Building a Reliable High Performance PanDA Facility

J. Hover[1], Y. Smirnov[1], D. Yu[1], X. Zhao[1], T. Wlodek[1], J. DeStefano[1], J. Smith[1], M. Karasawa[1], T. Maeno[1], T. Wenaus[1], A. Thor[3], J. Caballero[1], M. Potekhin[1], K. De[3], M. Sosebee[3], P. Nilsson[3]

[1] Brookhaven National Laboratory, PO BOX 5000 Upton, NY 11973, USA
[2] CERN, CH-1211 Geneva 23, Switzerland
[3] University of Texas, 701 S. Nedderman Drive Arlington, TX 76019, USA

E-mail: jhover@bnl.gov

Abstract. PanDA, ATLAS Production and Distributed Analysis framework, has been identified as one of the most important services provided by the ATLAS Tier 1 facility at Brookhaven National Laboratory (BNL), and enhanced to what is now a 24x7x365 production system. During this period, PanDA has remained under active development for additional functionalities and bug fixes, and processing requirements have increased geometrically, leading to challenges in service provision. We used a RedHat Satellite system, cfEngine, and custom scripting to streamline the deployment, provisioning, and maintenance of the OS, Grid Middleware, and PanDA. We deployed redundant hardware and multiple service instances for each critical Panda component, and added a high performance/high availability capability by introducing a Layer4/7 smart switch from F5 in front of some components. This cost-effective approach greatly improves throughput and reliability, and prevents any single point of failure caused by hardware, network, grid middleware, operating system, or local PanDA application issues. Its transparency allows flexible management of the heterogeneous service, with only minimal application-level configuration and coding necessary to support integration with the smart switch. We have also implemented an extensive monitoring and alert system using Ganglia, Nagios (with extensive custom probes), RT (Request Tracker), and a custom-written ticket opening/escalation system. These tools work together to alert us to problems as they occur, and greatly assist in quickly troubleshooting any failures. In Summary, our contributions in innovation hardware resilience, extensive monitoring and automatic problem report and tracking significantly enhance the reliability of the evolving Panda system while allowing Panda developers ready access to the system for software improvement. Our experiment shows that the Panda performance was more than triple that of the legacy Panda instance, and any single failure was transparent to ATLAS users.

1. Introduction

PanDA (Production and Distributed Analysis) is a pilot-based, data-driven workload management system developed by U.S. ATLAS. It has been handling U.S. ATLAS production since 2005 and has now been adopted ATLAS-wide for managed production.[1]

As a Tier 1 for ATLAS, Brookhaven National Lab (BNL) and the RHIC/ATLAS Computing Facility (RACF) have had the responsibility to pursue three goals simulaneously in our deployment of PanDA. First, maximize the *performance* of the PanDA system by leveraging the ways in which the PanDA architecture is scalable. Second, maximize the *reliability* of

the system by using good-quality hardware, building in as much redundancy as the PanDA architecture allows, and closely monitoring the services for any problems. And third, allow for ongoing active development of the PanDA software by the Physics Application Software (PAS) group.

2. PanDA Components and Architecture

PanDA consists of several loosely-coupled subsystems. Below we will briefly describe what they are and how they work. For each component, we will discuss how the RACF deployed the component to maximize reliability, performance, and flexibility.

2.1. F5 Switch

A crucial part of RACF's PanDA deployment is not, strictly speaking, a PanDA component. BNL/RACF, as part of its wider networking architecture strategy, purchased 2 F5 Big-IP 3600 smart switches [2]. These switches provide a way to introduce redundancy, load balancing, and fault tolerance at both layer 4 (IP) and layer 7 (application; HTTP/S) in a way transparent to network clients.

The simplest application of the F5's capabilities is to create a virtual IP for a stateless service. The F5 will then rewrite and forward packets to two or more servers providing that service. The decision about which server to forward to can be configured to be as simple as round-robin, or based a complex metric based on system load, number of connections, etc. Server-based metrics can be provided by a F5 plugin that runs on the server.

A sophisticated application of the F5's capabilities would be at layer 7. A relevant example to PanDA would be for a server, behind the F5, which is overloaded to respond to an HTTP query with a **503 Service Unavailable** response. The F5 would recieve the response, recognize that it indicates that the server is overloaded, and re-send the initial request to a different back-end server.

Specific ways in which we have used our F5s to improve PanDA reliability and performance will be discussed below within the section for the relevant component.

2.2. PanDA Autopilot System

As a pilot-based system, PanDA requires some way to get the initial PanDA pilot onto worker nodes at sites and executing. Since *how* the pilot arrives is independent of its function within the system, PanDA actually supports a number of different mechanisms (Condor-G, local batch submission via Condor-C, LSF, or PBS, Condor Glide-ins, and the EGEE WMS). More could be imagined.

For Grid-based usage within the U.S. cloud, BNL runs the PanDA Autopilot. This component is a set of long-running python scripts (typically one per Grid site batch queue), triggered via cron, which use Condor-G [3] to continuously keep the Globus gatekeepers at each site filled with a set number of waiting pilot jobs.

While Condor-G is quite scalable, it is stateful. So each Autopilot server is a single point of failure. But by running multiple Autopilot servers we at least avoid the possibility of a single server failure halting all U.S. production. Additionally, each Autopilot server is configured similarly, so adjusting which system serves which queues is done fairly easily, i.e. an Autopilot instance . This is commonly done to load-balance the service based on ongoing observation, and as part of the emergency response to trouble with a particular server.

As part of our efforts with Condor-G, BNL and the PanDA developers have been working extensively with the Condor-G developer(s) to provide additional features and configuration within Condor-G to help it handle pilot submission better. For example, pilot jobs are all the same, and don't really require the kind of careful tracking, holding, and re-submission

that Condor normally provides. Another example is enhancing the output of Globus-specific information from the *condor_status* command.

2.3. PanDA Pilot Code Download service

In the PanDA job lifecycle, small pilots are submitted to sites via Condor-G (or other methods). These small pilots do not contain all the functionality needed to request a payload job from the Panda Server. First, the complete set of pilot code is downloaded via HTTP from a central location. This code (maintained by the PanDA pilot developer(s)) resides in a Subversion repository .

Initially, code was retrieved directly from Subversion via HTTP. As production scale increased, Apache/Subversion was unable to maintain pace. The reason for this is that Subversion is essentially a database, and each web query triggers a complex lookup to assemble the response.

To address this issue, we established a separate Apache web server in which we configured a memory-based web proxy (Squid). The purpose of the cache is to reduce the request load on the back-end Subversion server. Now, only occasional queries trigger a full lookup on the back-end Subversion system, and most external queries are pulled from memory on the front-end web server.

The benefits of this system include a high-performance code download service combined with code updates still being immediately available as soon as they are committed to source code control.

2.4. PanDA Server

The PanDA server is a programmatic web service component. It runs on Apache, using mod_python, interacting with the back-end database running on separate servers. It uses the Apache worker model, with many independent processes handling client requests in parallel. Since all state is maintained in the central database, the PanDA Server application instance itself is stateless.

This component's primary purpose is to dispatch jobs to pilots that request them. It also accepts job submissions from the end-user PanDA client tool (pathena).

As a stateless database application that interacts with clients via HTTP/S, this component was a perfect candidate for integration with the F5 infrastructure at BNL. We created a virtual IP and related hostname (pandasrv.usatlas.bnl.gov) and configured three separate systems with the PanDA Server running on them. This arrangement allowed the pilot software to be simplified, by simply pointing at the hostname of the virtual IP rather than each pilot tied to specific back-end server.

2.5. PanDA Monitor

The PanDA Monitor is a web-based dashboard-style graphical application. It runs on Apache, using mod_python, and interacts with the back-end database for persistence. The Monitor is the primary way that users and administrators get a view into the status of current and past jobs, data movement, Autopilots, and even the configuration and staus of the PanDA Servers and Monitors themselves.

This component is also (largely) stateless, and therefore was integrated at BNL/RACF with the F5 switch under another hostname (pandamon.usatlas.bnl.gov).

To aid troubleshooting, the PanDA team added a label at the bottom of the pages showing which back-end server had actually handled the request.

2.6. Panda Logging Service

The PanDA logging service is a mechanism by which distributed PanDA components (both servers and pilots) can centrally log events for display and tracking. It is also an Apache-based web service using mod_python.

It is typically deployed along with the PanDA Monitor and runs within the same Apache process. It is stateful, and so is not amenable to virtual IP scalability schemes. But it is low-impact and simple, and has been run on a single Server/Monitor host without significant issues.

2.7. PanDA databases

The database back-end is one the most important components of PanDA system at BNL. It consists of 6 production MySQL servers and 2 development ones which support about 10 different databases.

Detailed description of Panda Database architecture, structure and performance can be found in [4].

3. PanDA Hardware

The PanDA Autopilot component, servicing the needs of the U.S. cloud, was deployed on three (3) rack-mounted 1U quad-core Xeon Dell servers (2x Xeon E5430, 18GB RAM; 1x Xeon 5160, 9GB RAM). Normal system load is 1.0-3.0, except when we deliberately place extra work on one system to push the scalability of Condor-G.

Front-end production PanDA services (Server, Monitor, Logger) were deployed on three (3) rack-mounted 1U quad-core Xeon Dell servers with 16GB RAM. Typical system load is 1.0-2.0. There are three primary PanDA systems, which ran the PanDA Server and PanDA Monitor components. One of these systems also ran the PanDA Logger component.

The back-end PanDA database service was provided on the RACF's production MySQL cluster. It consists of:

(i) 2 Dell PowerEdge 2950 severs with 4-core Intel Xeon(R) CPU 5150 2.66GHz , 16 GB memory using 64-bit RHEL4 OS and the hardware raid-10 over 6 15krpm 145GB SAS disk drives

(ii) 4 Penguin Computing Relion 2600SA machines with 8-core INTEL XEON CPU E5335 2.00Ghz, 16GB memory using 64-bit RHEL4 OS and six 750GB SATA disk drive with software RAID-10 implementation.

4. Software Infrastructure and Grid Middleware Dependencies

The base OS with the system middleware dependencies, and a user account for PanDA, were installed and provided to the PanDA team. The PanDA team then performed final installation and configuration of the PanDA services within the PanDA account home directory.

All PanDA components were deployed on Red Hat Enterprise Linux 4 (32-bit for middleware compatibility).

4.1. Base system dependencies

These are the core OS and packages provided by Red Hat or 3rd-party public Red Hat repositories for the core server systems (Server, Monitor, Logger). These include: Apache 2.0.x, Python 2.5, mod_ssl, mod_python, memcached, python-MySQLdb. For the Autopilot system, a Condor 7.x installation is also required.

4.2. System Grid Middleware dependencies
These are packages provided by EGEE and/or the HEP and Grid communities. They include: Glite 3.1 User Interface, gridsite, mod_gridsite.

4.3. PanDA install dependencies
These are packages directly installed within the Panda account home directory. They include: DQ2 (Don Quixote 2, the ATLAS data subscription application), matplotlib, Graphtool.

The benefits of this overall arrangement are:

(i) The PanDA team can configure PanDA to use the system-installed versions of the base software (e.g. Apache, Python, mod_python), which means that it is always updated to the most recent security patches.

(ii) The PanDA developers can upgrade and/or reconfigure the PanDA service whenever desired developers to troubleshoot and debug any application-level issues which arise.

(iii) The PanDA service runs as a non-privileged account, which contributes to system security.

5. PanDA Operations

5.1. System maintenance
All RACF production server OS installations are managed by way of a Red Hat Satellite server. This is a system which serves as a package update repository and system management utility.

5.2. Monitoring
The current and recent operational health of the PanDA service itself can be monitored interactively via the web interface of the Panda Monitor component. There is a dedicated Autopilot Monitor page which allows shifters to see if queues are full. There are also dedicated pages to view job status, both overall and on a site-by-site basis.

In addition, we use a number of automated monitoring mechanisms to detect problems with any components at BNL, and alert RACF and PanDA shifters to emerging problems. We implemented several probes to integrate PanDA into our facility Nagios system. Nagios in turn links PanDA into the BNL Request Tracker (RT) ticketing system. This ensures that any trouble with any of the server components is brought to the experts' attention promptly.

Examples of these probes include:

(i) Panda Server isAlive probe: The PanDA Server component has a callable function accessed through the HTTP/S interface which tests the server status and returns a basic HTTP 200 "OK" text response. Our Nagios probe simply calls that URL and monitors the output.

(ii) Panda Monitor page response: This probe merely requests the primary front page of the Panda Monitor web application and looks for characteristic text that indicates it is functional.

(iii) VOMS proxy status probe: This probe runs on the PanDA core and Autopilot servers to confirm that the VOMS proxy for the user account under which PanDA runs exists and is valid. This proxy is automatically renewed by a cron script on the hosts.

(iv) System memory usage: This probe runs on the PanDA servers and reports problems with memory usage.

(v) System disk usage. This probe runs on the PanDA servers and reports problems with disk usage. The PanDA system includes cleanup operations that are meant to keep directories with temporary files and logs clean (based on file time). This probe confirms that those mechanisms are functional.

These probes are in addition to a set of standard probes that run on all infrastructural hosts at BNL/RACF. If these probes detect a problem, they send a Nagios alert by email to RACF staff and PanDA shifters and other experts. Additionally, these alerts are taken in by a separate issue tracking and escalation SLA (Service Level Agreement) system.

6. Problems and Challenges

6.1. External Service Dependency Issues

In initial configurations, PanDA was run from an account whose home directory resided in an NFS-mounted partition. Likewise, some software used within PanDA resided in AFS-mounted filesystems. In both these cases, occasional network or service interruptions, slowness, or instability resulted in degraded functionality or failure of the PanDA services. In order to avoid such problems, PanDA was ultimately run from a local home directory on a system without NFS or AFS even installed.

6.2. Panda Server and Database Performance Issues

We have run the PanDA system at BNL in production mode since late 2005. In terms of number of connections, queries and user requests the load both on the Panda servers and on the Database back-end is quite high. For example, on Panda Archive/Meta DB servers it can reach 800 parallel threads and up to 2800 queries per second. Some other interesting details describing the database performance can be found in [2]. So taking this database access pattern into account a number of non-optimized user queries can be running quite a long time on the server. In order to avoid such behaviour we developed code which helps us to identify an intrusive query, and the particular users or applications which initiate the query. Then we can provide feedback to the user and help modify the badly-written query to get better performance and response time. This approach has significantly reduced the amount of non-optimized slow queries running on the database.

Nevertheless, there are still occasional queries which run longer than is acceptable, so we developed a special cron job which can detect this, creates the corresponding log record, periodically sends e-mail notifications to experts and kills these problematic queries. Sometimes the corresponding PanDA server processes or requests can't handle the MySQL disconnections quickly and can freeze. To resolve this at the moment Panda server process has to be restarted either manually or automatically by Nagios. In the future perhaps Panda server code can be improved by adding some error-handling to take this into account.

6.3. Condor-G/Globus Scalability Issues

In the course of running U.S. production with Autopilot, we began to encounter scaling issues with Condor/Condor-G when reaching 5000 jobs from a single submit host. We brought these issues to the attention of the Condor development team and they have enthusiastically assisted us in characterizing these issues and put significant effort into adding features and fixes into Condor-G to address them.

For the most part, the issues with Condor-G have derived from two implicit assumptions within Condor in the past. First, that all jobs are unique, and precious, and that failed jobs must be returned to the queue and tried again. Second, there has been an assumption that jobs have a fairly long duration, measured in at least tens of minutes if not hours.

PanDA's usage of Condor/Condor-G revealed that these assumptions are not correct. For the first assumption, this is a benefit, because it means that jobs need not be carefully tracked or cleanup up after. The Condor team added a config flag, **Nonessential**, which indicates to Condor-G that it need not be retried if it fails.

For the second assumption, it just meant that some internal throttles and data handling polling had to be re-thought so that Condor could cope with jobs that run for only a few minutes, and the larger jobs-per-site throughput that such short jobs cause.

6.4. OpenSSL/X509 Host Certificate Issues with Virtual IPs

Any service that is accessed via HTTPS has an SSL host certificate used to secure the connection between the server and the client. When introducing a smart switch with such services, there is the question of whether to simply forward connections or to proxy them.

If you simply forward the connection, there will always be a mismatch between the (virtual) hostname/IP the client is connecting to and the real hostname/IP of the server that is connected. This can be dealt with in two ways.

One can use a single host certificate in the name of the virtual host on all back-end servers. This is simple, but precludes clients from ever making direct connections to the back-end servers. For several reasons this is undesireable.

Or, one can specify a SubjectAltName in the host certificates for the back-end servers. This alternate name is that of the virtual hostname through which clients are connecting. The problem we encountered is that our Certificate Authority (DOEGrids) does not currently support SubjectAltNames in certificates.

The alternative to all this is off-loading SSL to the smart switch, and making *unsecured* connections (via HTTP) to the back-end servers. This also precludes (safe) client connections to back-end servers, and requires some minor adjustments to web application code to access the SSL variables.

7. Conclusions

Maximum reliability is achieved by a combination of two approaches.

First, reduce or remove single points of failure. For applications, this can be achieved by taking stateless front-end servers and presenting them transparently via a single IP. It can be achieved with databases by using clustering technology.

Second, implement thorough monitoring so that when (not if) an unavoidable single point failure occurs the service can be restored rapidly by either bringing a warm spare online or shifting the damaged function to another node.

Maximum performance is achieved by leveraging implicit application scalability via load balancing, either statically (by assigning different portions of the total tasks to different servers) or dynamically (by using redirection at some layer).

Maximum flexibility is achieved by striving for uniformity in underlying servers such that functions can be moved/migrated when needed, by maintaining as much modularity as possible so that changes to one subsystem don't affect others, and by allowing users as much autonomy as possible.

Pursuing all these goals simultaneously requires a willingness to find the suitable solution for each particular instance. Not all services will be amenable to, or even require, the same level of optimization and redundancy. And, in some cases, it may be necessary to get deeply involved with the developers of some external software upon which you depend in order to improve its ability to serve your needs.

References

[1] PanDA project URL: https://twiki.cern.ch/twiki/bin/view/Atlas/Panda
[2] F5 Networks URL: http://www.f5.com/
[3] Condor project URL: http://www.cs.wisc.edu/condor/
[4] Yuri Smirnov, T. Wlodek et al. Experience with ATLAS MySQL PanDA Database service – to be published in 2009 J.Phys.Conf.Ser. (CHEP'09 Proceedings).