The ATLAS PanDA Monitoring System and its Evolution

A Klimentov, P Nevski, M Potekhin[1], T Wenaus
Brookhaven National Laboratory, Upton, NY11973, USA

[1] Contact person

E-mail: potekhin@bnl.gov

Abstract. The PanDA (**P**roduction **an**d **D**istributed **A**nalysis) Workload Management System is used for ATLAS distributed production and analysis worldwide. The needs of ATLAS global computing imposed challenging requirements on the design of PanDA in areas such as scalability, robustness, automation, diagnostics, and usability for both production shifters and analysis users. Through a system-wide job database, the PanDA monitor provides a comprehensive and coherent view of the system and job execution, from high level summaries to detailed drill-down job diagnostics. It is (like the rest of PanDA) an Apache-based Python application backed by Oracle. The presentation layer is HTML code generated on the fly in the Python application which is also responsible for managing database queries. However, this approach is lacking in user interface flexibility, simplicity of communication with external systems, and ease of maintenance. A decision was therefore made to migrate the PanDA monitor server to Django Web Application Framework and apply JSON/AJAX technology in the browser front end. This allows us to greatly reduce the amount of application code, separate data preparation from presentation, leverage open source for tools such as authentication and authorization mechanisms, and provide a richer and more dynamic user experience. We describe our approach, design and initial experience with the migration process.

1. Introduction

PanDA is a Workload Management System built around the concept of Pilot Frameworks [1]. In this approach, workload is assigned to successfully activated and validated Pilot Jobs, which are lightweight processes that probe the environment and act as a ‘smart wrapper’ for the payload. This 'late binding' of workload jobs to processing slots prevents latencies and failures in slot acquisition from impacting the jobs, and maximizes the flexibility of job allocation to globally distributed and heterogeneous resources [2]. The system was developed in response to computing demands of the ATLAS collaboration at the LHC, and for a number of years now has been the backbone of its data production and user analysis.

A central and crucial component of the PanDA architecture is its database, which at any given time reflects the state of both pilot and payload jobs, as well as stores a variety of vital configuration information. It is driven by the *PanDA Server*, which is implemented as an Apache-based Python application and performs a variety of brokerage and workload management tasks, as well as advanced pre-emptive data placement at sites, as required by the workload.

By accessing the central database, another PanDA component – the *PanDA Monitoring System* (or simply *PanDA Monitor*) – offers to its users and operators a comprehensive and coherent view of the system and job execution, from high level summaries to detailed drill-down job diagnostics. Some of the entities that are modeled in the system are:

- Pilot Job Generators (schedulers)
- Queues (abstractions of physical sites)
- Pilot Jobs (or simply "pilots")
- Payload Jobs (the actual workload)

All of these are related to each other by reference, i.e. an instance of the Pilot Job Generator is submitting pilots to a particular queue, a queue in turn can contain any number of pilots in various stages of execution, and payload jobs are mapped to successfully activated, running pilots. Payload jobs are also associated with their input/output datasets (details of data handling in PanDA go beyond the scope of this paper). The purpose of the PanDA Monitoring System is to present these objects and their logical associations to the user with optimal affordance.

The original implementation of the PanDA Monitor aimed to be a simple and lightweight Python application embedded in the Apache server. The mechanism of data delivery to the consumer was defined almost exclusively as plain HTML, accessible with any browser. Database access was done via the *cx_Oracle* library, and the logic of HTML generation did not employ templates or other techniques that guarantee complete separation of "business logic" from presentation layer. While flexibility of this approach allowed for rapid development of the application driven by user requirements, and eventual success of the PanDA Monitor as a crucial component of the overall system, it became clear with time that due to size and complexity of the code accumulated over the years, it has become difficult to further enhance and maintain. In addition, a need for better integration with LHC-wide systems was realized, necessitating a departure from HTML as the data interchange format. This combination of factors provided the motivation for PanDA Monitor upgrade.

2. Choice of Data Interchange Format and its Design Implications

In order to implement a truly modular and client-agnostic data service, thus facilitating system integration, we chose JSON as the data interchange format. Motivations behind this particular choice, in addition to it being a popular standard, were: available robust implementations of AJAX techniques and JSON parsing in Javascript and Python, high readability and existing wide use of JSON in ATLAS.

Obviously, in cases when the client is a Web browser, rendering of data represented as JSON needs to be done programmatically. That means that the "center of gravity" of the code shifts dramatically in this approach – the server side becomes rather thin, performing mainly database queries and serialization of data, while the client becomes "rich", with function which include user interaction, construction of the DOM tree reflecting data content, managing AJAX communication, caching, etc.

3. The Server component of the PanDA Monitor

In the years since the PanDA Monitor was originally developed, quite a few of the Web Application Frameworks have reached maturity and widespread usage, such as Ruby on Rails, Pylons, Django etc. These products bring to the table a number of advantages over a plain Python application, such as Object-Relational Mapping (ORM); robust templating mechanisms; user authentication and authorization and support of user roles (note that these features have a varying degree of support in different frameworks). Applications Frameworks also make use of best security practices and tools, Given the existing expertise of ATLAS developers in Django Framework, we chose it as the basis for the server code of the PanDA Monitor. In this approach, the code of Django libraries as well as the application code is embedded into a Python capable Web server, such as Apache.

Recent versions of Django (such as 1.2.1 used at the time of this writing) have the following important features:

- Native handling of multiple databases
- Support of aggregate database functions (e.g. "count")
- Transparent caching of data with a variety of available back-end storage options

Django allows for accessing a variety of RDBM systems, such as MySQL, Oracle, PostgreSQL etc. Currently PanDA is utilizing Oracle as its RDBM (MySQL was used in earlier implementations [3]).

A few of the database tables in the PanDA system contain a very large number of entries, such as the tables recording the status of pilots and jobs. Without taking special measures, queries against such tables can result in (a) undesired extra load on the database server, resulting in diminished overall performance of the server, and (b) unacceptable latency in the client performing the query. To mitigate this problem, one has to use optimization techniques. Some of these include Oracle-specific tools like *bind variables* and *hints*. The former effectively results in pre-compiled query residing on the server, thus saving its resources, while the latter aims to make use of database indexes more efficient.

Currently available versions of Django do not have such functionality in their Oracle back-end implementation. The solution is, therefore, to identify the queries which need such optimization, and segregate the code in a way where it falls back on *cx_Oracle* library which allows for plain SQL to be used and is thus free of ORM limitations. Fortunately, this is a small fraction of the overall code and in itself does not represent a significant development overhead.

As mentioned earlier, in the evolved server, the bulk of the server functionality consists of serializing of data obtained from the database via ORM into JSON. Thus, in some cases the complete code for a function serving a particular client request may be reduced to one line, if using helpers such as serializers, resulting in examples like this one:

```
return HttpResponse(serializeArray(Pilotqueue.objects.filter(submithost=host).iterator()))
```

In certain cases the optimization of queries still does not yield optimal results in terms of responsiveness of the client and load of reduction in the server. It is important, therefore, to make provisions for caching data obtained from those queries where the validity period of the data allows for meaningful caching. Fortunately, Django has a comprehensive caching functionality that handles the validity period of the data transparently for the user, thus resulting in compact and easy-to-maintain code. We took additional measures to provide configurability to caching, whereby it's done automatically for those methods listed in a special dictionary along with corresponding validity times, thus allowing changes without touching the core application code. Properly encoded URLs are used as keys for the cache, and data is stored in JSON format, allowing for simple retrieval and delivery as needed. Testing has been done with a variety of back-end storage options, including *memcached*.
While detailed performance metrics aren't available at this point, the benefits of caching are already obvious in the near-instant delivery of cached data to the browser, which is a marked performance increase.

4. The Web Browser client component of the PanDA Monitor

By design, the server of the PanDA Monitor can interact with a variety of clients because of the use of JSON as the sole data interchange format. Of course, from the point of view of system usability, the most important case is the Web Browser client, which must handle user interaction and render Web pages based on JSON data, as opposed to handling static HTML as was the case in the original version of the Monitor. To implement such functionality, we chose Javascript as the base platform. In addition, we decided to employ the *jQuery* and *jQuery-UI* Javascript libraries, in order to leverage robust DOM-tree navigation facilities, easy-to-use AJAX functionality and interactive widgets that these libraries provide (see more at http://jquery.com). One immediate issue that one faces in an AJAX Web application is the need to maintain proper URLs for pages rendered by Javascript code (also referred to as "hash"), for bookmarking and history purposes. In our implementation of application-specific logic code to support this, we use the *jQuery BBQ* plug-in.

The user interaction is facilitated by widgets coming with the *jQuery-UI* Javascript library, including Tabs, Accordion, Buttons and others.

5. Modular structure of the Server and Client

The server code consists of reusable components common across various functionality areas, and of methods producing data pertaining to one or more major objects in the system, such as Pilots, Jobs etc. This allows for a natural and logical way to organize the server application code into distinct "sections" that support their respective functionalities. This organization of server code is effectively mirrored on the client side. This provides ample opportunities for team development and Staged Delivery Plan.

6. Conclusions

The ATLAS PanDA Workload Management System incorporates a comprehensive and feature-rich Monitor that aggregates and presents a variety of information characterizing the state of the system to the end-users and operators. In order to reap the benefits of emerging Web Application Platform technologies, enhance system interoperability and afford a more dynamic experience to the user, we are working on migration of the system to a combination of a Django-based server, which delivers data to its clients in JSON format, and a rich AJAX client making use of *jQuery* Javascript library. A working prototype is being deployed for beta-testing. The structure of the application makes it amenable to staged delivery.

7. References

[1] T Maeno 2008 PanDA: distributed production and distributed analysis system for ATLAS *J. Phys.: Conf. Ser.* **119** 062036

[2] P Nilsson 2008 Experience from a pilot based system for ATLAS *J. Phys.: Conf. Ser.* **119** 062038

[3] Graeme Andrew Stewart *et al* Migration of ATLAS PanDA to CERN *J. Phys.: Conf. Ser.* **219** 062028