CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **3**:1–2 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

gLite Job Provenance — a job-centric view

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

CESNET, z. s. p;o., Zikova 4, Praha, Czech Republic

SUMMARY

Job Provenance (JP), part of the gLite Grid middleware, is a 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, and annotations. During the first provenance challenge, we explored the relation between a specific job-centric Grid oriented provenance and a more general data provenance approach. The challenge represents a usecase which emphasizes fields that were not priorities in the original JP design. However, we proved that the design is sufficiently general to cope with this mode of use. We also identified several areas where it is feasible to extend the current implementation.

1. 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 (e. g. 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 important because the user does not have a complete control over the resources used for her computation. 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 the *Job Provenance* (JP). At the time of writing this manuscript, JP is deployed experimentally on a part of the EGEE* infrastructure.

JP is expected to be used by individual end-users for long-term storage of job data, by user communities for collaboration†, as well as by various Grid monitoring and statistics gathering tools. Some expected use cases are described in [7, 15]. However, as the goal of JP is to encompass the whole EGEE-like Grid production infrastructure, there is not yet sufficient experience that would lead to clear understanding of usage patterns. This uncertainty 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.

2. Job Provenance design

Pragmatic implementational requirements on JP 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, yielding growing storage requirements‡ that must be kept reasonable by making job records as compact as possible. At the same time efficient queries are required, which is virtually impossible on huge number of compact records. The overall JP design tries to keep these requirements in a reasonable balance. This section highlights key design features, details are out of scope of this paper and those relevant for the challenge can be found in [13] and in an extended form in [9, 15].

2.1. gLite jobs

The only way the user can access computational resources in gLite middleware [14] is through a *job*§. It is processed by several middleware components, and the processing trace is gathered by the *Logging and Bookkeeping* service (L&B) [12, 8, 15]. After the job completion, data from L&B are uploaded to JP, together with those job inputs that are not kept reliably elsewhere. Both during the job active life and afterwards the user can add arbitrary job annotations.

2.2. Data in JP

Job data enter JP in two forms: a “name = value” *tags*, and uploaded files. On the other hand, for queries a *logical view* is used, where all data are expressed in terms of job *attributes.*

*`http://www.eu-egee.org`

†An application-oriented use case [17] is being prepared in parallel with this manuscript.

‡E. g. EGEE aims at 1 M jobs per day; implications for JP are given in [15], as well as deployment considerations.

§Both “simple” jobs and elementary workflows — directed acyclic graphs (DAG) are supported.

An attribute is named, the name must fall to a specified namespace, its value may have an arbitrarily complex structure. Tags map to attributes directly, the uploaded files must be processed by file-type specific *JP plugins* which understand their internal structure and distill the attributes.

2.3. JP components

Primary storage (JPPS) stores, in a compact form, the job data permanently. Only small number of long-lived JPPS instances with large data storage is expected on a Grid (e. g. one per several virtual organizations). *Index server* (JPIS) provides the query interface to the end user. JPIS's can be set up semi-dynamically according to the needs of even fairly small user communities. A JPIS can be associated with more than one JPPS, but typically it stores just a fraction of data from these JPPS's.

3. The challenge

In this section we describe 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 in the introduction to this issue or at `http://twiki.ipaw.info/bin/view/Challenge`.

3.1. Workflow representation

We implement the challenge workflow as a gLite DAG job, see [13]. The structure of the DAG follows the specified workflow exactly, using the following mapping:

- *Procedures* become nodes of the DAG, they are turned into normal gLite jobs during the DAG processing. 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.
- *Data items*, both input and output, are external files w. r. t. the workflow implementation. Each job is responsible for downloading all its inputs and uploading all its outputs.

For the challenge we put the files on a dedicated GridFTP server and access them (both down- and upload) with the `gsiftp://` protocol. Consequently, the data items are identified with their full URL. Alternatively, the user might want to use the gLite data services [14] and identify files with GUID's or logical file names.

3.2. Provenance trace

The *raw representation* of the provenance trace is formed by the L&B data (containing all sufficient information) uploaded into JP. However, as discussed in [15], 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:

- JP system attributes: job identifier, job owner, and submission time.
- Attributes from L&B trace: id of job "parent" (whole workflow), name of computing element where the job run.
- Attributes from JDL, describing workflow structure: `ancestor` and `successor`, i. e. jobid(s) of immediately preceeding/following job(s) in the workflow). Currently these are not available for direct querying [13], therefore we supply them with an external utility parsing the JDL.
- Challenge-specific user tags, logged via L&B (Tab. I)[¶].

Some of the attributes may occur multiple times, e. g. as `softmean` must have been preceeded by 4 `reslice`'s in the challenge workflow, there are 4 occurrences of the `ancestor` attribute of the `softmean` nodes.

3.3. Characteristics of the system and provenance representation

In this section we comment specifically on the provenance systems classification criteria summarized in the introduction of this journal issue [16].

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 infrastructure (VOCE[‖] virtual organization), using prototype instances of L&B and JP services.

C1.3 Representation Technology. In JPPS, files from the execution environment, including L&B data, are kept in their native form. In JPIS, data are stored in RDBMS. On the logical level, data are represented as `namespace:name=value` attributes, see Sect. 2.2 and [13].

C1.4 Query Language. SQL-like but strongly restricted — one logical table, simplified structure of the 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.

C1.6 Challenge Implementation. Full execution, including the image processing programs.

C2.1 Includes Workflow Representation. The challenge implementation includes an explicit representation (in gLite JDL) and uses it to follow the workflow structure. However, it can be more or less unambiguously deduced from data dependencies too.

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

C2.3 Annotations. Considered to belong to provenance and fully supported in JP.

[¶]Their meaning 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 preceeding nodes in the workflow execution.
[‖]`http://egee.cesnet.cz/en/voce/`

Table I. Specific challenge L&B tags attached to each DAG node

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)

C2.4 Time. Timestamps are included with 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.

3.4. Provenance queries

Basic building blocks of the implementation are:

- *JPPS query*: given a jobid, return requested attributes of the job.
- *JPIS query*: find jobs matching criteria expressed with attributes, return specified attributes.

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

For the purpose of the challenge, following the intended deployment scenario [15], we set up a JPIS with specific configuration including all required attributes and indices [13].

**Available at `http://glite.cvs.cern.ch:8180/cgi-bin/glite.cgi/org.glite.jp.index/examples/pch06/`

3.4.1. Find the process that led to Atlas X Graphic. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.

Inputs: URL of the queried graphic file, 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 Alg. 1. Starting from the node which produced `Atlas_X_Graphic`, 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 [13].

Algorithm 1 Pseudocode of Query #1

```
1: JPIS query: find jobid of job having 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:   JPPS query: for job retrieve all values of ancestor
              and insert each one into job_list unless it is already there
6: end while
7: JPPS query: for each job in job_list retrieve values of attributes:
          IPAW_INPUT, IPAW_OUTPUT,
          IPAW_PROGRAM, IPAW_PARAM, IPAW_STAGE
8: sort job_list according to IPAW_STAGE
9: pretty-print job_list, including all the retrieved attributes
```

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. The queries can be also merged together in order to hit JPIS only once for each job.

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

The implementation is exactly the same as Query #1 with the only difference that the graph search (loop 3–6 in Alg. 1) is terminated also when a node matching `IPAW_PROGRAM = 'softmean'` is found.

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

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 preceeded in the workflow only with nodes of lower stage number, the search could be cut at `IPAW_STAGE = 3`, similarly to Query #2.

3.4.4. 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.*

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.

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.

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

Outputs: List of Atlas Graphic files matching the query

JPIS is queried for jobs matching `IPAW_HEADER = 'global_maximum 4095'`. The results are used to seed a graph search similar to Query #1 but following `successor` 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.

3.4.6. Find all output averaged images of softmean (average) procedures, where `softmean` *was preceded in the workflow, directly or indirectly, by an* `align_warp` *procedure with argument* `-m 12`*.*

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

[††]Approximation only. Exact run time is available in L&B data too, though not supported by current JP implementation yet.

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.

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

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

3.4.8. 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 (see [13] for more discussion).

3.4.9. 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. The implementation is a two stage query shown in Alg. 2.

Algorithm 2 Pseudocode of Query #9

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

4. Related work

JP gathers information on jobs first, with relationship to data being secondary. Therefore the captured data provenance information may become incomplete easily.

Number of other systems take different approach by capturing provenance data automatically during workflow runtime [3, 2, 11, 1], others combine knowledge about workflow design, execution, interpretation [22, 10]. Another set of models concentrates on efficient handling of specific types of workflows [5, 18]. These approaches typically gather more complete provenance record, especially when data storage is embedded or tightly controlled by the workflow system. On the other hand, support for "legacy" applications, which do not fit into the provenance framework easily, is more complicated.

JP approach is similar, though applied in quite different environment, to principles of `CODESH` [4] or Karma Provenance Framework [21, 20].

The amount and redundancy of data gathered in this way may also turn to be a limiting factor in large deployments. The work [6, 19] presents more fine-grained approach taken at operating system level, including also issues on pruning irrelevant data.

5. 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 using an explicit binding of a job with its output (annotations of a job that produced a piece of data become annotations of the data too). Seven out of nine queries were solved completely (#1–6, #9), one partially (#7); the remaining #8 clearly identifies the area of pure data annotation with no involved processing which falls beyond a feasible scope of JP.

The recent development in the field of provenance puts clear emphasis in workflow processing. However, following a rather marginal workflow support in gLite, the current JP design does not reflect workflows explicitely in its core. On the other hand, all necessary data on both workflow components and its structure are present in JP through the original submitted job description. Hence we defined extra job attributes that reflect the workflow structure in a directly queryable form. For the purpose of the challenge, we used an external utility to populate them; for a long term, the challenge motivated design extensions that would reflect workflow structure directly.

Other extensions (loopback query and plugin chaining) also emerge directly from the challenge requirements. More discussion on the lessons learnt can be found in [13].

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.

Acknowledgment

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

REFERENCES

1. Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. In *Proc. IPAW'06*, volume 4145 of *LNCS*. Springer, 2006.
2. Roger Barga. Automatic generation of workflow execution provenance. In *Proc. IPAW'06*, volume 4145 of *LNCS*. Springer, 2006.
3. Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 2007.
4. Dimitri Bourilkov, Vaibhav Khandelwal, Archis Kulkarni, and Sanket Totala. Virtual logbooks and collaboration in science and software development. In *Proc. IPAW'06*, volume 4145 of *LNCS*. Springer, 2006.
5. Shawn Bowers, Timothy McPhillips, and Bertram Ludaescher. A provenance model for collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
6. Uri Braun, Simson Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in automatic provenance collection. In *Proc. IPAW'06*, volume 4145 of *LNCS*. Springer, 2006.
7. CESNET. gLite Job Provenance service User's Guide. `http://egee.cesnet.cz/en/JRA1/`.
8. František Dvořák et al. Services for tracking and archival of grid job information. In *Proc. Cracow Grid Workshop*, pages 255–263, 2005.
9. František Dvořák et al. gLite Job Provenance. In *Proc. IPAW'06*, volume 4145 of *LNCS*, pages 246–253. Springer, 2006.
10. Jennifer Golbeck and James Hendler. A semantic web approach to tracking provenance in scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
11. 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 (OPODIS'04)*, volume 3544 of *LNCS*. Springer, 2004.
12. D. Kouřil 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.
13. Aleš Křenek et al. gLite Job Provenance — a job centric view, extended version. Technical report, CESNET, 2007. `http://www.cesnet.cz/doc/techzpravy/`.
14. E. Laure et al. Middleware for the next generation grid infrastructure. In *Computing in High Energy Physics and Nuclear Physics (CHEP 2004)*, 2004.
15. Luděk Matyska et al. Job tracking on a grid—the Logging and Bookkeeping and Job Provenance services. Technical Report 4/2007, CESNET, 2007. `http://www.cesnet.cz/doc/techzpravy/`.
16. Luc Moreau et al. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007.
17. Martin Petřek et al. Multiple ligand trajectory docking — a case study of complex grid jobs management, 2007. Accepted for presentation at EGEE User Forum.
18. Norbert Podhorszki, Bertram Ludaescher, Ilkay Altintas, Shawn Bowers, and Timothy McPhillips. Recording data provenance for Kepler scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
19. Margo Seltzer, David A. Holland, Uri Braun, and Kiran-Kumar Muniswamy-Reddy. Pass-ing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 2007.
20. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Querying capabilities of the karma provenance framework. *Concurrency and Computation: Practice and Experience*, 2007.
21. Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru. Performance evaluation of the Karma provenance framework for scientific workflows. In *Proc. IPAW'06*, volume 4145 of *LNCS*. Springer, 2006.
22. Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 2007.