Next Generation PanDA Pilot for ATLAS and Other Experiments

P Nilsson[1*], F Barreiro Megino[2], J Caballero Bejar[3], K De[1], J Hover[3], P Love[4], T Maeno[3], R Medrano Llamas[2], R Walker[5], T Wenaus[3]
on behalf of the ATLAS Collaboration
[1]University of Texas at Arlington (US), [2]CERN, [3]Brookhaven National Laboratory (US), [4]Lancaster University (UK), [5]Ludwig-Maximilians-Univ. Muenchen (DE)

E-mail: Paul.Nilsson@cern.ch

Abstract. The Production and Distributed Analysis system (PanDA) has been in use in the ATLAS Experiment since 2005. It uses a sophisticated pilot system to execute submitted jobs on the worker nodes. While originally designed for ATLAS, the PanDA Pilot has recently been refactored to facilitate use outside of ATLAS. Experiments are now handled as plug-ins such that a new PanDA Pilot user only has to implement a set of prototyped methods in the plug-in classes, and provide a script that configures and runs the experiment specific payload. We will give an overview of the Next Generation PanDA Pilot system and will present major features and recent improvements including live user payload debugging, data access via the Federated XRootD system, stage-out to alternative storage elements, support for the new ATLAS DDM system (Rucio), and an improved integration with glExec, as well as a description of the experiment specific plug-in classes. The performance of the pilot system in processing LHC data on the OSG, LCG and Nordugrid infrastructures used by ATLAS will also be presented. We will describe plans for future development on the time scale of the next few years.

1. Introduction

A common approach in grid computing is to use pilot jobs. In the case of ATLAS [1], pilot factories are used to send special lightweight jobs, called pilot wrappers, to the batch systems that execute them on the worker nodes. The pilot wrappers download the PanDA Pilot [2] and launch it using pilot options that are relevant to the site in question. The responsibility of the pilot is to download the actual payload from the PanDA Server and any input file from the local Storage Element (SE), execute the payload, upload the output to the SE, and send the final job status to the server.

PanDA [3], *Production and Distributed Analysis*, is a workload management system that has been used by the ATLAS Experiment since 2005. The PanDA system has been very successful in managing the distributed analysis and production requirements across all ATLAS grids; OSG [4], EGI [5] and Nordugrid [6]. Today PanDA is being considered for use beyond ATLAS by several other

* To whom any correspondence should be addressed.

experiments. To meet this need, it has been necessary to refactor the PanDA Pilot which until recently has been ATLAS specific.

2. General structure of the PanDA Pilot

The PanDA Pilot is written in the Python language and has a modular structure. The main code is divided into two modules, pilot and monitor, to facilitate the introduction of glExec [7], which offers user identity switching. While it would have been easier to do any user identity switching before the pilot is launched by the wrapper, it was not an alternative for ATLAS because of the highly useful multi-job mechanism that allows a single pilot to download and run multiple payloads from different users in sequential order until it uses up a predetermined time allocation. The pilot contains the multi-job loop, and starts the monitor after having successfully downloaded a job (defined as a payload and all the information needed to execute it). The monitor forks and monitors a subprocess module, which is in charge of serving the payload. Stage-in and stage-out are handled by Site Mover modules. A number of utility type modules exist that serve the main modules, including server communication, TCP messaging, error code handling and interpretation, job definition, as well as other utilities. More details on the main pilot modules and special mechanisms are given in the sections below.

3. Plug-in mechanism

A new PanDA Pilot user should in principle only have to implement certain methods (marked as "compulsory" in the code) in the plug-in *Experiment and *SiteInformation classes. The plug-in classes contain methods that are experiment specific but sorted into two different classes. The *Experiment classes contain methods related to payload setup, how the subprocess (see below) should be launched, how metadata should be handled, which files should be removed from the payload work directory before the job log file is created, etc. The *SiteInformation classes contain methods for handling site information from a DB, how it should be downloaded, from where, and how to verify its integrity, as well as how to manipulate it (e.g. parameter overwrites which can be useful for special testing), etc.

When the pilot wrapper launches the PanDA Pilot, it gives it a set of options that defines the initial state of the pilot. The pilot option "-F " is used to select a certain experiment. Internally, the value is used to generate the proper Experiment and SiteInformation objects via factories (ExperimentFactory and SiteInformationFactory) based on the Factory Design Pattern. More details about the *Experiment and *SiteInformation classes can be found in the sections below.

4. Workflow of the PanDA Pilot

This section outlines the workflow of the PanDA Pilot and provides information about when methods that are compulsory to implement are called. Several compulsory methods are already implemented in the base classes, which might or might not only be relevant for ATLAS. If any detail needs to be changed in these methods, the user simply makes a copy of the default method in the base class and add it to the derived class and make the necessary changes there. Major steps, indicating the need for experiment specific code modifications, are described below.

Upon launch, the PanDA Pilot starts with downloading PanDA site information, unless it is already downloaded by e.g. the wrapper (or by hand, which can be useful during interactive testing). A secure curl command is used to download the information from the PanDA server. In case experiment specific changes are needed, the corresponding methods for handling the site information can be overridden in the relevant subclass (*SiteInformation). A description of all site information DB fields used by the PanDA system can be found in ref [8].

The pilot then performs a set of special checks and displays information that can be unique to an experiment. For ATLAS, the pilot tests the import of the LFC module, verifies CVMFS via an external script and displays the CVMFS ChangeLog, as well as system architecture information. Any test failure can be used to abort the pilot.

A signal handler is declared that will handle and act on incoming kill signals. If such a signal is

received, the pilot will immediately inform the PanDA server and then proceed to kill the remaining subprocesses (if any). It is important that the SIGKILL signal does not arrive too soon after the initial kill signal, since it can take up to a few minutes for the pilot to kill all subprocesses and perform the clean up. The time difference between SIGKILL and other kill signals can be fine tuned in case the kill signal originates from the batch system. Approximately four minutes of time difference is an absolute minimum time limit, but a longer time is recommended.
If the site, where the pilot is running, has opted for job recovery, an algorithm will be run that looks for output files deliberately left behind by previous pilots that encountered problems during stage-out. The stage-out of these files will be re-attempted for a limited number of tries.

5. The Multi-Job loop

The main loop of the PanDA Pilot is the so-called multi-job loop. As mentioned in Section 2, this refers to the capability of running multiple jobs, from different users, sequentially until the pilot runs out of time. This is a site configurable feature controlled via the timefloor field in the PanDA site information DB. If unset, or set to zero, the pilot will only execute one single payload. If it is set to a value greater than zero, the multi-job loop will download and execute payloads until it runs out of time. The multi-job feature is of particular use when there are many short jobs are in the system (the pilot launch overhead is bypassed).
The multi-job loop begins with the creation of the current work directory. A job is downloaded from the PanDA server (job dispatcher). As an alternative, the job can also be downloaded in advance or be created by hand and be placed in a file, which will be found and used by the pilot. If the dispatcher cannot find a job, the pilot waits one minute and then tries again. If a job still cannot be found, the pilot aborts. When asking for a job, the pilot sends worker node details (e.g. CPU and memory information) along with other parameters (e.g. pilot user DN in case he or she only wants a job corresponding to the matching DN) to the job dispatcher which decides if it should return a job to the pilot or not. A successfully downloaded job definition dictionary is stored in a file for later use.

6. The monitoring loop

The PanDA pilot is equipped with glExec (currently in development). Because of the multi-job loop, the pilot wrapper cannot perform any user identity switching before the pilot is launched which would otherwise be the normal approach. To solve this problem, the inner part of the multi-job loop (henceforth referred to as the monitoring loop) is placed in a separate module called the monitor. The glExec user identity switching is done before the monitor is executed. The monitor module is used either directly by the pilot, in the case glExec is not used, or after the glExec user identity switching from inside the glExec interface.
The main method of the monitor module begins with creating a TCP server run as a separate thread. The TCP server will listen to messages sent from a subprocess, which is responsible for the payload. The pilot performs local checks, including a verification of the pilot user proxy and whether there is enough disk space left for running the job. The remaining time of the VOMS proxy extension must be over 48 hours, and the local disk must have at least 5 GB free space left.
The pilot extracts and verifies the existence of the software directory, if defined in the site information. Alternatively, it can get the software directory from an environmental variable.
The pilot proceeds with forking the subprocess. The parent process corresponds to the monitoring loop, while the child process can be any of the subprocesses described in the Subprocess module section below. The default subprocess is called runJob, which performs payload setup, stage-in of input files, payload execution and stage-out of output files. Additional subprocess modules are currently in development. The subprocess selection has been made experiment specific in case further subprocess modules need to be developed.
Every ten minutes, the main monitoring loop checks the remaining disk space, the number of running processes, that the entire pilot has not run out of time on the worker node, and that the payload is not looping (stuck). A 60 second sleep marks the end of a monitoring loop iteration. When a job has

ended, its work directory is tarred up and is staged out by the main pilot process. Any lingering orphan subprocesses (e.g. from a failed payload) will be identified and killed. As a last step, the payload work directory is deleted and the monitor returns to the multi-job loop in the pilot module.

7. Subprocess modules

The monitor module forks and monitors a subprocess that is responsible for the payload. This subprocess can in principle be any module that needs the full attention and supervision of the monitor. The primary PanDA Pilot subprocess module is called runJob, briefly mentioned in the Monitoring loop section above. Two additional subprocess modules are currently in development; runEvent will be used to read and process events from an Event Server, and runHPCJob (preliminary name) will be used for handling HPC jobs.

7.1. The runJob module

This section describes the main steps of the primary subprocess module, runJob.

Payload setup: Before the payload can be executed, the pilot must know how exactly it should be set up. This is handled by an experiment specific method that essentially prepares a string with the proper setup scripts, including the source commands, followed by the name of payload script and its arguments, or the corresponding wrapper script that in turn runs the actual payload. Any setup scripts should be verified that they exist and work by this method. The entire setup string will be executed in its own shell using commands.getstatusoutput().

Stage-in of input files: The pilot receives information (dataset names, local file names and GUIDs) about which input files should be staged in, if any, from the PanDA Server/Job Dispatcher when the job is downloaded. ROOT files in user analysis jobs can be opened remotely (a.k.a. remote I/O or direct access mode) in which case stage-in is not performed. The pilot still needs to give the full path to the input files in order for the payload wrapper script (runGen or runAthena in the case of ATLAS) to be able to read the file remotely, which means that the pilot has to translate SURL based file paths to TURLs. This is done using information from the PanDA site information DB or by using the lcg-getturls command.

For normal stage-in, the pilot needs to know 1) which copy tool to use, and 2) how to lookup the file information in a file catalog. The copy tool to be used is defined by the copytool or copytoolin fields in the site information DB. Two fields are needed since stage-in and stage-out can use different copy tools. Common copy tools include: lcg-cp and xrdcp. Otherwise, both fields need to be set. For a given copytool value, a Site Mover factory is used to return a corresponding Site Mover object. E.g. for copytool = "xrdcp", the Site Mover factory returns an instance of the xrdcpSiteMover class. Each copytool supported by the pilot has a corresponding class, inheriting from the parent SiteMover class containing common code. The pilot currently supports LFC lookup and for Rucio [9].

Stage-out of output files: The pilot needs to know how and where exactly it should transfer the output files to. The copy tool to be used for stage-out is defined by the copytool field (copytoolin is never used for stage-out) and the pilot selects the corresponding Site Mover accordingly.

The site mover uses experiment specific methods (belonging to the *SiteInformation class) for generating the stage-out paths based on information from several PanDA site information DB fields (se, seopt, se[prod]path, setokens). For Rucio style file path generation, it is enough to add /rucio at the end of se[prod]path. When the pilot encounters this substring, it will automatically use Rucio style paths. In that case, the file path will depend on dataset name, local file name and scope.

8. Recent new features of the PanDA Pilot

The user can request live payload debugging when he or she launches the grid jobs using pathena or prun [10]. When the pilot downloads a corresponding job, it will receive a special instruction in the job definition to frequently upload debugging information about the payload (essentially the last several stdout lines). This information will then be made available on the PanDA Monitor job page for

viewing by the user. Normally the pilot sends a heartbeat message to the server every thirty minutes, but in the case of user debugging, the frequency is increased to five minutes.
The PanDA Pilot has several retry mechanisms. In the case stage-in fails due to a temporary SE related problem, the pilot will re-attempt the stage-in a second time after a few minutes. If that fails as well, the pilot has the option to attempt stage-in from a remote SE using the Federated ATLAS XRootD (FAX) [11] system. The FAX system consists of several dozens of sites accessed by hundreds of clients that act like a single storage resource. A special FAX Site Mover was developed for the PanDA Pilot, which means that the pilot can also use it as a primary copy tool, and not only as a fail-over mechanism which makes it potentially interesting for "diskless" sites.
The pilot is equipped with a mechanism for stage-out to an alternative SE. The idea is that if the pilot fails (partially or completely) to stage-out the output files at the primary SE (Tier-2 in ATLAS), it will download new site information from the DB for an alternative/secondary SE in the same cloud (Tier-1 in ATLAS) and re-attempt the stage-out there.

9. Plans for future developments

The PanDA Pilot has been refactored to facilitate the development of experiment specific classes. The Common Analysis Framework collaboration between ATLAS and CMS [12], has resulted in the development of experiment classes for CMS. Furthermore, AMS [13] is planning to adopt the PanDA system. Improving and further developing the PanDA Pilot for serving multiple experiments is of highest priority. To this end, several projects are foreseen including a new version of the job recovery mechanism [2], providing a full PanDA Pilot documentation, improving error reporting in a multi-experiment environment, continue glExec integration, pilot support for Event Server jobs and HPC's.

10. Performance of the pilot system

The PanDA system is serving over 100k production jobs and 35k user analysis jobs concurrently. It is performing very well with high job efficiency. The error rate in the entire system is very low. For production jobs the majority of the errors are site or system related, while for user analysis jobs the most common issues are related to application software. Pilot mechanisms like job recovery and FAX failover contribute to the robustness against site related failures.

11. References

[1] The ATLAS Experiment: *ATLAS Technical Proposal*. ATLAS Collaboration. CERN/LHCC/94-43, 1994
[2] P. Nilsson et al, Proc. of the 19th Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP2012)
[3] T. Maeno et al, Proc. of the 18th Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2010)
[4] Open Science Grid: http://www.opensciencegrid.org
[5] European Grid Initiative: http://www.egi.eu
[6] M. Ellert et al., NIM A, 2003, Vol. 502
[7] D. Groep, O. Koeroo, G. Venekamp, J.Phys.:Conf.Series 119 (2008) 062032
[8] https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/SchedconfigParameterDefinitions
[9] http://rucio.cern.ch
[10] https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/DAonPanda
[11] R. Gardner et al., Proc. of the 19th Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2012)
[12] CMS Collaboration, *JINST* **3** (2008) S08004, doi:10.1088/1748-0221/3/08/S08004
[13] R. Battiston, Nucl. Instrum. Methods Phys. Res., Sect. A 588, 227 (2008)