Grid workflow job execution service ‘Pilot’

Lev Shamardin, Alexander Kryukov, Andrey Demichev and Vyacheslav Ilyin
Division of Theoretical High Energy Physics, Scobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Leninskie Gory 1, GSP-1, Moscow, 119991, Russia.
E-mail: shamardin@theory.sinp.msu.ru

Abstract. ‘Pilot’ is a grid job execution service for workflow jobs. The main goal for the service is to automate computations with multiple stages since they can be expressed as simple workflows. Each job is a directed acyclic graph of tasks and each task is an execution of something on a grid resource (or ‘computing element’). Tasks may be submitted to any WS-GRAM (Globus Toolkit 4) service. The target resources for the tasks execution are selected by the Pilot service from the set of available resources which match the specific requirements from the task and/or job definition. Some simple conditional execution logic is also provided.

The ‘Pilot’ service is built on the REST concepts and provides a simple API through authenticated HTTPS. This service is deployed and used in production in a Russian national grid project GridNNN.

1. Introduction
Workflow jobs are very important for efficient grid utilisation because they allow to automate long computations involving multiple dependent stages of processing. Such computations are common in many fields of physics and chemistry research. In this work we describe the architecture and interfaces of the grid job execution service ‘Pilot’ which is used for running workflow jobs in GridNNN project [1]. In contrast to many modern grid services which are implemented as WSRF web services [2], it is built as a RESTful grid service [3, 4]. A RESTful Web service provides access to a collection of resources using standard HTTP methods like GET, PUT, POST or DELETE and representations which are agreed with the service consumers. It is very important here that the logical operations with the resource conceptually match the HTTP method used for the operation. Grid service built on REST architecture differs from a web service by taking special care of resource lifetime management and resource creation process. This allows to design a simple but powerful API while preserving all properties of a grid service.

There are many other known grid workflow systems, for a comprehensive review and classification see for example [5]. Our ‘Pilot’ service provides a combination of features vital for using workflows with HPC clusters and MPI computational jobs which includes support for abstract workflow specification, centralized dynamic just-in-time scheduling strategy for execution, task-level fault tolerance. Some unique features of ‘Pilot’ are support for partial workflow execution in case of individual task failures. It also support dynamic group task scheduling: a user may require an arbitrary number of workflow steps to be scheduled to the same (but not known in advance) grid resource.

In this work we describe the purpose of the service, the notion of workflow jobs and tasks which can be executed and discuss the implementation of the Pilot grid service.
2. Jobs and Tasks

 Computations carried out in the GridNNN grid infrastructure are represented as jobs, which are composed of tasks. A grid job, or simply a job is a directed acyclic graph composed of tasks. Each node of the graph corresponds to a task, edges define the order of tasks execution. A task is a minimal executable element of a job which is a submission of an executable to some GridNNN Computing Element (CE) which is generally a host running a Globus WS-GRAM grid service. Each task has a set of resource requirements attached defining which grid resources are compatible with a task. These requirements may specify minimal amounts of available memory and storage, number of CPUs and so on. Implicit requirements are satisfied by any CE in the grid.

 The job execution service accepts a job from a user and executes this job on a set of grid resources. It will track the execution progress of each task and will try to run any task for which the prerequisites are met on the best resource matching the task resource requirements. It is possible to specify the criteria to determine the success of completion for each task based on its exit code, and to stop the execution of the entire job if some task fails, or only part of the job which will not be able to complete due to failed task.

2.1. Syntax

 Jobs and Tasks definitions are written as JSON documents, there is a JSON Schema definition for the jobs/tasks formats [6, 7]. A job definition consists of a list of tasks, indications of task dependencies and some common options for the whole job. default resource requirements for all tasks. The definitions for tasks may be done inline in the job definition file, or stored in separate files referenced from the job definition. Listing 1 provides an example of a job definition. This definition corresponds to a simple job with two tasks, a and b, and structure of a — b. Task a is defined in a separate file task_a.js, provided on listing 2. The definition of task b is contained in the job definition.

 These examples demonstrate some of the features of the jobs and tasks definitions for the Pilot service:

 - Task dependencies; task b is started only after completion of task a.
Listing 2 Example task definition (task_a.js).

```json
{
    "version": 2,
    "description": "example task defined in a separate file",
    "executable": "worker.sh",
    "input_files": {
        "worker.sh": "gsiftp://tb05.ngrid.ru/home/john/examples/worker.sh"
    },
    "stdout": "test.log",
    "environment": { "CFLAGS": "-O2" }
}
```

- Staging of standard input/output as well as arbitrary other files to and from external storage.
- Using paths relative to `default_storage_base` instead of absolute URLs in files specifications.
- Setting environment variables prior to task execution.

There are many other features supported by the jobs and tasks syntax not covered in this introductory description. Complete documentation on the job, tasks and resource requirements syntax as well as a number of examples is available at the GridNNN project web site [1] under Pilot service documentation section (http://www.ngrid.ru/sw/pilot/docs/).

2.2. Resource requirements

Resource requirements for tasks may be specified both in job and in task definitions. The requirements keys specified in the task override the values from the job. Pilot supports all of the following:

- Node and operating system parameters: CPU speed and architecture, available memory and disk storage, SMP size, operating system name and release.
- Batch system parameters like time limits, queue lengths.
- Requirements for preinstalled software.

GridNNN information system publishes information about the software preinstalled on the CEs by system administrators and virtual organizations, including software versions. Pilot support specifying software requirements optionally based on expressions with software versions. For example software requirements of “mvapich, abinit > 6, gcc == 3.5.5” will be satisfied by any CE which has any version of `mvapich` package, `abinit` with any version greater than 6 and `gcc` of the specified version 3.5.5.

3. Pilot

Pilot is a grid service which can run jobs and tasks as defined in section 2. The service is developed as a part of the GridNNN project. It is designed as a RESTful grid service [3, 4]. All requests to service are done through authenticated HTTPS connections with peer authentication using user X.509 certificate or proxy certificate [8]. Pilot uses VOMS extensions [9] to determine the virtual organization which the user belongs to. Requests with a certificate without VOMS extensions are allowed only to access existing data owned by the user on the service.

All requests to the service which have payload except the request headers are sent in JSON format. Most of the service replies are also JSON documents, however a client may specify alternative representations in Accept header, and the service will try to satisfy such requests if possible. This allows for example to produce a human-friendly information about the job status as an HTML document if the job URI is opened in a web browser.
| GET                | PUT             | POST                                      | DELETE               |
|--------------------|-----------------|-------------------------------------------|----------------------|
| /jobs/             | N/A             | Submit a new job                          | N/A                  |
|                    |                 | (non-idempotent interface)                |                      |
| /jobs/¡jobid¿/     | - Modify job definition. | N/A                                      | Cancel and delete the job |
| Job status information and task URIs | - Perform an operation with job. |                  |                      |
|                    | - Submit a new job | (idempotent interface).                |                      |
| /jobs/¡jobid¿/¡taskid¿/ |                | Modify task definition.                   | N/A                  |
| Task status information |                  |                                          |                      |
| /accounting/period/¡start¿-¡stop¿/ | Accounting information for the specified time span | N/A                  |                      |

Table 1. Pilot API summary

3.1. Service API

Pilot service API is outlined in table 1. User jobs are accessible through the /jobs/ collection and each job itself is a collection of job’s tasks. To create a job a user may submit a POST request to /jobs/ service URI creating the job resource and all dependent tasks resources. The job and tasks then may be modified using PUT requests. Jobs are not started automatically after their creation, they are started only after the corresponding operation request from the user, which is done through the PUT request to the job URI.

Task resources are created and deleted only implicitly based on the job definition. It is not required to define and submit all tasks to the service while creating a job. Any task definition may be modified if it has not been started yet. This allows to define tasks at virtually any moment of time, even after the job has been started. This allows the user to make corrections to job tasks without interrupting the job workflow execution.

Pilot records all events relevant to any job or task processed by the service. Jobs and tasks individual history is stored in corresponding resource documents.

Accounting information is represented as a dynamic read-only collection whose URI determines the time span requested. Besides the JSON representation, accounting information may also be returned as a CSV document if the client requests this format. The amount and scope of accounting events depends on the certificate used for request authentication. Pilot can provide full accounting information access to a limited number of users specified in the configuration file (this is useful for external accounting aggregation services). A user has access to accounting information for his own jobs, and a VO manager (recognized based on VOMS role) has access to accounting information for his virtual organization.

3.2. Implementation

Pilot service is written in Python language. It uses M2Crypto library [10] for the SSL/X.509 functions and libvomsnc libraries from the gLite project for parsing VOMS attributes[11]. SQLAlchemy object relationship manager is used for the database interface. Currently Pilot is compatible with PostgreSQL and SQLite databases. MySQL support is not available due to the lack of under second precision in date-time fields. PostgreSQL is recommended for production installs, SQLite may be used for low-load testing.

The service is split into two daemon processes (see figure 1). The first process, pilot-httpd, handles all HTTPS authentication and authorization, and processes the requests to the Pilot
service. It is implemented as a Pylons WSGI application running in a custom service container supporting X.509 proxy certificates with VOMS extensions. The pilot-httpd daemon handles most jobs and tasks operations by storing to and retrieving from the database all of the corresponding information. Globus WS-Notification [12] events are accepted by the pilot-httpd and passed to the pilot-spooler daemon. For job matchmaking requests pilot-httpd makes queries to the matchmaker service, which is available as an internal pilot-spooler RESTful HTTP service.

The second daemon, pilot-spooler, contains the following parts:

- Job processor. Analyzes the jobs stored in the database, schedules tasks for the execution.
- Task submission queue. Submits the tasks to the resources where they are executed. Current implementation supports only Globus WS-GRAM resources.
- Task status queue. Polls the status information for the running tasks which did not receive status notifications for a long time; sends status notifications based on poll results to the notifications service.
- Notifications service. Processes task status notifications coming from status queue and from grid resources via WS-Notification protocol.
- Matchmaker service. Returns lists of preferred resources based on requirement sets, virtual organization information. This service is also responsible for querying the aggregated informational system or individual resources for up to date information on the grid resources availability and properties.

For the ease of installation Pilot service and its dependencies are available as an RPM package for CentOS 5 Linux distribution.

3.3. Command Line Interface
There is a command line interface package available for the Pilot service. It provides a set of batch-like commands pilot-something, including:

- pilot-job-submit, pilot-job-status, pilot-job-info (contrary to pilot-job-status also includes information for all job tasks), pilot-job-cancel;
- pilot-task-status;
- pilot-job-matchmake (gives a list of resources compatible with a task for each task of the job and an overall estimation if the job can be executed on current grid resources);
• pilot-query-jobs (lists all jobs owned by the user).

The CLI is available as an RPM for the CentOS 5 Linux distribution and requires a minimal number of dependencies.

4. Conclusion

The workflow jobs described in this work are useful because they allow to automate multistage computations. In cases where some of the stages are independent, workflows may speed up the computations by running corresponding tasks in parallel. In this work a a simple syntax for workflow jobs with computational tasks based on JSON language was described. The grid service ‘Pilot’ for executing such jobs was developed. It was designed as a RESTful grid service. The outline of the service API was described, and some implementation details are given. The Pilot service is running in production on GridNNN project testbed resources.

References

[1] GridNNN project web site URL http://www.ngrid.ru/
[2] Foster I, Frey J, Graham S, Tuecke S, Czajkowski K, Ferguson D, Leymann F, Nally M, Storey T, Vambenepe W et al. 2004 Globus Alliance
[3] Demichev A, Kryukov A and Shamardin L 2009 Software Products and Systems 4 172–176 ISSN 0236-235X URL http://www.swsys.ru/index.php?page=article&id=2409
[4] Demichev A, Ilyin V, Kryukov A and Shamardin L 2010 Numerical Methods and Programming 11 62–65 ISSN 1726-3522 URL http://num-meth.src.csc.ru/english/journal/tom_2010/v11r206.html
[5] Yu J and Buyya R 2005 SIGMOD Rec. 34(3) 44–49 ISSN 0163-5808 URL http://doi.acm.org/10.1145/1084805.1084814
[6] Crockford D 2006 The application/json Media Type for JavaScript Object Notation (JSON) Tech. rep. IETF Network Working Group RFC4627 URL http://tools.ietf.org/html/rfc4627
[7] Zyp K 2010 A JSON Media Type for Describing the Structure and Meaning of JSON Documents Tech. rep. IETF Network Working Group draft-zyp-json-schema-02 URL http://tools.ietf.org/html/draft-zyp-json-schema-02
[8] Tuecke S, Welch V, Engert D, Pearlman L and Thompson M 2004 Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile Tech. rep. IETF Network Working Group RFC3820 URL http://tools.ietf.org/html/rfc3820
[9] Groep D 2010 The VOMS Attribute Certificate Format OGF Draft, artf6312 URL http://forge.gridforum.org/sf/go/artf6312
[10] M2Crypto — a python crypto and SSL toolkit URL http://chandlerproject.org/Projects/MetooCrypto
[11] Virtual Organization Membership Service URL http://glite.web.cern.ch/glite/
[12] Graham S, Hull D and Murray B 2006 Web Services Base Notification 1.3 (WS-BaseNotification) Tech. rep. OASIS Standard