Production System 2: Technical Design Report

ATLAS TIM
May 16th 2013

Maxim Potekhin,
BNL Physics Applications Software Group
Brookhaven National Laboratory

potekhin@bnl.gov

The Scope of this TDR

- ATLAS is using a variety of systems which depend on, interface with, or provide necessary services for PanDA
- The component known as the "Production System" has evolved over the years and in itself contains a few subsystems.
- The Technical Design Report which is the focus of this discussion deals with two specific subsystems of the new Production System:
  - Database Engine for Tasks (DEFT)
  - Job Execution and Definition Interface (JEDI)

The Document

- Filed as ATL-COM-SOFT-2013-007
- Its scope is as described above
- Has been available for review and comments since May 8th 2013

Purpose of this presentation

- To have a round of discussion based on the TDR and get feedback useful for further development

ProdSys2 components

- **DEFT**: A database engine and an interface to support creation, management, monitoring and general logic of the workflow objects such tasks, and Meta-Tasks, which are managed collections of tasks

- **JEDI**: A system for dynamic generation of jobs based on task attributes, and dispatching of jobs to computational resources, utilizing existing PanDA interfaces and protocols

- General application of component-based design

- **People**: T.Maeno is the lead developer of JEDI, and M.Potekhin is responsible for design and implementation of DEFT. It is anticipated that some additional manpower will be assigned to this project.

.

Highlights from TDR…

ProdSys2: recap of motivations

- **Scalability of Workflow Management:** Need to actively manage Meta-Tasks (collections of tasks)
- **Operator intervention and Meta-Task recovery:** Support a comprehensive set of features to enable full control of the workflow by operators, and provide capabilities to recover from common failure modes automatically. Related to monitoring, mentioned in Wolfgang's talk.
- **Dynamic job definition:** There are advantages that can 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. Also see the talk by Wolfgang earlier today.
- **Maintainability**
- **Ease of use:** implies an easy-to-use human-readable format for task definition and versioning, and a GUI to facilitate control
- **Support of analysis workflow**

DEFT

Note that due to relative autonomy of DEFT, i.e. its agnostic approach to the computational resources used (which are actively managed by the JEDI component), it is appropriate to describe it as a state machine, in which the states of the workflow components undergo a series of transitions based on certain predefined rules and external events, such completion of a PanDA job managed by JEDI, or establishment of data transfer for a dataset. By itself, DEFT is not a workload management or resource provisioning system. It is a higher level abstraction level that allows PanDA users, production managers and experts to concentrate on the design, monitoring and effective management of their workflows.

It must be noted that DEFT is a service, with appropriate APIs for users as well for external systems. JEDI is a prime example of an external system that will need to interact with DEFT.

Task and Meta-Task description and Persistence Mechanism

As per TDR, we use a XML-based format (a standard schema design in industry) to describe the graph model of the Meta-Task and its components. Addresses the requirement for task copying and task template.

A simple and proven approach is used to “flatten” graph for persistence in RDBMS, which is a set of tables for the nodes and for the edges.

As a bonus, standard tools and packages can (and are already) used for parsing, serialization, editing and visualization of the workflow.

Version control becomes trivial as Meta-Tasks templates can be put in SVN.

Complete transparency is achieved between the XML description and data in RDBMS, i.e. it’s easy to read/write/persist the Meta-Task definitions.

JEDI

JEDI is a component of the PanDA server with necessary logic to dynamically define jobs based on task definitions coming from DEFT. Main goals of JEDI development are to make the PanDA system task-aware and to have a coherent design of DEFT and JEDI databases (which maintain records of PanDA tasks and individual jobs) in an optimal manner. The following sections highlight main concepts and functionality of JEDI.

Some Use Cases (JEDI perspective)

- **Open Dataset Handling:** Some tasks could be submitted before input datasets are frozen. The issue here is that new files may be appended to an open dataset after the task is submitted. If the task is defined to run on all files including newly appended files, JEDI periodically check the dataset contents. When new files are found, JEDI defines jobs. Once the dataset is closed and all files are processed, the task is completed.
- **Lost File Recovery:** In this example, DDM consistency service would provide a list of lost files and it would be decided to reproduce those files. In this case, new task is submitted rather than changing status of the old task which produced the files. The task parameter specifies names of lost files. Then JEDI finds how those files were produced and defines new jobs to reproduce the files. Rucio will provide API to repair files which allows changing metadata for files in closed datasets. If the input files for the lost files are also lost (e.g., the dataset has already been deleted), another task which re-generate those input files is submitted using the DEFT-JEDI communication table and those tasks are chained.

Some Use Cases (DEFT perspective)

- **Manual Control, Suspension, Holding and Termination:** there must be a way to prevent individual tasks from entering the active state until the human operator decides otherwise (for any reason, including analysis of failure modes, debugging or a change of resource allocation).This can be achieved by including "armed/disarmed" states in the task state enumerator. Important cases pertaining to control over Meta-Task include suspension and termination. Suspension can be achieved by setting the state of the next inactive node to “pending”, and “disabled” for termination.
- **Submission of a Partially Defined Meta-Task**
- **Meta-Task Augmentation (Incremental Execution):** Once a task is has completed, the Physics Coordinator might want to decide that a larger statistics or additional data samples are needed in a particular step. From the DEFT perspective, this can be resolved by Meta-Task Augmentation, whereby edges are added to the graph, and the states of adjacent nodes are appropriately updated.
- **Meta-Task Recovery:** recover as much data as possible in case jobs re failing.

Discussion?