1 Summarized Requirements

The designed service must keep a permanent (for several years) record of each registered job. This record contains information that is necessary to re-run the job, achieving the same results. The length of each record must be as minimal as possible, data which are stored elsewhere should not be included. The service should scale over the extent and throughput of current Grid middleware (e. g. EGEE reports [11] 20k jobs per day, i. e. 7.5M per year). The user should be allowed to add free-form annotations to each job record. A querying interface must be provided, allowing to inspect and retrieve records according to criteria specified on either the job data or the user annotations.

We have also more practically oriented goals, related to the fact that the service will be part of the gLite Grid middleware (Sect. 2.1). The first release should be deployed in at least moderate scale in 2006, so that feedback and eventual design revision may happen within the EGEE II project, i. e. by the end of 2007.

1.2 Related Work

A primary work on provenance is part of the EU Project *Enabling and Supporting Provenance in Grids for Complex Problems* [2]. They proposed a broad definition of provenance — *the provenance of a piece of data is the process that led to the data* — and presented proof of concept work on provenance in a Service Oriented Architecture [8]. Most current provenance architectures are concerned with incorporation of provenance capabilities into a Web Service-based Grid environments [4,5,10,12,13]. The usually considered scenario of execution of a composite service by trusted workflow engine provides the ability to collect and archive the provenance about the transformation of data during invocation of web services. This architecture is not practically suitable for our work as we need to support considerable number of "legacy", non-WS-based services.

In EGEE related projects, number of HEP experiments have their own workflow systems that also provide some provenance and annotation (metadata) capabilities. The ATLAS experiment uses AMI database application framework [7] for production physics bookkeeping. AMI also manages "tasks" — the transformations that can be applied to datasets and their configuration parameters — and stores links between tasks and datasets. AliEn (ALICE Environment) [1] is a Grid framework that takes care of job splitting and execution, manages datasets, and keeps track of the basic provenance of each executed job or file transfer. AliEn File Catalog also provides interface for attaching new annotation (metadata) database tables to its standard directory tables. SAM [6] used in the DZero Experiment is a data handling system designed to store and retrieve files and associated metadata, including a complete record of the processing which has used the files. The major drawback of those systems is that they do not typically record activities of other middleware services and do not provide full provenance data due to their position at the top of the job submission (service invocation) chain. On the contrary, the Job Provenance is designed as an essential Grid service, defining a unified but still flexible framework for keeping records of jobs.

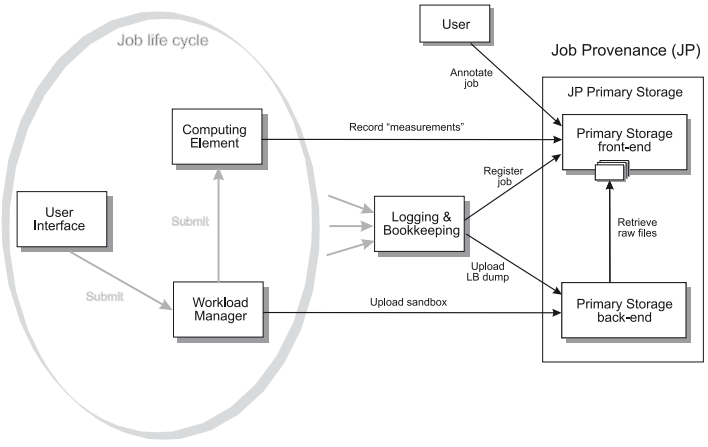

Fig. 1. Data flow into gLite Job Provenance

2 General Design

2.1 The Place of JP in gLite

The gLite Workload Management System (WMS) [3] consists of a set of Grid middleware services that facilitate convenient and efficient distribution and management of tasks across Grid resources. It accepts and dispatches users' requests for job management (mainly submission and cancellation), the decision which resource should be used is the result of a matchmaking process between requests and available resources. The gLite job properties and requirements are described in the job description language (JDL). The submission request can contain auxiliary input files (input sandbox) copied to an execution machine by WMS.

Currently, the information about jobs submitted to gLite Workload Management System is collected by the Logging and Bookkeeping (L&B) service. L&B tracks jobs in terms of events received from WMS components and CEs that are instrumented with the L&B calls. The events are collected by the bookkeeping server that processes them in a real time to give overall view on the actual job state. The user may query the bookkeeping server to obtain either the raw events or the computed job state, she may also register for receiving notifications on particular job state changes.

The L&B takes care of job information only during the job lifetime, purging the database when the job leaves the Grid. Also, L&B does not keep the input and output sandboxes, they are handled separately by WMS. The purpose of the gLite Job Provenance is to provide the permanent storage of the job related information as stored within the L&B, to couple it with the input sandboxes and other system oriented information necessary to reproduce the environment where a particular job run. Fig. 1 depicts basic gLite middleware components and their interaction with the Job Provenance.

2.2 Data Gathered from Middleware and User Annotations

The storage capacity requirement for a service to keep permanent track of a very large number of Grid jobs may become enormous. Consequently the data recorded for each job must be strictly limited. As a rule of thumb we store only volatile data which are neither stored reliably elsewhere nor are reproducible by the job. The data gathered from the gLite middleware fall into the following categories:

- job inputs, directly required for job re-running
  - complete job description (JDL) as submitted to WMS
  - miscellaneous input files (gLite WMS input sandbox) provided by the user (but job input files from remote storage *are not* copied to JP)
- job execution track, witnessing the environment of job execution
  - complete L&B data, i. e. when and where the job was planned and executed, how many times and for what reasons it was resubmitted etc.
  - "measurements" on computing elements, e. g. versions of installed software, environment settings etc.

In addition, the service allows the user to add arbitrary annotations to a job in the form of "name = value" pairs. Annotations can be recorded either during the job execution or at any time afterward. Besides providing information on the job (e. g. it was a production-phase job of particular experiment) these annotations may carry information on relationships between the job and other entities like external datasets, forming the desired data provenance record.

2.3 Raw and Logical Data Representation

At the *raw level*, data enter JP and are stored in two ways: (i) small size *tags*, i. e. "name = value" pairs, and (ii) uploaded bulk files. At the *logical level*, any piece of information stored in JP is represented as a value of particular named *attribute*. In this representation both the user annotations and the "system" middleware data are unified in a single view.

Data stored as tags map to attributes in a straightforward way, name and value of the tag becoming name and value of an attribute. An uploaded file (L&B log, job sandbox, ...) is usually a source of multiple attributes. JP defines a *file-type plugin interface API*; the task of the plugin is parsing a particular file type and providing calls to retrieve attribute values.

For the purpose of extensibility an attribute name always falls into a namespace. Currently we declare namespaces for JP system attributes (e. g. job owner or registration time), attributes inherited from L&B, and unqualified user tags.

2.4 Typical Usage

Propagation of data from other middleware components to JP is done transparently. The user may specify both the destination JP and which data are gathered via special parameters in the job description, however, these settings

may be overridden by WMS or CE policy. The user also interacts with JP directly when recording annotations. Retrieval of information on *a concrete job* is fairly straightforward — the job is the primary entity in JP and all data are organized on a per-job basis, hence easily available.

But the principal purpose of JP is *searching for jobs* according to some criteria, notably jobs that either produced or used a given piece of data, freeing the user of the burden to keep complete records on her jobs. Such a search would result in scanning through all data stored in JP which are expected to be huge, being unacceptable for frequent user queries. Instead we define an architecture that allows batch pre-processing of configurable queries. The result of such query, a superset of certain user query type, is further indexed in order to provide fast response to concrete user queries.

3 Architecture

JP is formed of two classes of services: permanent *Primary Storage* accepts and stores job data while possibly volatile and configurable *Index Servers* provide an optimized querying and data-mining interface to the end-users.

3.1 Primary Storage

The JP *Primary Storage* (JPPS) is a permanent service responsible for gathering the job data and their long-term archival. The primary data are kept in as compact form as possible, and only minimal metadata (job ID and owner, registration time) are maintained and indexed.

A single instance of JPPS is formed by a front-end, exposing its operations via a web-service interface[1], and a back-end, responsible for actual data storage and providing the bulk file transfer interface. In the current implementation metadata are stored in a relational database. The back-end uses Globus grid-ftp server enriched with authorization callbacks accessing the same database to check whether a user is allowed to upload or retrieve the given file. Both the front- and back-ends share a filesystem so that the file-type plugins linked into the front-end access their files via POSIX I/O.

Job registration. Each job has to be explicitly registered with JP. The registration is done transparently, the L&B server calls JPPS front-end `RegisterJob` operation upon job submission (in parallel with the job registration in L&B).

Data upload. The `RecordTag` operation records the "name = value" tags. Uploading a bulk file is a more complex, three-stage sequence:

- Call `StartUpload` front-end operation. If authorization check succeeds, the service responds with upload URL with a limited time span.
- Upload the file to the specified URL. The authorization callback of the grid-ftp server checks whether this particular URL was "opened" in the previous step for the concrete user and is still valid.

[1] Described in detail in [9], documented web service definitions can be found at `http://egee.cesnet.cz/en/WSDL/`

- Confirm the finished upload with `CommitUpload` front-end operation. This "closes" the URL for upload, makes the file available to the front-end, and opens the URL for eventual download.

Index Server feed is the data-mining interface called by JP Index Servers described in Sect. 3.2.

Data retrieval. The only direct data retrieval supported by JPPS is keyed by ID of jobs. There are two operations, both taking job ID as their argument:

- `GetJobAttributes` retrieves attribute values, either stored as user tags or extracted from uploaded files via the file-type specific plugins.
- `GetJobFiles` returns URL's pointing to the job files stored at the Primary Storage back-end. The user may retrieve the raw files via the back-end interface, and parse them on her own.

Only limited number of JPPS installations must be deployed even on a large Grid to concentrate the provenance data. At most one JPPS per a virtual organization is envisaged for the EGEE environment. This mean each JPPS must be able to deal with data on millions of jobs. The typical size of an L&B dump is around 10 kB per compressed record, and gLite users are encouraged not to use large job sandboxes, too. Consequently, the back-end storage requirements are at the order of 10-100 GB. JPPS metadata are formed by a single tuple for each job and for each file, with unique indices on job ID and file name. The used MySQL database engine is capable to handle millions of such records.

Primary Storage covers the first set of requirements specified in Sect. 1.1 — storing a compact job record, allowing the user to add annotations, and providing elementary access to the data.

3.2 Index Server

The role of *Index Servers* (JPIS) is processing and re-arranging the data from Primary Storage(s) into a form suitable for frequent and complex user queries. A typical interaction is shown in Fig. 2.

1. The user queries one or more JPIS, receiving a list of ID's of matching jobs.
2. JPPS is directly queried for additional job attributes or URL's of stored files.
3. The required files are retrieved.

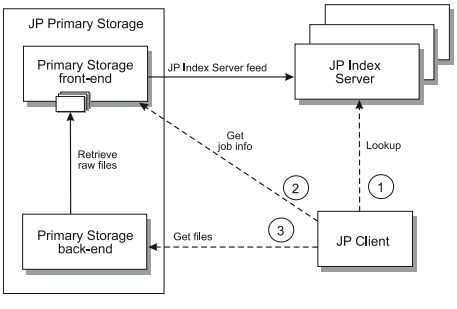

Fig. 2. Index Server interactions

The current format of the user query is a list of lists of conditions. A condition is comparison (less, greater, equal) of an attribute w. r. t. 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 implementation.

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 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 it reflects 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.

Communication between JPIS and JPPS involves two complementary web-service operations: JPIS calls the `FeedIndex` operation of JPPS, specifying the list of attributes and conditions. Unlike the user queries, the query on JPPS is a single and-ed list, allowing less complex processing on JPPS where significantly larger data set are involved. JPPS responds by calling the `UpdateJobs` operation of JPIS (repeatedly to split up large dataset).

The following flags in the `FeedIndex` call specify the query mode:

- *history* — JPPS should process all its stored data, giving the user the guaranty that if her query is a logical restriction of the JPIS configuration, it returns a complete result. This type of query is usually necessary to populate JPIS but it imposes rather high load on JPPS.
- *continuous* — JPIS registers with JPPS for receiving *future updates* when data matching the query arrive. This type of query allows JPIS to be kept up to date while imposing minimal load on JPPS.

The current JPIS implementation keeps the data also in a MySQL database. Its schema is flexible, reflecting the Server configuration (columns are created to hold particular attribute value, as well as indices). There is no prescribed relationship between Primary Storage and Index Server installations. An Index Server may retrieve data from multiple Primary Storages and vice versa.

4 Conclusion

The gLite Job Provenance service is a specific provenance that helps to keep track of millions of jobs, their environments and inputs in large Grids. Data collected directly by the Grid middleware could be arbitrarily annotated at any time. While designed as an independent service, the JP implementation is integrated into the EGEE gLite middleware.

End users can query JP to obtain data about a specific job. In addition, user communities are supposed to install JP Index Servers to process continuously the JP data (even collecting data from several JP Primary Storages) and to

provide logical views optimized for specific users' queries. In this way the users obtain more complex information and knowledge from the JP data. Independent Index Servers can be deployed in order to provide different logical views, giving thus fast access to even the largest collections of job related data.

The gLite Job Provenance service is currently being deployed on the EGEE Grid. A set of Index Servers is pre-configured to provide most common logical views on the data. The large scale deployment will test the architecture and its scalability and it will also provide a necessary feedback for further extensions: the plug-ins, specific IS configurations, expressing power of the annotations, specific tools for manipulation with the primary data (e.g. automatic and parametrized job re-submission), etc. Also, the deployment will be used to test and extend the current simple authorization model used in the first JP implementation.

References

1. AliEn (ALIce ENvironment). `http://aliceinfo.cern.ch/AliEn`.
2. EU FP6 Programme Enabling and Supporting Provenance in Grids for Complex Problems. `http://twiki.gridprovenance.org/bin/view/Provenance/ProjectInformation`.
3. gLite—Ligthweight Middleware for Grid Computing. `http://glite.web.cern.ch/glite/default.asp`.
4. Mygrid Provenance Outline. `http://phoebus.cs.man.ac.uk/twiki/bin/view/Mygrid/ProvenanceOutline`.
5. PASOA: Provenance Aware Service Oriented Architecture. `http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/AboutPasoa`.
6. SAM (Sequential data Access via Meta-data). `http://d0db.fnal.gov/sam/`.
7. The Atlas Metadata Interface. `https://atlastagcollector.in2p3.fr:8443/AMI/`.
8. Liming Chen, Victor Tan, Fenglian Xu, Alexis Biller, Paul Groth, Simon Miles, John Ibbotson, Michael Luck, and Luc Moreau. A proof of concept: Provenance in a Service Oriented Architecture. In *Proceedings of the fourth UK e-Science All Hands Meeting, Nottingham, UK*, 2005.
9. EGEE JRA1. EGEE Middleware Design—Release 1. `https://edms.cern.ch/document/487871/`.
10. Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS04)*, 2004.
11. EGEE JRA2. Job Metrics. `http://egee-jra2.web.cern.ch/EGEE-JRA2/QoS/JobsMetrics/JobMetrics.htm`.
12. Shrija Rajbhandari and David W. Walker. Support for Provenance in a Service-based Computing Grid. In *Proceedings of the third UK e-Science All Hands Meeting, Nottingham, UK*, 2004.
13. Paul Townend, Paul Groth, and Jie Xu. A provenance-aware weighted fault tolerance scheme for service-based applications. In *Proceedings of the* 8^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, 2005.