Pandjango: review of PanDA Monitoring migration, testing and staged delivery plan

ATLAS Software and Computing Week Monitoring Workshop

November 30th, 2010

Maxim Potekhin for BNL Physics Applications Software Group
Brookhaven National Laboratory

potekhin@bnl.gov

Overview

Quick review:

- Technology review (one last time!)
- Project history

Pandjango:

- Architecture of Pandjango application
- Caching
- Code organization

Testing and delivery:

- Staged Delivery Plan
- Effort profile
- Current status and testing
- Pandjango vs other monitoring systems
- Conclusions

Technology Review: Intro

With Pandjango project being 6 months old, it's a good time to take a step back and look at the technology choices (were they correct?), and chart the path forward.

Most of popular Web application frameworks all appeared around 2005, and weren't mature or widespread enough to be used in PanDA. Currently deployed Monitor is a Web application (*Apache+mod_python*) with the following characteristics:

- Data is being served to the Web browser client in HTML, which is the primary data interchange format. There is little to no support for other types of clients
- Construction of HTML code is not cleanly separated from business logic, making code reuse and refactoring difficult

Technology Review: Motivation

We observe emerging needs for, and focus on:

- systems integration, such as feeding job status data to systems external to PanDA, e.g. LHC dashboard(s) and other monitors
- systems evolution, such as possible adoption of novel high-performance database solutions for demanding tasks, e.g. PanDA job archival
- possibility of customization, and more flexibility in the UI – we will have a better UI now, and/or we can have a completely new generation of the Web presentation layer later on, if necessary, and never touch the server code

It was determined that PanDA Monitor needs to evolve.

Technology Review: Solution

The solution is to create a client-agnostic server that does not include any elements of the presentation layer (such as HTML generation etc).

For such an implementation, a "natural" choice of data interchange format would be either XML or JSON, which would be rendered into Web pages inside the browser client, as needed. This dovetails with using AJAX technology.

That will also make the system future-proof with regard to novel database solutions, should they be implemented.

Technology Review: Frameworks

Regarding server technology, we observe that:

- 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, including but not limited to:
  - Object-Relational Mapping (ORM)
  - Sophisticated templating mechanisms
  - Flexible and powerful mapping of URLs to methods
  - User authentication and authorization
  - Robust account administration and support of user roles
  - Use of best security practices and tools

Technology Review: Django

- Given the existing expertise of ATLAS developers in Django Framework, we chose it as the basis for the server code of the evolved PanDA Monitor, codenamed Pandjango (PanDA+Django)

- Recent versions of Django (such as 1.2.1 currently used in the development effort) have the following important features:
  - Native handling of multiple databases with variety of back-ends
  - Support of aggregate database functions (e.g. “count”)
  - Transparent caching of data with a variety of available back-end storage options

Technology Review: Oracle (1)

A few of database tables in the PanDA system contain a very large number of entries, such as the tables recording the status of jobs. Without taking special measures, queries against such tables can result in:

1. undesired extra load on the database server, resulting in diminished overall performance of the server
2. 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.

* *Caching will be discussed separately*

Technology Review: Oracle (2)

Currently available versions of Django do not have the functionality to support both hints and bind variables 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 (in terms of number of distinct queries being managed) and in itself does not represent a significant development overhead – but still something to be aware of!

Technology Review: Client (1)

As an immediate consequence of removing the presentation layer from the server we get

- lean server
- relatively "fat" (or "rich") client

This setup is in fact more conducive to team development than a monolithic Python application because part of the functionality is implemented in a set of separate code units (see one of the following slides).

We have evaluated , and rejected the Google Web Toolkit as a platform for this project due to configuration issues we experienced in 2009 as well as a very steep learning curve (which we can ill afford). We decided to use a much simpler industry-standard tool, a combination of Javascript libraries *jQuery* and *jQuery-UI*. The former allows to convenient navigation, construction and modification of the DOM-tree in the browser, while the latter

Technology Review: Client (2)

With use of *jQuery* we get (for free)

- Asynchronous and feature-rich AJAX data retrieval
- Automatic parsing of either JSON or XML
- A host of convenient helper functions (like array iterator)
- Convenient navigation, construction and modification of the DOM-tree

With use of AJAX we lose correlation between the content of the address bar in the browser and the data being accessed, i.e. we no longer have valid URLs for bookmarking, history and "back" and "forward" functionality.

- This needs to be reconstructed in the client code, with help of a state-keeping plug-in (like BBQ) and application-specific logic. Example: a URL maps to the state of the application which has a particular tab open, and a particular row of a table highlighted

Technology Review: Client (3)

The use of *jQuery-UI* facilitates the development of highly interactive and dynamic UI, with a good choice of configurable widgets. There is a vibrant community effort to produce fancier or better widgets (often mutually exclusive). Things to look out for:

- When tempted to use nested widgets or fancy behavior, it's easy to write obfuscated code
- Certain widgets or their combinations incur a performance penalty (latency) which sometimes is not easy to attribute in a working application

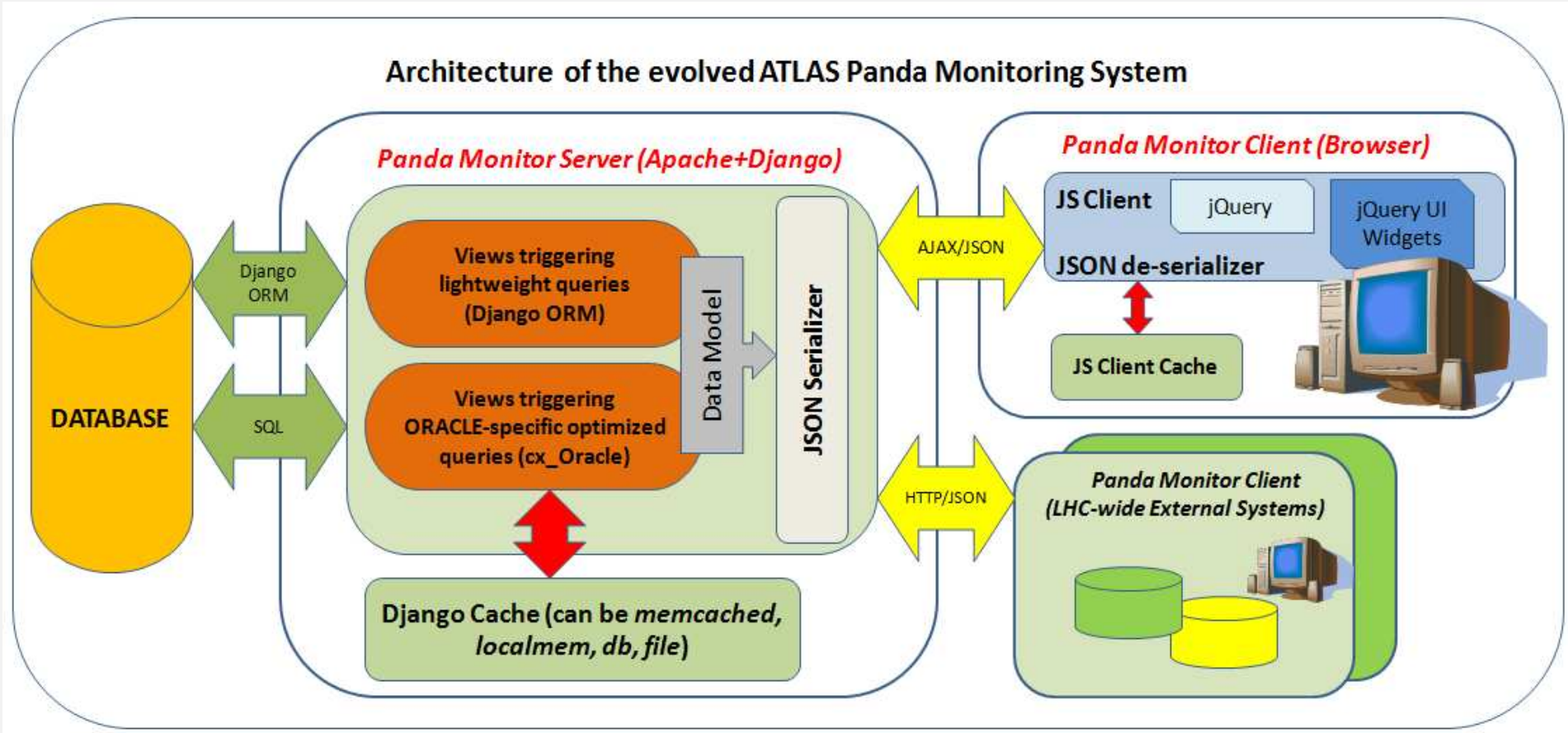
Project history pre-2010

- Discussions in ADC and choice of JSON and AJAX as underpinnings of new monitor architecture
- Evaluation of GWT (Google Web Toolkit). Configuration and compatibility issues discovered, as well as steep learning curve
- Evaluation of Django and its deficiencies (such as lack of multi-DB support – since resolved!)

Project history in 2010

- Concept presented at the Monitoring Workshop at BNL in late March
- A simple technical demonstration was ready in mid-April, and consensus was reached on technology choices (Django+JSON/AJAX/jQuery)
- A working prototype of Autopilot section of Panda Monitor was ready in July
- Major code refactoring and implementation of Oracle-optimized queries done in August
- Django-native caching implemented and tested with a variety of back-end storage options (like *memcached*) in September
- November: prototype ready for beta-testing

Pandjango Architecture



Caching (1)

It is mandatory for certain queries, and desirable for others, to have the results cached, with optimal lifetime, to alleviate the load on the server and provide optimal response time. Caching is possible on both server and client side.

- On the client side, a degree of caching happens naturally when data structures are populated prior to page being rendered (need to control lifetime).
- There are third party modules that "borrow" memory from the browser itself to create an automatic cache system totally transparent to Javascript application. Currently, we consider these non-essential.

Caching (2)

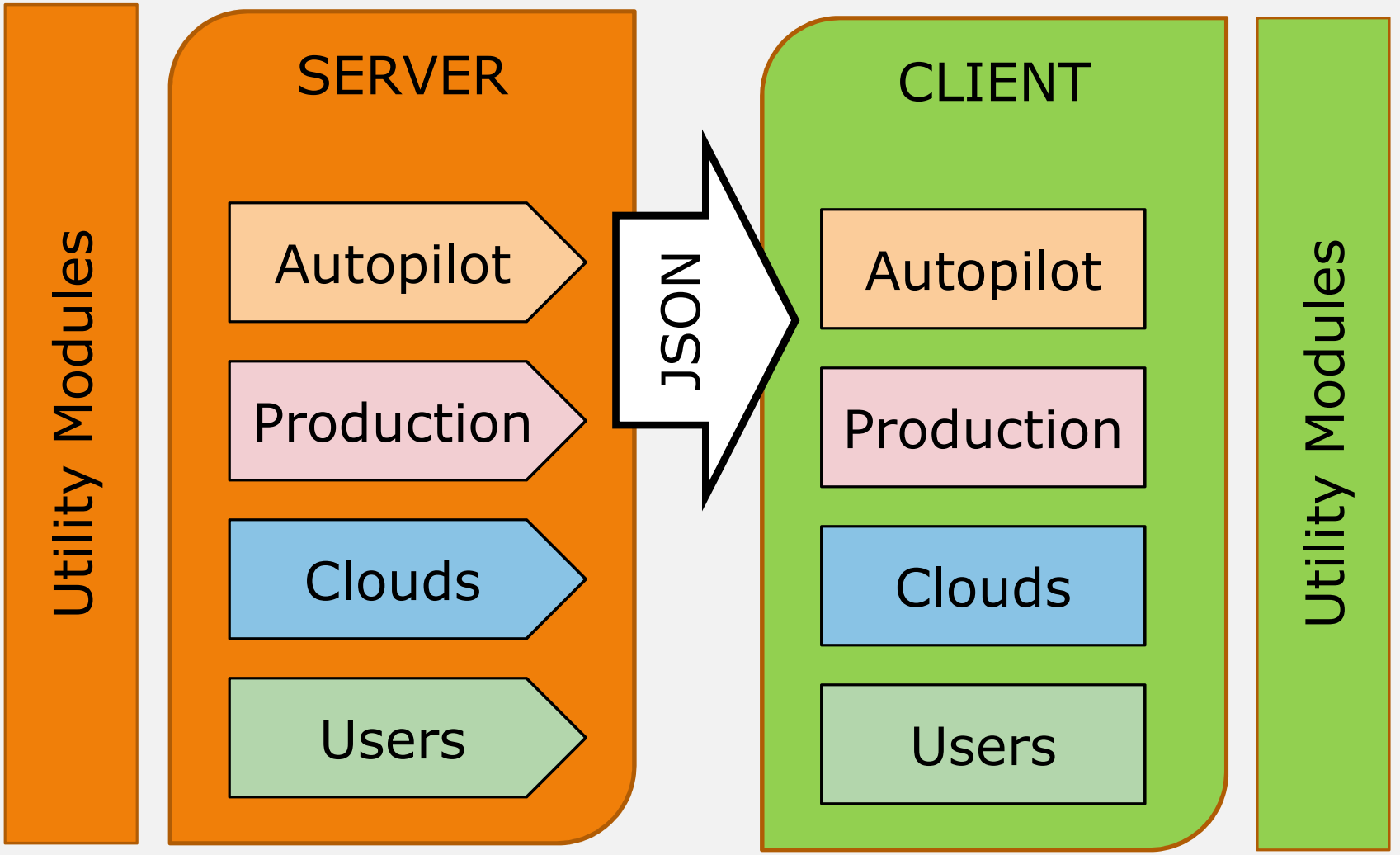
- On the server side, Django provides robust and transparent built-in caching with a variety of back-ends: file, *memcached*, in-memory and DB. The first two were tested in Pandjango. We prefer to have *memcached* deployed already at that stage, for beta testing (we need host(s)!)
- urlencoded query values (URLs) are used as keys across all types of cache
- In Pandjango, caching points and lifetimes are defined in a configuration file, which allows for quick modification without touching the core code
- We also have a file-based "developer's cache" which is more transparent to developers due to obvious location an naming of data files, which allow for more selective refresh and easy inspection – great aid in development!

Code organization (1)

The code is organized in “sections”, or “silos”, each corresponding to a link (a tab) on top of the existing Panda Monitor page. Pandjango prototype is currently limited to a working “Autopilot” section, and a partial one for “Production” and stubs for the rest.

- On the server side, each section is represented by a code unit with corresponding name, e.g. *autopilot.py, production.py* etc, making code navigation trivial. There are a few service modules shared across the application
- Likewise, the code for the browser client is organized into directories named *autopilot, production* etc, being a mirror of the server code tree. There are also a few utility modules shared across the application

Code organization (2)



Code organization (3)

- We have a simple and natural way of keeping the code amenable to team development
- Granularity of code units and their (relative) independence from each other facilitate implementation of the **Staged Delivery Plan**
- For further flexibility in delivery, it is possible to define which sections are accessible to beta-testers or end-users, by simple changes in the configuration file, e.g.

Staged Delivery Plan (1)

- Staged Delivery is a process where the target feature set is delivered to the users over a period of time in pre-defined steps
- It allows for more effective beta-testing and increases time value of the project
- In case of Pandjango, the delivery steps naturally map onto "sections" of the Panda Monitor
- As mentioned earlier, each section is easily switched on/off, which is great for staged delivery
- Q: When we start to phase in new software, how do we provide features that are yet to be implemented, in the staged delivery scenario?
- A: By properly integrating the existing Monitor and forwarding appropriate requests to it (next 2 slides)

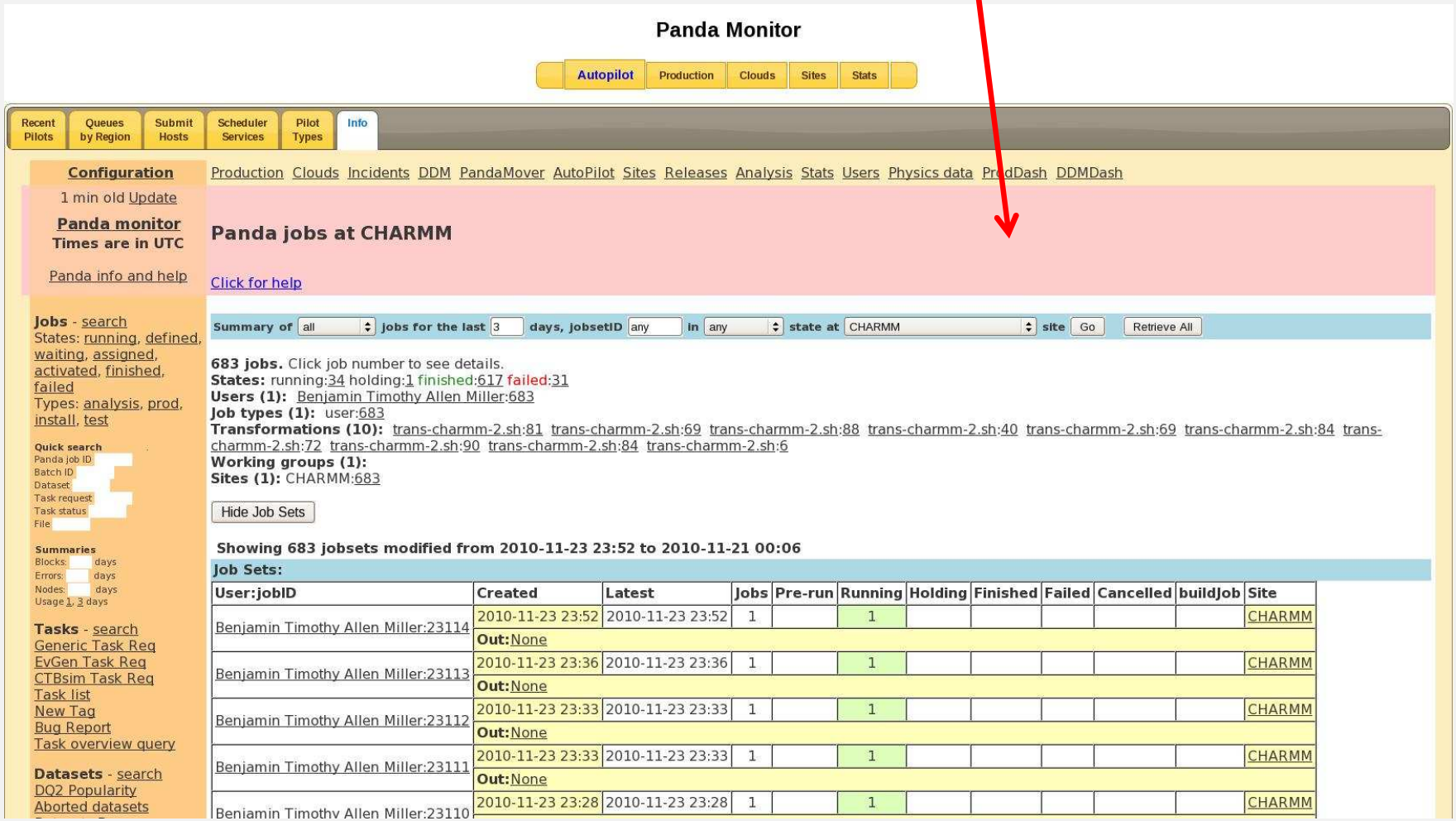
Staged Delivery Plan (2)

Integration of the UI across the new and the old Monitor: certain requests from Autopilot section are forwarded…

ID	Type	Site	Queue	Tstart	Tstate	Status	State	Err	Err Info	Trun
tp_gridui12_13295_20101121-210010_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-21	2010-11-21	done	finished	0	Dispatcher has no jobs	
tp_gridui12_28399_20101121-090009_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-21	2010-11-21	done	finished	0	Dispatcher has no jobs	
tp_gridui12_31454_20101124-000010_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-24	2010-11-24	done	finished	0	Dispatcher has no jobs	
tp_gridui12_2876_20101122-120009_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-22	2010-11-22	done failed	failed	3000	Job failed: Non-zero failed job ret...	
tp_gridui12_1830_20101123-230016_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-23	2010-11-23	done	finished	0	Dispatcher has no jobs	
tp_gridui12_3379_20101122-210016_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-22	2010-11-22	done	finished	0	Dispatcher has no jobs	
tp_gridui12_30075_20101123-161012_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-23	2010-11-23	done	finished	0	Dispatcher has no jobs	
tp_gridui12_2106_20101123-210010_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-23	2010-11-23	done	finished	0	Dispatcher has no jobs	
tp_gridui12_25471_20101120-210009_1	atlasOfficial2	AGLT2_Install	AGLT2-condor	2010-11-20	2010-11-20	done	finished	0	Job successfully completed	

Staged Delivery Plan (3)

…and HTML obtained from the "old" Monitor rendered in the same UI framework using jQuery



Effort Profile (1)

The amount of effort that needs to be put into Pandjango depends on the scope of the deliverable. We can define the target feature set either as

- Short List: functionality similar to that of the PanDA monitor, on a better platform, or
- Wish List: greater degree of integration across the various "dashboards" – production dashboard, global ADC job monitoring etc.

Regardless of which option is chosen, we must strive to stick with a "single data source" paradigm, i.e. core database tables should be only accessed from one server application. This will allow to better manage "hard" queries and take full advantage of caching capacity in the new system (Pandjango where applicable).

- need to have a registry of queries done by apps?

Effort Profile (2)

PanDA Monitor has 14 sections, our of which 3 are in the data movement domain hence belong to the “wish list”. The amount of work that go into each sections will vary greatly. We observe that at the current development stage most of the effort does NOT go into

- UI/client/jQuery design
- Server design and optimization

…because these item have been already worked on. Instead, it is consumed by reverse engineering of the existing code, which is motivated by our desire to keep backward compatibility in terms of general logic of data presentation, cross-links in the data and affordance of the UI. That makes it a hard project, and harder to scope the effort.

Effort Profile (3)

Pandjango effort profile in 2010 consisted of ~0.8 FTE. Pandjango is not a good project for summer students' involvement because

- Unfortunately, summer is still far in the future 
- Being productive in navigating PanDA database requires a lot of foot work and will present a learning curve that's a bit too long for the summer cycle

With that assumption, we can make a rough ETA estimate:

- At the current level, 60% of the short list can be completed in 6 months
- Given +1 FTE, all of the short list can be done in 6 months – that is a preferable solution because it would free up manpower currently tied in maintenance of the existing application

Effort Profile (4)

What about the "Wish List", i.e. better monitoring systems integration across the board? We need a few things to make it happen

- Executive sponsorship, i.e. a group of people or a person who have authority to direct evolution of every component
- A person (or a small group) to oversee the creation of a single look-and-feel UI
- Finalized ADC plotting service (prototype exists)
- Consolidation of data access across applications (such as through Pandjango server)

Are we up for it?

Current status and testing (1)

There is a functional section of Pandjango which corresponds to the "Autopilot" tab in the original application (will be presented a video demonstration). As discussed earlier, we need to test the application characteristics such as affordance of the UI, performance of the server and its cache and remove the remaining bugs. The plan is

- To select a group of beta-testers
- Deploy a Pandjango instance at CERN (which needs an instance of *memcached*)
- Deliver code updates with a short turnaround time, and taking direction for the development of other sections from the test results and feedback

Current status and testing (2)

We plan to use the machine lxbuild002 (located at CERN), which is currently being rebuilt. Access to the server ports from outside CERN will need to be done via SSH tunnels.

Memcached deployment: T.B.D. Would really prefer a semi-dedicated box for development and testing effort.

Plotting service: T.B.D.

Fallback plan: to use the existing development server out of BNL (advantage: open ports visible globally).

Pandjango vs other monitoring systems

There are monitoring systems, such as the Global ADC Dashboard, which will benefit from data feeds coming from PanDA. What is the relationship between PanDA Monitor proper, and these other monitors?

- We consider these as complimentary. PanDA keeps and furnishes information that is specific to its Workload Management logic, which is beyond the scope of other information systems and is characterized by larger number of links among objects being modeled. PanDA Monitor is a major control and debugging tool for the whole system, and we'll need it in this role going forward.
- The new Monitor will serve data to external consumers in a highly portable format (JSON), providing single point of access, caching and other optimization.

Conclusions

- With regards to plans previously made, we are pretty much on target
- We are satisfied with the choice of technologies for this project
- Adding modest manpower to the project appears optimal
- There is potential to better integrate various existing monitoring components under one umbrella, but this needs to be managed