Journal of Physics: Conference Series **513** (2014) 032078 doi:10.1088/1742-6596/513/3/032078

Task Management in the New ATLAS Production System

K De[1], D Golubkov[2], A Klimentov[3], M Potekhin[3] and A Vaniachine[4] on behalf of the ATLAS Collaboration

[1]University of Texas at Arlington, Arlington, TX, USA
[2]IHEP, Protvino, Russian Federation
[3]Brookhaven National Laboratory, Upton, NY, USA
[4]Argonne National Laboratory, Argonne, IL, USA

E-mail: potekhin@bnl.gov

Abstract. This document describes the design of the new Production System of the ATLAS experiment at the LHC [1]. The Production System is the top level workflow manager which translates physicists' needs for production level processing and analysis into actual workflows executed across over a hundred Grid sites used globally by ATLAS. As the production workload increased in volume and complexity in recent years (the ATLAS production tasks count is above one million, with each task containing hundreds or thousands of jobs) there is a need to upgrade the Production System to meet the challenging requirements of the next LHC run while minimizing the operating costs. In the new design, the main subsystems are the Database Engine for Tasks (DEFT) and the Job Execution and Definition Interface (JEDI). Based on users' requests, DEFT manages inter-dependent groups of tasks (Meta-Tasks) and generates corresponding data processing workflows. The JEDI component then dynamically translates the task definitions from DEFT into actual workload jobs executed in the PanDA Workload Management System [2]. We present the requirements, design parameters, basics of the object model and concrete solutions utilized in building the new Production System and its components.

1. Introduction

Data intensive research programs such as high energy and nuclear physics, astrophysics, earth science, and material science will generate exabytes of data in the near future. These data are often highly distributed and accessed by large international collaborations, and require a sophisticated Workload Management System (WMS) to manage its distribution and processing. One of the most successful WMS in High Energy Physics is PanDA (acronym for Production and Distributed Analysis System), used by thousands of physicists in the ATLAS experiment at the Large Hadron Collider (LHC) [1], and another good example is DIRAC (used in LHCb) [3].

PanDA delivers transparency of data access and processing in a distributed computing environment to ATLAS physicists [2], and in particular it

- provides execution environment for a wide range of experimental applications,
- automates centralized data production, distribution and processing,

- enables analysis activity of physics groups and custom workflows of individual physicists,
- provides a unified view of distributed worldwide resources,
- presents status and history of workflow through an integrated monitoring system.

The rich set of features provided, coupled with support for heterogeneous computing environments, makes PanDA a major enabling factor for HEP research in ATLAS, and additionally makes it well suited for other data intensive sciences. Functionality of PanDA is greatly enhanced through use of add-ons such as the Production System, which helps systematize the workload and its management according to tasks defined for each unit of work. In this paper we describe the principle of operation of the Production System, motivations for its evolution and the design of its next implementation.

2. Background

2.1. PanDA Workload Management System and its Components

PanDA consists of a number of subsystems, of which the most important are:

- **The PanDA server:** The central components of the server are implemented in Python and run under Apache as a web service. The server is the heart of PanDA, and one of its subcomponents is a "dispatcher" which receives requests for jobs from pilots and dispatches job payloads by taking priorities, resource allocation policy, and "retry" strategies into account. The server also contains the brokerage module, which works to prioritize and assign work on the basis of job type, priority, etc.
- **The PanDA database and information system :** A system-wide job database that records comprehensive static and dynamic information on all jobs and other crucial objects in the system. The sum of static and dynamic information is used throughout PanDA to configure and control system behavior from the region level down to the individual queue level.
- **The PanDA pilot system:** Pilot jobs are used for acquisition of processing resources. Workload jobs are assigned to successfully activated and validated pilots by the PanDA server based on brokerage criteria. This 'late binding' of workload jobs to processing slots prevents latencies and failure modes in slot acquisition from impacting the jobs, and maximizes the flexibility of job allocation to resources based on the dynamic status of processing facilities and job priorities.
- **Pilot factory:** An independent subsystem manages the delivery of pilot jobs to worker nodes via a number of schedulers ("pilot factories") serving the sites at which PanDA operates. A pilot once launched on a worker node contacts the dispatcher and receives an available job appropriate to the site.
- **Job submission interface:** A simple, python based client interface allows easy integration with diverse front ends for job submission to PanDA.
- **PanDA monitoring:** A comprehensive monitoring system supporting production and analysis operations; user analysis interfaces with personalized "My PanDA" views; detailed drill-down into job, site and data management information for problem diagnostics; usage and quota accounting; and health and performance monitoring of PanDA subsystems and the computing facilities being utilized.

In addition to the subsystems described above, the majority of the workflow handled in PanDA is orchestrated by its *Production System*, or "ProdSys" [4]. Its function and characteristics are explained in the following section.

2.2. The ATLAS Production System

The central ATLAS Production System (often referred to as ProdSys) was designed and deployed more than seven years ago to address the needs of managed production and organization of workflow for PanDA. The need for this system comes from the fact that the natural unit of workload handled by PanDA (by design) is a single payload job, which is deployed, monitored and accounted for by the PanDA services. At the same time, defining the exact nature and content of the payload, source and destination of the data pertaining to the job and various other parameters that characterize it, is outside of the scope of the core PanDA itself – to be handled by its Production System.

The ATLAS Production System therefore serves an important role as a top layer of abstraction, responsible for defining jobs for a large part of the PanDA workload in a scalable and automated manner. Jobs are defined in large collections that are termed tasks, and are formulated to fulfill "task requests". A task is defined based on a request, and it has a number of attributes, set in accordance with that request. The next step in the lifecycle of the task is its automatic translation into a number of individual job definitions, which sometimes is quite large (tens of thousands). In the existing system, individual job definitions are set based on the parent task parameters and remain static for the duration of the whole task execution [2]. While the Production System has proven itself as an important and useful tool, it has to be significantly enhanced and updated, based on motivations explained below.

3. Motivations for the Upgrade and Major Components of the New Production System

3.1. Most Important Motivations

- **Scalability of Workflow Management:** The initial design parameters contained an assumption that a relatively small number of tasks will be maintained in the system, while each task would be translated into a large number of job definitions, to be submitted for execution. With more PanDA experience, production managers have learned to use the system in more sophisticated ways, formulating the workload in a more fine-grained fashion, i.e. multiple tasks of smaller size, often logically connected to each other. This means a completely different scale of the number of tasks, which are now created at a rate that keeps growing almost exponentially. These tasks needs to be handled and monitored, and it is necessary to support related groups of tasks (workflows). The concept of the workflow or as we call it in this context, the *Meta-Task*, was absent in the original ProdSys.
- **Operator intervention and Meta-Task recovery:** Once the Meta-Task paradigm is implemented, there must be adequate opportunities for operators and production managers to direct the Meta-Task processing, be able to start certain steps before others are defined, augment a task, and recover from Meta-Task failures in an optimal way.
- **Dynamic job definition:** Dynamic job definition means that the job is defined at some point after the creation of its parent task, and taking into account operating conditions that were not known at the task inception. There are a number of advantages to be realized once there is a capability to define jobs dynamically, based on the actual resources and other conditions present once the task moves into the execution stage.
- **Ease of use:** There is currently a great amount of detail that the end user must be aware of, and carefully define in the interface, in order to formulate a valid task request. A lot of manual labor can be saved by implementing template capability, automation and ability to clone and/or augment existing tasks etc.
- **Support of analysis workflow:** Currently, the main focus of the workflow support is in the area of managed production. Analysis workflow is managed by the users in an ad hoc manner.

For brevity, this new system is broadly referred to as ProdSys2.

3.2. Major Components of the new Production System

To avoid inherent fragility of monolithic systems, we separated the core ProdSys2 into two major components functioning together:

- A database engine and an interface to support creation, management, monitoring and general logic of the workflow objects such as tasks and Meta-Tasks, which are managed collections of tasks.
- A mechanism for dynamic generation of jobs based on task attributes, and dispatching of jobs to computational resources, utilizing existing PanDA interfaces and protocols.

This design allows for clean separation of the generic workflow machinery, confined to the first component, and the PanDA-specific dynamic component orchestrating creation of payload jobs and their deployment to the computing sites. The two components are named respectively:

- Database Engine for Tasks (DEFT).
- Job Execution and Definition Interface (JEDI).

We provide more details on these components in the following sections. In summary, the 'big picture' of the new Production System is as follows: DEFT provides the front end, task monitoring and editing functionality to the user. JEDI processes the information maintained in DEFT in order to create complete task definitions for PanDA, and create job definitions in a format that PanDA understands. Finally, PanDA manages distribution and execution of individual jobs. Information regarding the state of jobs and tasks is collected in JEDI and fed back into DEFT for monitoring purposes.

4. DEFT: Database Engine for Tasks

4.1. The Meta-Task and its description

As previously noted, the concept of the Meta-Task was absent in the original system, while supporting it will be crucial for future reliable and efficient operation of the Production System. It is a group of interdependent tasks, where dependencies exist mostly in the form of data (i.e. data produced by one task being a prerequisite for another). Such would be the case with a "chain" type Meta-Task, where the data goes through a series of transformations such as event generation, simulation, digitization and finally reconstruction, with possible other steps like merging interspersed in between. This is illustrated in a simplified diagram in figure 1.

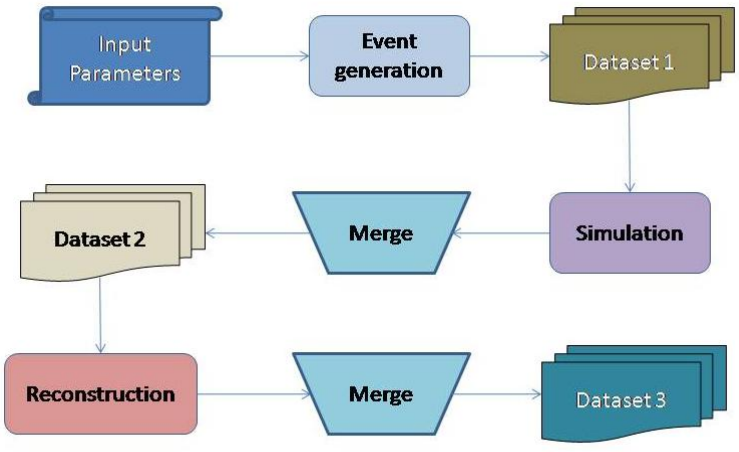

Figure 1. Example of a Meta-Task.

It is obvious that the Meta-Task can be thought of as a workflow, defined in the context of PanDA. To clarify this further, a task is a unit of *workload* which has specific data (a group of files or "datasets") as its input, and some data (datasets) as its output. Processing functionality of a task is delivered by a collection of individual *jobs* assigned to the task. The task functionality can vary significantly. In PanDA, the payload for each job (i.e. what gets done) is defined based on certain parameters of its parent task.

On the other hand, a Meta-Task is a workflow defined as a *group* of related PanDA tasks. It is natural to model it as a graph, essentially a DAG with tasks as the nodes, and edges expressing dependencies in the group of tasks – these dependencies ultimately being the data connecting the tasks. We could immediately consider "chains" of tasks, as in the above diagram, or "bags" (or "baskets"), or in fact more complex DAG cases that we may need to support in our architecture.

4.2. Principal Requirement for DEFT

- Full support of the Meta-Task model
- Implementation of the Analysis task concept
- A User Interface that represents the Meta-Task in a readable and easy to manipulate manner
- Incorporation of a workflow (Meta-Task) description language, which is human-readable, lends itself to version control and is easily parsed in software.
- Control over the Meta-Task lifecycle
- Easy to use template functionality and capability to "clone" existing tasks, and support for a library of standard templates

One common feature of the existing and the new Production Systems is that they use a RDBMS as the back end. The database is also the principal mechanism of interaction between DEFT and JEDI components of ProdSys2.

4.3. The DEFT Application

We elected to use an industry-standard XML schema, the "GraphML" [5], as a medium to present the graph description as a human-readable text, that can be readily parsed by both custom applications and a number of existing and standard tools for graph analysis and visualization. This approach allows us to create, process, revision-control and catalog DAGs with added flexibility and convenience of text files. At the application level, however, a DAG is "serialized" into two database schemas, one for the nodes and the other for its edges. This is a standard and efficient technique for storing graphs ("adjacency table" approach).

The application designed based on requirements presented above has two main components:

- "DEFT core", which is a CLI utility to import and export Meta-Tasks in GraphML data, into and from RDBMS respectively. It also allows the user to manipulate the parameters of the objects in the database and can be used to control the execution of the workflow by setting specific states to the nodes of a Meta-Task.
- "DEFT UI" which is a Web service and an interface to the database, providing monitoring and control capability via a user-friendly interface.

While the main functionality of the "DEFT core" is largely complete, the development of the UI component is under way at the time of writing.

5. Job Execution and Definition Interface – JEDI

5.1. JEDI Requirements

JEDI is a component of the PanDA server with necessary logic to dynamically define jobs based on task definitions coming from DEFT. A complete description of the requirements for JEDI is beyond the scope of this paper. JEDI's principal function is dynamic creation of jobs to be processed by the PanDA WMS, based on task descriptions (which can be more abstract since they miss the details defined by JEDI during its analysis of the current allocations, policies and status of resources), which contain parameters for a potentially large number of jobs. JEDI is required to optimally divide and direct the workload according to a variety of factors, from resource allocations to specifics of the payload job configuration to the current status of remote computing resources managed by PanDA.

In contrast to DEFT, where the user interface and user control are prominent elements of its functionality, JEDI operates automatically for the most part. Nevertheless, monitoring capabilities are essential in order to debug the system and execute proper operational support later on.

5.2. DEFT-JEDI Interaction and Operation of the Production System

Modification of entries in the database, with certain elements being used as tokens, is the principal mechanism of communication between DEFT and JEDI. One simple use case can be described as follows:

- The operator uses the DEFT UI to fetch a workflow template (stored as a Meta-Task prototype in the database) and formulate a Meta-Task, which results in new entries in the table containing tasks.
- During a periodic database sweep, JEDI picks up task definitions and processes the tasks for submission. This includes dynamic job definition for each task.
- Tasks are executed and their status (as well as the status of the data being produced) is continuously updated in the database. DEFT makes this information available via its monitoring system.
- The operator may take action with regard to a Meta-Task, such as to suspend the execution of one of its elements pending further decision. The operator can also choose to cancel and unwind the task and its associated data.

5.3. Queues and Shares

Work queues are partitions of workload in JEDI. For example, if the system is running short of jobs in a work queue, JEDI will define and submit jobs for that work queue. Tasks are mapped to work queues based on their attributes, such as task type, task owner, physics working group to which the owner belongs, and so on. The definition of work queue is configurable in the database.

Work queues have *shares* to define CPU resource allocation. The number of CPUs available for each work queue is dynamically calculated by taking into account the number of active tasks and the number of running jobs in addition to the share value.

5.4. Core Functionality

JEDI features plug-in design for its job definition engine, in order to be able to handle widely varying types of tasks. Examples include Group Production Tasks, ROOT-based analysis tasks, tasks with file-level splitting etc. An interesting feature of JEDI is generation of scout jobs for certain types of tasks, whereby a computing resource is being "probed" by running a limited number of jobs first. This allows for the resource to be validated and metrics to be collected and subsequently used in decision-making in JEDI. JEDI also includes logic for job "retry" and reassignment, in order to increase execution efficiency on the Grid.

When used at scale, PanDA handles its data in units termed "datasets", which are collections of files. In case a small number of files become corrupt or lost due to storage malfunction or some error condition in the data management system, the dataset can become essentially incomplete. JEDI contains provisions to "repair" such broken dataset by recreating the missing pieces of data, where possible.

5.5. The JEDI Application and Integration with DEFT

JEDI has been prototyped and rolled out as an "alpha" version in 2013. It was tested by creating data adaptors that allow the operators to feed tasks request from the previous version of the production system interface, into JEDI databases. At present, these distinct components of the Production System are approaching the integration phase.

6. Conclusions

We have reviewed the functionality of the ATLAS Production System based on PanDA and explained how this crucial tool is used by production managers. We observe that managing science workflows led to explosive growth of the rate and volume of tasks that must be handled in PanDA. That leads to a number of motivating factors driving the development of the new Production System. We described the basics of its design and the two main components, DEFT and JEDI, which work in tandem to provide a scalable solution for task management. DEFT provides the user interface and control over the task configuration and execution, while JEDI's principal function is dynamic job definition and optimal matching of jobs to resources.

Acknowledgements

This work was supported by the U.S. Department of Energy.

References

[1] The ATLAS Collaboration, Aad G et al. 2008 The ATLAS experiment at the CERN Large Hadron Collider, *J.Inst.* **3** S08003
[2] PanDA: Distributed production and distributed analysis system for ATLAS. T Maeno *et al*, Journal of Physics vol **119** part 6, 2008
[3] DIRAC optimized workload management. S K Paterson et al, Journal of Physics vol **119** part 6, 2008
[4] ATLAS Grid Data Processing: system evolution and scalability. D Golubkov *et al* 2012 *J. Phys.: Conf. Ser.* **396** 032049
[5] The GraphML File Format, http://graphml.graphdrawing.org/