Overview of Workload Management System

Panda is built on the concept of Pilot Job Framework. Workload is designed to successfully activated and validated Pilot Jobs, which are lightweight processes which probe the environment and act as a "smart wrapper" for the payload. This Tute 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 globally distributed resources. 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.

By accessing this database, another Panda component - the Monitoring System - 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.

Characteristics of the currently deployed Panda Monitoring System

Panda Monitoring System is a Web application that performs complex tasks of database queries and data aggregation, in order to provide the end user with a variety of interconnected views of the state of the system and its configuration. The state of the system includes the status of pilot jobs, payload jobs, data that's being transmitted in support of payload execution, and other parameters. All of this information is reflected in the database, which in its current implementation relies on Oracle RDBM. As with most Panda components, Python is the main language platform, with cx_Oracle libraries employed in the DB access layer. Where necessary, such as in case of "expensive" queries, data is cached in local file system using custom logic. Presentation layer consists of Python code generating HTML on the fly, by simple string manipulation (no templates).

Motivations For System Evolution

Lack of sufficient separation of query logic and presentation layer in the existing implementation reduces flexibility in user interface, simplicity of communication with external systems, and makes code modification and maintenance less optimal. Since the creation of Panda Monitor a few of Web Application Frameworks have achieved maturity, and offer industry-standard security, administration, templates and other standard functionality that gives us an opportunity to significantly reduce the amount of application code while simultaneously improving its robustness. Simultaneously, the issue of interoperability with other systems can be resolved by establishing a common flexible data exchange for the Monitoring System such as based on JSON (instead of serving HTML), and delegate the presentation layer functions to client software (e.g., "rich client").

Evolution of the ATLAS Panda Monitoring System

The ATLAS working group has reached a consensus on general architecture: Panda Monitoring is seen as a combination of client-agnostic web service delivering data via HTTPS and AJAX capable rich client for the User Interface. This provides considerable flexibility in interfacing systems external to Panda, and allows for a customizable, richer and more dynamic user experience.

Choice of Technologies

- Server side: given the choice of server technologies, preference was given to a proven platform already used by a few groups in ATLAS so as to reuse existing expertise and open possibilities for sharing code and practices (Django Web Application Framework).
- Client: our intent to create a more dynamic user experience and better responsiveness in the system by using AJAX technology stipulates heavy use of JavaScript in the web user interface. We chose the powerful, industry-standard javascript libraries jQuery and jQuery UI, which also already were in the arsenal of ATLAS developers.

Architecture of the evolved ATLAS Panda Monitoring System

In the new design, JSON messages are the only API for serving data to the client. Because of that, the bulk of the Django server code does not have any page-rendering functionality and the amount of code is significantly reduced. At the same time, the amount of client code becomes substantial as it performs majority of page-building tasks in addition to some caching logic. We use jQuery to navigate the DOM tree and provide AJAX functionality. We rely on jQuery UI Library and its widgets in order to afford a more dynamic and interactive experience to the user.

Data caching is given special consideration. Due to extensive amount of data in some of the database tables, certain queries may take a long time and put an undesirable load on the DB server if performed frequently. In the evolved Panda Monitoring System, we take advantage of transparent and robust Django caching functionality in conjunction with highly scalable memcached network service. Caching policy is fine-grained and flexible, i.e. caching is mandated for specific queries and cache lifetime is specified on-per-query basis, in the server configuration file (no application code needs to be changed to make changes to caching policy).

For optimal performance, proper optimization techniques must be used when sending resource-intensive queries to the server (Oracle RDBM). A typical example includes simultaneous use of "hints" and "bind variables". Being RDBM specific, such combination does not have adequate support in Django ORM. In such cases, we fall back to using "native" cx_Oracle libraries and serialize results in the common JSON format.