

gLite Job Provenance – a job-centric view

Aleš Křenek, Jiří Sitera, Luděk Matyska, František Dvořák, Miloš Mulač, Miroslav Ruda, Zdeněk Salvet

1 Abstract

Job Provenance (JP) is a Grid service that keeps long-term trace on completed computations for further reference. It is a job-centric service, keeping records about job life cycle, its environment, inputs/outputs, user parameters etc. The data collected from the Grid middleware where the job has run can be complemented with user annotations that add a personalized view.

JP is a part of the gLite Grid middleware developed within the EU EGEE project by CESNET based development group. This technical report summarizes participation of CESNET team in the first provenance challenge, event organized by data provenance community.

During this challenge we explored the relation between a specific job-centric Grid oriented provenance and a more general data provenance approach. We show how JP can store data about complex workflows and how these data can be used to answer user queries.

As the implementation of the challenge workflow is realized in a real production level Grid system (gLite based EGEE Grid) it also provides an insight how the workflow tasks can be implemented and run on a Grid.

Keywords: GRID, Job Provenance, Data Provenance, Job State Tracking

2 Motivations

A general requirement in the environment of traditional experimental science states that *any result must be verifiable by redoing the experiment*. Nowadays, many “experiments” are carried in a virtual environment as computations. However, the same principle applies – a result of a computation is questionable and generally not accepted by the community if it is not verifiable by an independent computation.

On the other hand, results of a computation critically depend both on its exact specification (input data, parameters etc.) and environment where the computation is run (for example versions of used software). Therefore, keeping

a long-term accurate trace of computations is critical for the scientific value of the computed results.

In the environment of a computational Grid the requirement is even more critical because the user does not have a complete control over the resources used for her computation (this is an intrinsic property of the Grid).

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).

A related service, *Logging and Bookkeeping* (L&B) [Kou04] was already available from the EU DataGrid project. It is designed for tracking jobs during their active life time, and for providing users with the information on the job state. Due to the optimization for this purpose L&B is not suitable for long-term storage, old data are purged from L&B regularly in order to keep the active data size reasonable for performance purposes. However, users start complaining that the job data they are used to query from L&B suddenly “disappear”. Moreover, not all data required for job re-execution are present in L&B, namely job inputs. Nonetheless, experience with L&B was a good starting point for the JP design.

At the time of writing this manuscript, JP is deployed experimentally on a part of the EGEE¹ infrastructure, and it is in a candidate state for integration into the gLite 3.1 middleware release. It is expected to be used by individual end-users as a long-term extension of L&B, user communities for collaboration, as well as various Grid monitoring and statistics gathering tools. Some expected usage patterns are described in [JPUG06] and an application-oriented use case of JP is being prepared in parallel with this manuscript (see [Pet07]). However, as the goal of JP is to encompass the whole EGEE-like production Grid infrastructure, there is not yet sufficient experience that would lead to clear understanding of usage patterns. This is reflected in the design, allowing high flexibility in configuration as well as further extensions.

We participated in the First Provenance Challenge for two main reasons. First, comparison with other systems was tempting and likely to identify weak points in both JP design and implementation, as well as to bring new ideas for further development. The other reason was testing the usability of JP for a task oriented more on data than process (or job) provenance, that is quite different from the original intention.

¹<http://www.eu-egee.org>

3 Job Provenance design

Pragmatic implementational requirements on JP emerging from its main purpose are rather contradictory. Information on each job should be sufficiently detailed in order to allow job re-execution, 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 [Mat07], 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 a brief overview. Details can be found in [Dvo06].

3.1 gLite job processing in a nutshell

Despite its design being general, capable of handling virtually any Grid jobs, Job Provenance was developed as a part of the gLite middleware [Lau04], and gLite-based Grid was used to run the challenge workflows as well as the queries. Therefore we describe gLite job processing briefly here. Further details can be found in [Ave03].

The only way the user can access computational resources in gLite middleware [Lau04] is through a *job*. Despite not completely restricted to, gLite is designed to support large number of traditional batch, non-interactive jobs.

Upon creation the job is assigned a unique immutable *job identifier* (jobid). The jobid is used to refer to the job all the time during the job active life and afterwards.

The user describes the job (executable, parameters, input files etc.) using the *Job Description Language (JDL)* [Pac06], using the extensible *Classified Advertisement (ClassAd)* [RLS98] syntax. The description may grow fairly complex, including requirements on the execution environment, proximity of input and output storage etc.

Processing of the job can be summarized as follows (denoting gLite components in italics):

- the job is submitted via a *User Interface* (either command line or graphical)
- *Workload Manager (WM)* queues the job and starts finding a suitable *Computing Element (CE)* to execute it

- the job is passed to the chosen CE and runs there
- after completion, the user can retrieve the job output
- all the time, the job is tracked by *Logging and Bookkeeping* (L&B) service, which provides the user with the view on the job state and further details of job processing
- after the user retrieves the job output, the middleware data (namely the job trace in L&B) on the job are passed to JP and purged in their original locations
- annotations can be added to the job during its active life time via L&B (even from inside of the running application) or any time afterwards via JP.

Besides simple jobs gLite supports also complex ones, job workflows in the form of *Directed Acyclic Graphs* (DAG). A DAG is completely described, using a nested JDL syntax, as a set of its nodes (simple jobs) and execution dependencies among them. DAG processing is implemented by interfacing the WM planning machinery with the Condor DAGMan².

3.2 Data in JP and their organization

The primary data organization in JP is on a per job basis, a concept following the L&B model. Every data item stored in JP is associated to a concrete Grid job. The following data are gathered from the Grid middleware:

- job inputs, directly required for job re-running: complete job description (the JDL record) as submitted to the WM system (WMS) and miscellaneous input files (gLite WMS input sandbox) provided by the user (however, job input files from reliable remote storage are not copied to JP; this would not be feasible in a large scale)
- 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., and “measurements” on computing elements, for example versions of installed software, environment settings etc.
- JP user tags – the JP service allows the user to add arbitrary annotations to a job in a form of “name = value” pairs. These annotations can be recorded either during the job execution or at any time afterward. Besides providing

²<http://www.cs.wisc.edu/condor/dagman/>

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, forming the desired data provenance record (Section 5.2).

Figure 1 shows the dataflow channels from Grid middleware components (gLite) into JP.

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 or the job input sandbox, 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. JP allows installing plugins that handle particular file formats (see below), understanding the file structure and extracting required information.
- *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, name and value of a tag becoming name and value of an attribute.

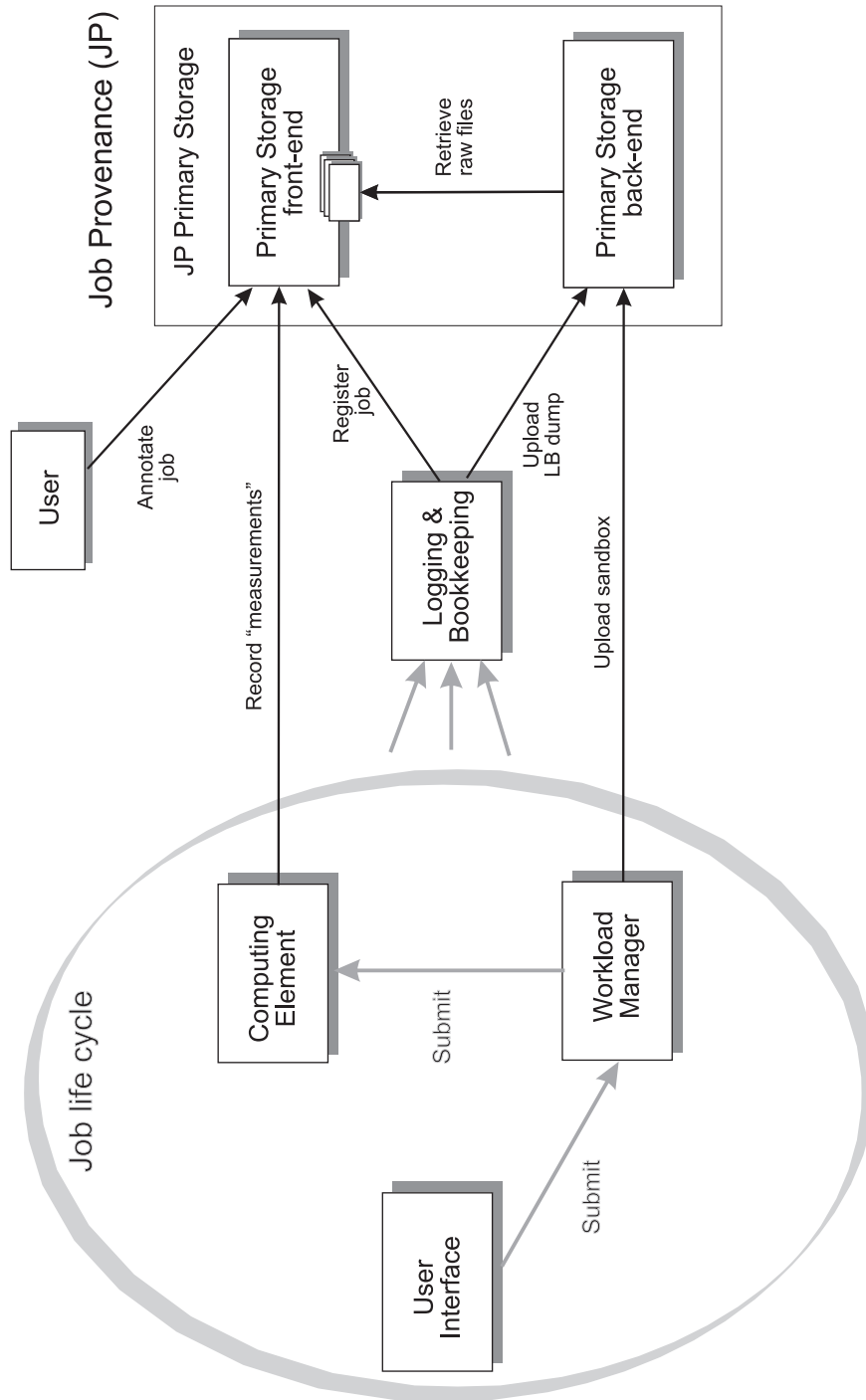


Figure 1: Data flow into gLite Job Provenance

- 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, CE 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.

3.3 JP components

JP provides two classes of services: a permanent *Primary Storage* (JPPS) accepts and stores job data, while possibly volatile and configurable *Index Servers* (JPIS) provide an optimized querying and data-mining interface for the end-users (see Figure 2). 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.

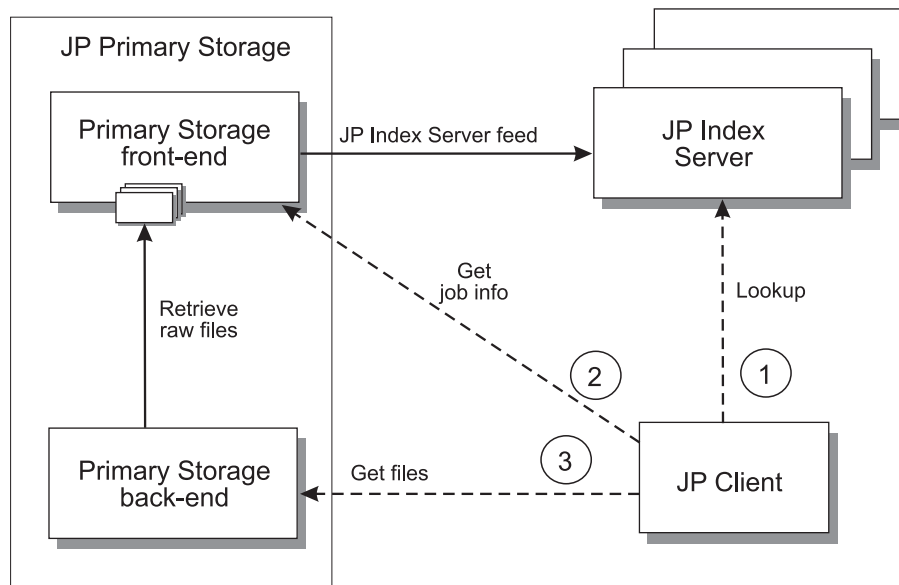


Figure 2: Job Provenance components

3.3.1 Primary Storage

A single instance of JPPS, shown in Figure 2, is formed by a front-end, exposing its operations via a web-service interface([EGEE04]), and a back-end, responsible for actual data storage and providing the bulk file transfer interface using

arbitrary file-oriented transfer protocol(s). 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.

JPPS operations fall into the following categories:

- *Job registration.* Each job has to be explicitly registered with JP. Currently the registration is done transparently by the L&B server upon job submission (in parallel with the job registration in L&B, though not blocking the job submission).
- *Tag recording.* Add user tags (annotations) in the “name = value” form to JP job records.
- *Bulk file upload.* File properties (type, optional name etc.) are specified via the front-end interface, the upload itself goes directly to the back-end (using `gridftp` transfer).
- *Index Server feed* allows JPIS to ask for batch feed as well as register for incremental updates.
- *Data retrieval.* The only direct data retrieval supported by JPPS is keyed by the jobid. Both individual attributes and the whole files can be retrieved.

Primary Storage covers 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.

The current implementation uses MySQL relational database to store basic job metadata and Globus `gridftp` server as the back-end.

3.3.2 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 Figure 2, consisting of following steps:

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

A 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.

For example, query all jobs named “align” executed with parameter “-m 12” and ran on Monday or Tuesday:

```
(program="align") AND (param="-m 12") AND (day="Monday" OR day="Tuesday")
```

JPIS provides the following operations:

- *User queries* – it is the primary interface for end users as described above. A query specifies a set of attributes to be retrieved and conditions determining the set of matching jobs.
- *JPIS-JPPS communication* – implements the data flow from JPPS to JPIS. Each JPIS can subscribe to JPPS according to its configuration to receive data matching the configuration or just ask for data matching the configured criteria.
- *Administrative* – calls to change JPIS configuration without interfering with its normal operation.

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.

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). Currently all attributes are handled as strings, however, we are considering type-extension mechanisms that would allow processing complex attribute types, as well as adding further operators besides simple comparisons.

4 The challenge

In this section we describe in detail our approach to the First Provenance Challenge. Details on the challenge, specification of the challenge workflow as well as the prescribed queries can be found at Provenance Challenge Wiki³.

4.1 Workflow representation

We implement the challenge workflow as a gLite DAG job (Section 3.1). The structure of the DAG follows the specified workflow exactly, using the following mapping:

- *Procedures* become nodes of the DAG, which means they are turned into normal gLite jobs during the DAG processing, and they are executed on the Grid computing resources. Besides down- and uploading the data files (see below) each such job involves running the appropriate AIR, FSL, or ImageMagic utility.
- *Dependencies* among procedures are reflected in dependencies of the DAG. Therefore for example all four `align_warp` invocations can run in parallel but `softmean` must be preceded by successful completion of all four `reslice` instances.
- *Data items*, both input and output, are external files. A unified shared filesystem cannot be assumed on the Grid computing resources, therefore each job is responsible for downloading all its inputs and uploading all its outputs.

Appendix A shows a fragment of the workflow JDL.

For the challenge we put the files on a dedicated GridFTP server and access them (both down- and upload) with the `gsiftp://` protocol (solving also access control – a running gLite job possesses delegated user credentials). Consequently, the data items are identified with their full URL in our implementation.

³<http://twiki.ipaw.info/bin/view/Challenge>

In real world the user might want to use the gLite data services [Lau04] and identify files with GUID s or logical file names. For our experimental runs this would make the implementation even more complicated while additional features are not important from the challenge point of view (replica management).

4.2 Provenance trace

When the execution of a workflow is finished, JP can collect traces of the workflow life from various Grid subsystems (Section ??). Currently only L&B is instrumented to provide the trace, however, the encompassed data are completely sufficient for the challenge. Therefore the *raw representation* of the provenance trace is formed by the L&B data uploaded into JP.

However, as discussed in Section 3.2, the strength of JP is at the *logical level*, mapping the gathered data into JP attributes. At this level the provenance trace is formed by attributes of the following classes:

1. JP system ones (namespace and schema <http://egee.cesnet.cz/en/Schema/JP/System>):
 - `jobid`: identifier of the job
 - `owner`: identity of the job submitter
 - `regtime`: when the job was registered with the middleware
2. digested from L&B trace, conforming to schema <http://egee.cesnet.cz/en/Schema/LB/JobRecord>
3. digested from JDL, describing workflow structure (namespace and schema <http://egee.cesnet.cz/en/Schema/JP/Workflow>):
 - `ancestor`: jobid(s) of immediately preceding job(s) in the workflow
 - `successor`: jobid(s) of immediately following job(s) in the workflow
4. unqualified user tags, logged via L&B (see Section 3.1); they are reported in the namespace <http://egee.cesnet.cz/en/WSDL/jp-lbtag>

All the attributes may occur multiple times, for example as `softmean` must have been preceded by 4 `reslice` s in the challenge workflow, there are 4 occurrences of the `ancestor` attribute of the `softmean` nodes.

For the specific purpose of implementing the challenge workflow and answering the challenge queries we use L&B tags to store additional information about the workflow nodes. JP turns these values into attributes of the 4th kind on the list

Attribute name	Meaning
IPAW_OUTPUT	names of files generated by this node
IPAW_INPUT	names of input files for this node
IPAW_STAGE	name (number) of workflow stage of this node
IPAW_PROGRAM	name of process this node represent
IPAW_PARAM	specific parameters of this node processing
IPAW_HEADER	named anatomy header (currently used for global maximum only)

Table 1: Specific challenge L&B tags attached to each DAG node

above. Table 1 summarizes their meaning, which is self-explanatory with the only eventual exception of “stage” – we consider it to be a logical portion of the workflow, which may or may not match the number of preceding nodes in the workflow execution.

4.3 Characteristics of the system and provenance representation

In this section we comment specifically on the provenance systems classification criteria defined in the context of the first provenance challenge.

C1.1 Execution Environment.

JP was developed as a part of gLite middleware and currently it supports gLite jobs, including their support for simple workflows (based on Condor DAGMan). However, due to the clean distinction between physical data and logical attributes, JP is ready to handle different environments as long as similar data on workflow execution are available.

C1.2 Execution Environment (for the challenge).

The execution was done on production EGEE infrastructure (VOCE⁴ virtual organization), using prototype instances of L&B and JP services.

C1.3 Representation Technology.

In the permanent storage, files from the execution environment are kept in their native form (plugins are used to parse them), including L&B data, annotations are kept separately, also in files. In the volatile index servers (optimized for queries), data are stored in RDBMS. On the logical level, data are represented as `namespace:name=value` with arbitrarily complex structure of `value`.

C1.4 Query Language.

⁴<http://egee.cesnet.cz/en/voce/>

SQL-like but strongly restricted – one logical table, simplified structure of WHERE clause.

C1.5 Research Emphasis.

Main focus is the balance of *storing* huge amounts of data and still efficient *querying*. Some attention is paid to *recording* jobs trace, *execution* is out of JP scope itself.

C1.6 Challenge Implementation.

Full workflow, including the image processing programs, was executed.

C2.1 Includes Workflow Representation.

The challenge implementation includes an explicit representation (in gLite JDL) and uses it to follow the workflow structure. However, this is not strictly necessary, the structure can be more or less unambiguously deduced from data dependencies.

C2.2 Data Derivation vs. Causal Flow of Events.

Event flow.

C2.3 Annotations.

User annotations are considered to be in the scope of provenance and are fully supported in JP.

C2.4 Time.

Timestamps are included with virtually all data in JP. However, precise timestamping, synchronized clocks etc. are not required for capturing correct provenance records.

C2.5 Naming.

JP requires to identify jobs uniquely. Data items are not primary entities in JP, therefore their naming is not required by the JP core. In the challenge we identify files (data) with their URL; unique naming of files is required to support data-related queries.

C2.6 Tracked data, Granularity.

Arbitrary granularity can be handled by JP in general, it depends on what gets stored there. In the challenge, file-level granularity was used.

C2.7 Abstraction mechanisms.

JP uses a simple namespace: `name=value` logical-level representation as a unifying and extensible mechanism of view on any data. However, `value` in this model can be of arbitrary complex XML type.

4.4 Provenance queries

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Y	Y	Y	Y	Y	Y	Y	Y	N TBI

Table 2: Provenance queries matrix row for JP

In this section we describe details of our implementation of the challenge queries. Table 2 shows JP row of queries matrix – which queries were answered successfully during the challenge (TBI means “To Be Implemented”, query in scope, but not yet implemented; the queries Q1 to Q9 are described in this section). Basic building blocks of the implementation are queries to JP components (details in Section 3):

- *JP Primary Storage*: given a jobid, return specific attributes of the job.
- *JP Index Server*: find jobs matching criteria expressed with attributes, return specified attributes.

For JPPS query the only requirement is a known jobid, any attributes can be retrieved. On the contrary, both the criteria and the list of required attributes in JPIS query must match the concrete server configuration. For the purpose of the challenge we set up a JPIS with specific configuration summarized in Table 3. However, setting up such specialized instance is not a “demo only” approach. JP architecture assumes exactly this treatment of JPIS s – they are volatile, set up and populated with data according to particular user needs.

The queries to JP components are glued together in a client program. Processing of the query results still requires non-trivial logic (such as graph search) but, on the other hand, it is always done on a tiny amount of data only, dozens of records typically.

At the time of the challenge there was no sophisticated JP client available. Therefore we implemented the queries in an ad-hoc manner, with Perl scripts wrapping basic CLI clients of both JP services. In this section we use pseudocode in order to describe the functionality, the implementation⁵ is available from the authors.

⁵<http://glite.cvs.cern.ch:8180/cgi-bin/glite.cgi/org.glite.jp.index/examples/pch06/>

Attribute name	Indexed	Meaning
jps:jobId	Y	job identification
jps:owner	Y	job owner, as specified with RegisterJob JPPS operation
jps:regtime	N	job registration time
jpw:ancestor	Y	ancestor(s) in the job workflow
jpw:successor	Y	successor(s) in the job workflow
lb:CE	N	where the job is being processed
lb:parent	N	jobid of the workflow
lb:host	N	worker node where the job is executed
lbt:IPAW_PROGRAM	Y	
lbt:IPAW_HEADER	Y	
lbt:IPAW_OUTPUT	Y	
lbt:IPAW_PARAM	Y	
lbt:IPAW_*	N	all the other tags from Table 1

Table 3: Configuration of the challenge JPIS. Unlike L&B tags in Table 1 namespace prefixes are mandatory for JP, shortcuts jps, jpw, lb, and lbt refer to namespaces described in Section 4.2

4.4.1 Find the process that led to Atlas X Graphic / everything that caused Atlas X Graphic to be as it is. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.

Inputs:: Identifier of the queried graphic file, denoted by its URL, referred to as `Atlas_X_Graphic` further on.

Outputs:: List of instances of workflow nodes that contributed to the queried file, starting from stage 5 (invocation of `convert`), traced by data dependencies, and ending with stage 1 (`align_warp`). Each entry in the output includes:

- jobid of the workflow node
- identifiers (URL) of the input and output files
- stage of the workflow, program name and parameter values

Pseudocode of the implementation is shown in Algorithm 1. The implementation uses JPIS query to find the workflow node which produced `Atlas_X_Graphic`. Starting from this node, the workflow dependency graph is searched in a reversed order (steps 3-6), using value of the `ancestor` attribute retrieved from JPPS repeatedly. Finally the list is sorted and printed. Sample output can be found in Appendix B.

In this query implementation we trade off performance for readability of the code. All the JPPS queries, which may easily become a bottleneck of the whole system, can be avoided, provided that JPIS configuration includes all the retrieved attributes (it is true for the configuration in Table 3). The queries can be also merged together in order to hit JPIS only once for each job.

4.4.2 Find the process that led to Atlas X Graphic, excluding everything prior to the averaging of images with softmean.

Inputs:: Identifier of the queried graphic file

Outputs:: Same as for Query #1 (Section 4.4.1)

The implementation is exactly the same as Query #1 with the only difference that the graph search (loop 3 7 in Algorithm 1) is terminated also when a node matching `IPAW_PROGRAM = 'softmean'` is found.

4.4.3 Find the Stage 3, 4 and 5 details of the process that led to Atlas X Graphic.

Inputs:: Identifier of the queried graphic file

Outputs:: Same as for Query #1 (Section 4.4.1)

Exactly the same as Query #1, with the final output filtered to contain only jobs having `IPAW_STAGE` equal to one of 3, 4, 5. Such implementation is not optimal but more general, we do not impose any special semantics on the value of the `IPAW_STAGE` attribute. With the additional knowledge that a node is preceded in the workflow only with nodes of lower stage number, the search could be cut at `IPAW_STAGE = 3`, similarly to Query #2.

4.4.4 Find all invocations of procedure `align_warp` using a twelfth order nonlinear 1365 parameter model (see model menu describing possible values of parameter `"-m 12"` of `align_warp`) that ran on a Monday.

Inputs:: N/A

Outputs:: Time, stage, program name, inputs, outputs of the matching workflow nodes

JPIS is queried for jobs matching `IPAW_PROGRAM = 'align_warp'` and `IPAW_PARAM = '-m 12'`. Among the other attributes the job registration time is also retrieved, and the output filtered to jobs that run on Monday.

Job registration time, i.e. the submission time, is only an approximation of running time (the job may have spent long time in a queue). The actual job run time is available in the LB trace, though the current JP implementation cannot extract it yet. Therefore this is a technical only, not principal restriction.

The filter “ran on Monday” is quite challenging. Currently, we implement it at the client side which is not a scalable solution – the number of retrieved records can grow unacceptably. However, we are working on a *type plugin* concept that extends JPIS data processing capabilities in order to allow efficient implementation of similar queries.

Algorithm 1 Pseudocode of Query #1

1. JPIS query: find jobid of job having `lbt:IPAW_OUTPUT = 'Atlas_X_Graphic'`
2. initialize `job_list` with the retrieved jobid
3. **while** there are unprocessed elements in `job_list` **do**
4. pick an unprocessed element `job`
5. JPSS query: for `job` retrieve all values of `lbw:ancestor` and insert each one into `job_list` unless it is already there
6. **end while**
7. JPSS query: for each `job` in `job_list` retrieve values of attributes:
`lbt:IPAW_INPUT, lbt:IPAW_OUTPUT,`
`lbt:IPAW_PROGRAM, lbt:IPAW_PARAM, lbt:IPAW_STAGE`
8. sort `job_list` according to `lbt:IPAW_STAGE`
9. pretty-print `job_list`, including all the retrieved attributes

4.4.5 Find all Atlas Graphic images outputted from workflows where at least one of the input Anatomy Headers had an entry `global_maximum = 4095`.

Inputs:: N/A

Outputs:: List of Atlas Graphic files matching the query

JPIS is queried for jobs matching `IPAW_HEADER = 'global_maximum 4095'` (and `IPAW_PROGRAM = 'align_warp'` eventually). The results of the query (jobid s of the matching jobs) are used to seed a graph search similar to Query

#1 (Section 4.4.1) but following the `successor` attribute of workflow nodes rather than `ancestor`. The output files of nodes having `IPAW_STAGE = 5` are gathered and sorted to exclude multiple occurrences.

An expression `IPAW_PROGRAM = 'convert'` can be used instead of `IPAW_STAGE = 5` as a condition identifying the final output files. Alternatively, they can be identified as outputs of nodes which have no successors.

The code can be also easily modified to record the graph traversal (details on workflow nodes) leading to a particular file, and display it with the file in a similar way as in previous queries.

4.4.6 Find all output averaged images of softmean (average) procedures, where the warped images taken as input were `align_warped` using a twelfth order nonlinear 1365 parameter model, i.e. "where softmean was preceded in the workflow, directly or indirectly, by an `align_warp` procedure with argument `-m 12`".

Inputs:: N/A

Outputs:: List of identifiers (URL s) of direct outputs of `softmean` invocations in workflows matching the query

JPIS is queried to retrieve `IPAW_PROGRAM = 'align_warp'` jobs having `IPAW_PARAM = '-m 12'`. The result is used to seed graph search, following the `successor` attribute. The search is cut at `IPAW_PROGRAM = 'softmean'`, and its outputs are printed.

4.4.7 A user has run the workflow twice, in the second instance replacing each procedures (`convert`) in the final stage with two procedures: `pgmtoppm`, then `pnmtjpeg`. Find the differences between the two workflow runs.

Inputs:: Identifier of the queried graphic file

Outputs:: Formatted in the same way as for Query #1, while the different workflow nodes are displayed.

We use Query #1 implementation to show details of the workflows. Then the differences are apparent – there is one more stage of the workflow, and `IPAW_PROGRAM` attribute values of the two final stages are `pgmtoppm` and `pnmtjpeg` respectively.

This is a slightly poor-man approach. It should be complemented with generating a graph representation of the result (all the information is present there) and feeding it into some graph comparison tools. However, this is processing

that is out of scope of JP. Therefore we consider this query to be addressed only partially.

4.4.8 A user has annotated some anatomy images with a key-value pair center = UChicago. Find the outputs of align_warp where the inputs are annotated with center = UChicago.

Job Provenance gathers and organizes information with the grid job being a primary entity of interest. Despite annotations of a *job* are its intrinsic part, direct annotations of *data* are not, therefore this query can't be answered in a reasonable way. This topic is discussed further in Section 5.2.

4.4.9 A user has annotated some atlas graphics with key-value pair where the key is studyModality. Find all the graphical atlas sets that have metadata annotation studyModality with values speech, visual or audio, and return all other annotations to these files.

Inputs:: Value of the `studyModality` annotation

Outputs:: List of matching graphics files, together with their additional annotations.

As mentioned with Query #8, JP does not provide means of adding annotations to data directly. However, annotations can be added to jobs (via JPPS interface) and it makes good sense to consider job outputs to be annotated with the job annotations too (see Section 5.2). We assume the annotations to be assigned to whole workflows (not its subjobs) in the form of JP attributes in a dedicated namespace, see <http://twiki.ipaw.info/Challenge/CESNET/Annotations> (referred to as *annot* further on). Then the implementation is a two stage query shown in Algorithm 2.

Algorithm 2 Pseudocode of Query #9

1. JPIS: find jobs (workflows) having `annot:studyModality = input value`
2. **for** each found workflow **do**
3. JPIS query: find jobs where `lbt:parent = wf` and `lbt:IPAW_STAGE = 5`
4. JPPS query: retrieve `lbt:IPAW_OUTPUT` and other annotations for the jobs
5. **end for**

Currently neither JPPS nor JPIS supports a query “all attributes of this job”. If the annotation names are not known a priori, the following approaches are possible:

- A simple workaround is storing all annotations in a similar way as IPAW_HEADER tag, in an attribute holding both annotation name and value. However, this approach would not allow more complex queries on the annotation values.
- A better workaround is attaching a known Annotation attribute to each job. This attribute would hold names of all existing annotations for this job. The user would first query for values of this attribute, and then choose which real annotations (JP attributes) to retrieve.
- Extending JPPS query interface to support queries like “all attributes of this job falling into namespace X”.

5 Lessons learned

5.1 Workflow representation

Recent development in the field of provenance puts clear emphasis in workflow processing, which can be clearly seen in the program of the IPAW 06 workshop [MF06] as well as in the specification of the current Provenance Challenge.

When JP was originally designed as a part of the gLite middleware, compound jobs (DAGs) were supported in gLite but their real usage was marginal. Therefore the current JP design does not include any explicit concept reflecting compound jobs.

However, provenance trace of a DAG job in gLite contains sufficient information to reconstruct the workflow structure:

- *Workflow components*: JDL of a submitted DAG (see Appendix A) contains nodes ClassAd attribute which is a list of nested JDL s of all the DAG subjobs. These are identified with their internal JDL names (align1 etc. in the example). During the DAG processing global jobids are assigned to the subjobs, and added to the subjob JDL s as their edg_jobid ClassAd attributes. Then the augmented DAG JDL is logged into L&B.
- *Dependencies among the subjobs*: DAG JDL also contains dependencies attribute, a list of pairs of node names expressing dependencies among them.

- *Membership in a workflow*: When a DAG with its subjobs is registered in L&B, the information “I belong to this DAG” is recorded for each subjob in its parent L&B attribute.

As complete L&B data are uploaded to JP, this information is available, allowing complete reconstruction of a DAG structure from JP data. However, most of the information is hidden rather deeply – contained in a specific L&B job attribute (JDL) which has rather complex structure itself. Therefore accessing the information via JP interface is not straightforward. In the following we discuss possible approaches to overcome this difficulty.

During the challenge implementation we defined the `ancestor` and `successor` attributes (Section 4.2) to describe the workflow structure in JP view of data more directly as well as to allow efficient implementation of its traversal (search). So far we generate values of these attributes semi-manually via an external utility. Given jobid of a DAG, the utility retrieves the appropriate DAG JDL from JPPS, parses its structure, extracts the mapping of internal names to jobids, and according to the `dependencies` list it stores the `ancestor` and `successor` values for each node back into JPPS as user annotations.

In this scenario, JP raw data on *one job*, the DAG, provide JP attributes of *other jobs*, the DAG nodes. Unfortunately, this way of processing does not fit into the current JP design where jobs are treated autonomously, a job raw data generate JP attributes of this job only.

Participation in the challenge showed this design drawback and motivated its ongoing revision. The following options are being considered:

- *Introduce the concept of compound job into JP*. Despite being quite straightforward, this approach goes somewhat against the original JP concepts, namely its generality and independence on the data content. JP itself was meant to define as few requirements on the job data as possible, leaving all the logic to job type specific plugins. On the contrary, whatever general compound job model we adopted, it would necessarily impose some restrictions that would appear at the JP level, not specifically for a certain job type.
- *Allow loopback query in JPPS*. In order to retrieve values of `ancestor`, `successor` or similar attributes for a job, JPPS would allow the specific job type plugin to query JPPS for attribute of another job (JDL of the DAG) and process it appropriately (the logic of the current external utility). This approach seems to be cleaner from the JP design point of view, however, it may imply non-trivial technical complications, namely in performance area.

Both these options are tentative, we have no finalized solution yet. In either case, the emerging discussion clearly demonstrates the contribution of the provenance challenge to our activity.

5.2 Data vs. process provenance

Job Provenance is job centric, it is focused on process view on the provenance problems. The computation (Grid job) is the primary subject about which provenance metadata is collected, not the results of the computation. Apparently, this is a different approach from many other provenance systems (see Section 6) that deal primary with data, not processes. Reasons for this difference can be found in the original design requirements – JP was designed to record information on jobs, not data.

EU Provenance project⁶ broadly defines *the provenance of a piece of data is the process that led to the data*. Within this view, recording a clear relationship between a piece of data (job output) and the process that led to this data (details on the grid job, stored as its JP record) should be sufficient to provide provenance of the data too. In our representation of the challenge workflow the relationship is recorded in the IPAW_OUTPUT job attribute.

Formulation of the challenge queries strongly suggests a data oriented approach. However, we can further classify both the queries and their solution in JP as follows:

- #1 3 and #7 lead to a JP search of concrete value (the Atlas X Graphics file name) of IPAW_OUTPUT, retrieving details on the workflow execution then. Therefore we can treat them as process-related. As discussed above, the recorded relationship of a job and its output is the necessary glue that allows queries originally formulated as data-related to be served by JP.
- #5 and #6 search for job outputs satisfying conditions on a process parameter (“-m 12” in #6) or data feature which is computed during the process (global_maximum in #5), both being an attribute of the processes. Therefore they are again process-related, despite the main result of the query is a list of output file names representing data items.
- #8 and #9 query on data annotations, therefore being data-related. As discussed in Section 4.4.9 it makes sense to understand process annotations to be also annotations of its outputs, therefore #9 can be solved by JP. On the contrary, #8 refers to input annotations which can t be directly handled by JP.

⁶<http://twiki.gridprovenance.org/bin/view/Provenance/ProjectInformation>

To sum up, a fairly straightforward solutions of almost all the challenge queries show that the data-oriented queries can be mapped into JP quite easily, provided that the necessary relationship to inputs and outputs is recorded with the jobs. The only unsolved query is specific in the sense that there is no processing involved. This kind of queries falls beyond a feasible scope of JP, unless artificial processes are introduced.

5.3 Semantics interpretation levels

In Section 5.1 we describe introduction of the ancestor and successor attributes. However, agreement on this approach was not straightforward, we discussed an alternate solution of keeping the whole logic of parsing JDL and extracting the required information in the client program which implements the challenge queries.

These discussions document certain underestimation of the importance to define strictly the levels where the *semantics of JP data* is interpreted. In general, the level defines where JP attributes (the logical view, Section ??) are extracted from the raw data (if it is done at all for a particular piece of the raw data). We identify three such levels:

- *JP intrinsic* interpretation level should be kept as minimal and as fixed as possible. JP itself is aware of a job as the primary entity, its minimal metadata kept by JPPS (currently owner and registration time), and files attached to the job. Foreseen extensions are sharing files among jobs (it may be desirable for example in case of workflow sandboxes), or some reflection of the compound jobs structure (see discussion in Section 5.1). No further assumptions on the data semantics are done at this level, especially in relation to the content of the files.

- *Job type* interpretation level is specific to for example particular Grid middleware (currently gLite), it should understand structure and processing of this job type, however, it should not deal with the user payload of the jobs.

For example, parsing and processing the L&B data (yielding the set of appropriate attributes) including JDL parsing, and extracting file names from uploaded gLite job sandboxes belongs to this level. It is desirable to implement this interpretation in terms of JP plugins in order to shield the user or other application from this data interpretation (which may be non-trivial).

In our discussions on the workflow structure we also concluded that extracting it from the gLite JDL belongs to this level.

- *Application or user specific* level is related to the job payload. In the challenge, example of this level is the specific treatment of IPAW_* tags.

This level may be left to client programs (as done in the challenge). Alternatively, it can be implemented as application-specific JP plugins (typically to extract specific attributes from some sandbox file) to serve the needs of a bigger user community.

Despite being aware of this classification before, the challenge showed that boundaries between these levels tend to be rather vague and subjective. Therefore making clear design decision with each type of job is critical. On the other hand, it was exactly the broad scale of the semantics interpretation levels that allowed process-oriented JP to handle data-related queries in a fairly straightforward way.

In addition, solving these problems motivated a design of *plugin chaining*, an extension to the existing JP plugin mechanism. In order to retrieve a value of a particular attribute (some JDL field for example) JP should call a specific plugin (ClassAd parser), which calls another plugin (L&B) in turn to retrieve another attribute (the JDL) from the raw data file. In the time of writing this manuscript development of prototype implementation is in progress.

6 Related work

JP uses middleware components capable to automatically create basic provenance data from captured internal events of workload management and data references of Grid jobs. This approach minimizes adaptations required in applications and middleware. It is best focused on process level of provenance, the computation (Grid job) is the primary subject about which provenance metadata is collected, not the inputs or results of the computation. On the other hand, using the references to inputs and outputs of jobs and generic extensibility and annotation features, the JP could be queried to provide provenance on the data level (related to data files or other results). However, thus provided information may be incomplete in principle, as JP is not deeply integrated with data storage.

Number of systems proposed have taken radically different approach by tightly integrating exhaustive provenance tracking in workflow execution environments or application frameworks. Some of them are oriented towards relatively straightforward extensions to a workflow enactment engine so it can capture basic provenance data automatically during workflow runtime (e.g. [BD07]), others strive to cater for analysis combining origin and domain knowledge about workflow design, execution, interpretation [Zha07]. Yet another set of models concentrates on efficient handling of specific types of workflows [BML07]. It

is definitely possible to collect much more complete provenance records this way, especially when data storage is also embedded in or tightly controlled by the workflow system. However, there are considerable disadvantages too, especially very limited support for “legacy” scripted application workflows and a proneness to creation of too large amounts of provenance data that may be unfeasible to handle when vast number of jobs on a large scale Grid has to be tracked.

The basic idea behind collection part of JP – automatic creation of persistent logs of the work done (combination of high level observed and disclosed provenance approaches) – is similar to principles of CODESH project [Bou06]. However, CODESH is oriented to interactive work (virtual sessions), while our system is heavily biased towards storing and searching provenance data on vast amounts of batch jobs in Grid environment in an efficient way. Another system which is generic and simple similarly to ours is Karma Provenance Framework [Sim06].

In [Bra06], more fine-grained observed provenance approach is taken at operating system level and associated challenges related to data granularity, overheads, and irrelevant provenance data pruning are discussed. Associated query model is discussed in [Sel07].

7 Conclusions

The name *First Provenance Challenge* turned to have a very literal meaning for our team. It was the first opportunity to compare the capabilities of Job Provenance with other provenance systems, and it became particularly challenging due to its emphasis in fields that were not our original design priorities.

Virtually all the challenge queries are related to the provenance of data, while JP is strongly process (job) oriented. However, we managed to find a suitable representation of the challenge workflow so that most of the queries can be mapped to JP in a reasonable and straightforward way. Eight out of nine queries were solved; the remaining one clearly identifies the area of pure data annotation with no involved processing which falls beyond a feasible scope of JP.

Solving the challenge also identified certain JP design drawbacks, namely the lack of explicit support for compound jobs (workflows), and it initiated their reconsideration. Other extensions to the design (loopback query and plugin chaining) also emerge directly from the challenge requirements.

Altogether we consider the participation in the First Provenance Challenge successful, proving that the intended generality of JP can be leveraged even in its not-directly-foreseen applications.

8 Acknowledgment

This work was done within the EGEE-II project funded by the European Union, INFISO-RI-031688.

9 Appendix. Fragment of workflow JDL

The strings BASEx, REFERENCE, and ATLAS are placeholders for real file names (URL s).

```
[
  type = "dag";
  nodes = [
    align1 = [ description = [
      executable = "align.sh";
      arguments = "BASE1 REFERENCE";
    ] ];
    align2 = [ description = [
      executable = "align.sh";
      arguments = "BASE2 REFERENCE";
    ] ];
    ...
    softmean = [ description = [
      executable = "softmean.sh";
      arguments = "BASE1 BASE2 BASE3 BASE4 ATLAS";
    ] ];
    ...
  ];
  dependencies = {
    { align1, reslice1 },
    ...
    { { reslice1, reslice2, reslice3, reslice4 }, softmean },
    ...
  };
  ...
]
```

10 Appendix. Query #1 sample output

```
$ ./query1.pl gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla-x.gif 2>/dev/null
```

```
Results
```

```
=====
```

```
jobid https://skurut1.cesnet.cz:9000/hvkvZCsRsiqrxs5K_bo7Ew:
```

```
  attr IPAW_STAGE: 5
```

```
  attr IPAW_PROGRAM: convert
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla-x.pgm
```

```
  attr IPAW_OUTPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla-x.gif
```

```
  attr CE: skurut17.cesnet.cz:2119/jobmanager-lcgpbs-voce
```

```
jobid https://skurut1.cesnet.cz:9000/02ZaAADKyebzggYPp4M9tA:
```

```
  attr IPAW_STAGE: 4
```

```
  attr IPAW_PROGRAM: slicer
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla.hdr
```

```
                  gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla.img
```

```
  attr IPAW_OUTPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla-x.pgm
```

```
  attr CE: skurut17.cesnet.cz:2119/jobmanager-lcgpbs-voce
```

```
jobid https://skurut1.cesnet.cz:9000/wGMnTvCILTISTi7Z0QwfTQ:
```

```
  attr IPAW_STAGE: 3
```

```
  attr IPAW_PROGRAM: softmean
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy1-resliced.img
```

```
  ...
```

```
  attr IPAW_OUTPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla.img
```

```
                  gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/blabla.hdr
```

```
  attr CE: skurut17.cesnet.cz:2119/jobmanager-lcgpbs-voce
```

```
jobid https://skurut1.cesnet.cz:9000/9d0XMwfPuefR9woAFkDp1Q:
```

```
  attr IPAW_STAGE: 2
```

```
  attr IPAW_PROGRAM: reslice
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy3.warp
```

```
  ...
```

```
  attr IPAW_OUTPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy3-resliced.img
```

```
  ...
```

```
  attr CE: skurut17.cesnet.cz:2119/jobmanager-lcgpbs-voce
```

```
jobid https://skurut1.cesnet.cz:9000/RglBtUz0IzwSeM32KLnHPg:
```

```
  attr IPAW_STAGE: 2
```

```
  attr IPAW_PROGRAM: reslice
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy4.warp
```

```
  ...
```

```
  ...
```

```
jobid https://skurut1.cesnet.cz:9000/wdWQHL0-RXkd3VeNcSrTaw:
```

```
  attr IPAW_STAGE: 2
```

```
  attr IPAW_PROGRAM: reslice
```

```
  attr IPAW_PARAM:
```

```
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy1.warp
```

```
  ...
```

```

...

jobid https://skurut1.cesnet.cz:9000/xwIsN2JgGfsRuvYwh0QXsw:
  attr IPAW_STAGE: 2
  attr IPAW_PROGRAM: reslice
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy2.warp
  ...
...

jobid https://skurut1.cesnet.cz:9000/yM3sz8v6WCIPgi5-0m8L4w:
  attr IPAW_STAGE: 1
  attr IPAW_PROGRAM: align_warp
  attr IPAW_PARAM: -m 12, -q
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy4.img
                    gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/reference.img
  attr IPAW_OUTPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy4.warp
  attr CE: skurut17.cesnet.cz:2119/jobmanager-1cgpbs-voce

jobid https://skurut1.cesnet.cz:9000/s47ihjBHQxqPkkNwA2iazg:
  attr IPAW_STAGE: 1
  attr IPAW_PROGRAM: align_warp
  attr IPAW_PARAM: -m 12, -q
  attr IPAW_INPUT: gsiftp://umbar.ics.muni.cz:1414/home/mulac/pch06/anatomy2.img
  ...
...

```

References

- [BD07] Barga R. S. and Digiampietri L. A.: Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 2007.
- [Bou06] Bourilkov D. et al. Virtual logbooks and collaboration in science and software development. In: Moreau L. and Foster I. (editors), *International Provenance and Annotation Workshop (IPAW 06)*, Chicago, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.
- [BML07] Bowers S., McPhillips T. and Ludaescher B.: A provenance model for collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
- [Bra06] Braun U. et al.: Issues in automatic provenance collection. In: Moreau L. and Foster I. (editors), *International Provenance and Annotation Workshop (IPAW 06)*, Chicago, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.

- [JPUG06] CESNET: *gLite job provenance service user s guide.*. Available online⁷.
- [Dvo06] Dvořák F. et al.: gLite job provenance. In: Moreau L. and Foster I. (editors), *International Provenance and Annotation Workshop (IPAW 06)*, Chicago, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.
- [EGEE04] EGEE Project: *EGEE Middleware Design Release 1*. Deliverable EGEE-DJRA1.2⁸, 2004.
- [Ave03] Avellino G. et al.: The first deployment of workload management services on the EU DataGrid testbed: feedback on design and implementation. In: *Computing in High Energy and Nuclear Physics (CHEP03)*, La Jolla, CA, March 2003.
- [Kou04] Kouřil D. et al. Distributed tracking, storage, and re-use of job state information on the grid. In: *Computing in High Energy and Nuclear Physics (CHEP04)*, 2004.
- [Lau04] Laure E. et al.: Middleware for the next generation grid infrastructure. In: *Computing in High Energy Physics and Nuclear Physics (CHEP 2004)*, 2004.
- [Mat07] Matyska L. et al.: *Job tracking on a grid the Logging and Bookkeeping and Job Provenance services*. In preparation.
- [MF06] Moreau L. and Foster I. (editors): *International Provenance and Annotation Workshop (IPAW 06)*, Chicago, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.
- [Pac06] Pacini F. et al.: Job description language attributes specification. EGEE Project, 2006. Available online⁹.
- [Pet07] Petřek M. et al.: Multiple ligand trajectory docking – a case study of complex grid jobs management. Accepted for *EGEE User Forum*, 2007.
- [RLS98] Raman R., Livny M. and Solomon M.: Matchmaking: Distributed resource management for high throughput computing. *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, 1998.
- [Sel07] Seltzer M. et al.: Passing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 2007.

⁷<http://egee.cesnet.cz/en/JRA1/>

⁸<https://edms.cern.ch/document/487871/>

⁹<https://edms.cern.ch/file/590869/1/EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8.pdf>

- [Sim06] Simmhan Y. L. et al.: Performance evaluation of the karma provenance framework for scientific workflows. In: Moreau L. and Foster I. (editors), *International Provenance and Annotation Workshop (I-PAW 06)*, Chicago, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.
- [Zha07] Zhao J. et al.: Mining taverna s semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 2007.