The next generation PanDA Pilot for and beyond the ATLAS experiment

Paul Nilsson[1,*], *Alexey* Anisenkov[2,3], *Doug* Benjamin[4], *Daniel* Drizhuk[5], *Wen* Guan[6], *Mario* Lassnig[7], *Danila* Oleynik[8,9], *Pavlo* Svirin[1], and *Tobias* Wegner[7] on behalf of the ATLAS Collaboration

[1] Brookhaven National Laboratory, Physics Department, United States
[2] Budker Institute of Nuclear Physics, Russia
[3] Novosibirsk State University, Russia
[4] Argonne National Laboratory, United States
[5] National Research Centre Kurchatov Institute, Russia
[6] University of Wisconsin-Madison, Department of Physics, United States
[7] CERN, European Laboratory for Particle Physics, Switzerland
[8] University of Texas at Arlington, Department of Physics, United States
[9] Joint Institute for Nuclear Research, Russia

Abstract. The Production and Distributed Analysis system (PanDA) is a pilot-based workload management system that was originally designed for the ATLAS Experiment at the LHC and to use with grid sites. Since the coming LHC data taking runs will require more resources than grid computing alone can provide, the various LHC experiments are engaged in an ambitious program to extend the computing model to include opportunistically used resources such as High Performance Computers (HPCs), clouds and volunteer computers. To this end, PanDA is being extended beyond grids and ATLAS to be used on the new types of resources as well as by other experiments. A new key component is being developed, the next generation PanDA Pilot (Pilot 2). Pilot 2 is a complete rewrite of the original PanDA Pilot which has been used in the ATLAS Experiment for over a decade. The new Pilot architecture follows a component-based approach which improves system flexibility, enables a clear workflow control, evolves the system according to modern functional use-cases to facilitate coming feature requests from new and old PanDA users. This paper describes Pilot 2, its architecture and place in the PanDA hierarchy. Furthermore, its ability to be used either as a command tool or through APIs is explained, as well as how its workflows and components are being streamlined for usage on both grids and opportunistically used resources for and beyond the ATLAS experiment.

[*] Corresponding author: paul.nilsson@cern.ch

1 Introduction

The PanDA Pilot [1] has been used by ATLAS [2] and other experiments for well over a decade. Over time it became more and more challenging to maintain the aging code base and to add new features to it, which are often highly complex. It is a well-known problem in software engineering that eventually even a solid code base will grow old and contain outdated functions. At that point one has to make the decision whether to refactor the code significantly or to rewrite it. Out of necessity, we had already spent years of partial refactoring and therefore decided to rewrite the original PanDA Pilot from scratch and create a Pilot 2 to benefit from a more modern design approach and get rid of all of the old code constructs at once. The rewrite was also necessary in order to meet the demands of further extending PanDA [3] beyond grids and ATLAS. After a two-year effort, the Pilot 2 (henceforth referred to as 'Pilot' for simplicity) project is now in its final stages of development and has entered testing in the production system.

1.1 What does the Pilot do?

The task of the Pilot is to monitor and execute work units on a worker node, either on the job level or on the more finely grained event level [4]. On the job level (the normal run mode), the work unit is a payload that a user or production system wants to execute. The payload is downloaded from a server and has certain requirements, e.g. input and output files, that are normally staged by the Pilot, and needs a working environment (incl. containers) that is setup by the Pilot. In the so-called direct access mode, the normal stage-in can be skipped and the input files are opened remotely by the payload itself. On the event level, the Pilot running in the event service mode launches and feeds a payload with event ranges (a set of events to be processed) that are downloaded from a server. This has the major advantage that if the resource disappears and the Pilot is aborted, all previously processed events are saved since they have already been uploaded to a server and in the case that the Pilot does not have time to abort, only the last event is lost – as opposed to losing the entire job with all processed events which can happen in the normal run mode.

1.2 How does the Pilot fit into the PanDA hierarchy?

The Pilot is executed on the worker nodes on local resources, on grids and clouds, on HPCs and on volunteer computers. It is downloaded and run by wrapper scripts that are sent by Pilot factories to the worker nodes via batch systems. A Pilot interacts with the PanDA server either directly, via a local instance of the ARC Control Tower (a job management framework used on Nordugrid [5]) or with the resource-facing Harvester service which provides resource provisioning and workload shaping [6].

2 Components

As opposed to the original PanDA Pilot, the new Pilot version is component based, with each component being responsible for different tasks. The main tasks are sorted into controller components, such as Job Control, Data Control, Payload Control and Monitor Control. Job Control handles the job objects that are either downloaded from a server or read from file. Data Control takes care of replica lookups, selection and transfers. Payload Control prepares for execution, selects run mode (normal or event service), executes and monitors the payload until it finishes by verifying parameters that are relevant for the payload (e.g. size checks). Monitor Control keeps track of the Pilot itself and makes sure that it does not run longer than it is allowed to. It also monitors all threads spawned by the other components. The

Information System component presents an interface to a server-side database containing knowledge about the resource where the Pilot is running (e.g. which copy tool to use and where to read and write data).

3 Pilot APIs

Normally, the Pilot is used as a command-line tool. In case this is not wanted but some Pilot functionality is still needed, an external user may call relevant functions via simplified Pilot APIs that provide convenient access to internal Pilot modules and functions that otherwise may be difficult to use. E.g. Harvester is using the Pilot Data API in production for data transfers on HPCs. Other APIs include the Communicator API (server interactions) and Services API (benchmarking, memory monitoring and analysis package).

4 Pilot workflows

The Pilot workflows refer to the particular run mode the Pilot should use, such as the Standard workflow and HPC Pilot workflow. A desired workflow is selected by the wrapper script that launches the Pilot, and it determines how the jobs will be processed.

4.1 Standard workflow

The Standard workflow relies on internal python queues to keep track of the job objects, and consists of multiple steps executed in parallel using threads (see Figure 1). Each thread polls a queue until it gets a job object to process; after processing, the result is put in another queue for further processing and the thread starts polling its input queue again. The job object itself, is an entity that contains all necessary information about running the payload such as software release version, parameters for payload setup, transfer type of input files, etc.

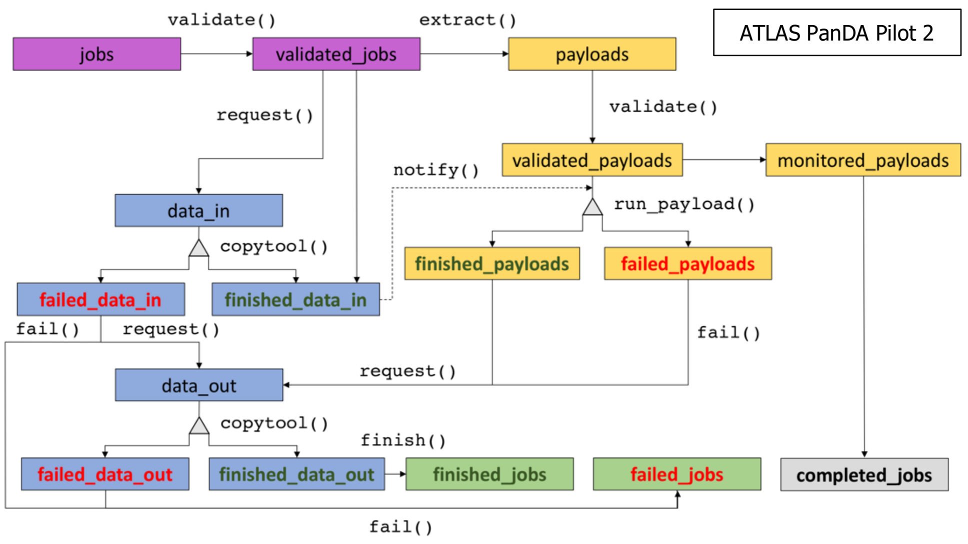

Fig. 1. The internal generic flow of job objects. A downloaded job is inserted into the jobs queue and ends up in either the finished_jobs or failed_jobs queue. The completed_jobs queue contains a copy of a job object that has completed running (finished or failed) and is used for internal bookkeeping.

Specifically, in the Standard workflow the Pilot performs payload download, setup, stage-in, execution and stage-out, along with various verifications, monitoring and server job updates at selectable intervals.

4.2 HPC Pilot workflow

The HPC Pilot refers to a dedicated workflow used on HPCs. When this mode is selected, the normal workflow of the Pilot is skipped in favour of a streamlined workflow more relevant for HPCs. Resource specific code, such as environmental setup, is kept in plugins. Dedicated plugins are in development and testing for Titan (OLCF) [7, 8], Theta (ALCF) [9], Cori and Edison (NERSC) [10, 11] and BNL [12].

4.2.1 Generalized workflow on a typical HPC

The architecture on an HPC may be quite different from a typical grid site. An important common difference is that the outbound connections from the worker nodes may be restricted. Another limitation is the number of job slots in the local batch system of the HPC. While the size of a job slot can be very large (tens or even hundreds of thousands of cores), the number of job slots can be very limited (e.g. on the Leadership Computing Facilities) and is defined in the local scheduling policy. On the Titan supercomputer, only a few batch jobs may run at the same time and only four can be queued [13]. To cope with these issues, a special workflow was implemented in the Pilot.

The Pilot acts like an MPI application under the control of the Harvester service and runs a set of jobs in an assembly. Harvester takes care of prestaging input data, stage-out of output (using the Pilot Data API) and communication with the PanDA server, while the Pilot intercommunicates with Harvester through the shared filesystem. The Pilot reads the job definitions from pre-placed files and declares outputs for later processing by the Harvester service. Since all external communications were delegated to Harvester, the general Pilot workflow could be simplified in the communication part but was extended with optional functionality required to cover the specifics of the HPC. Different HPCs may require different preprocessing and post-processing steps: for example, it may be needed to define additional environment variables or add additional setup steps. The HPC Pilot workflow has a common procedure for all HPCs and performs the following steps: job description retrieval, working directory setup, payload execution, declaration of execution results, output files and logs for later stage-out. All special procedures are placed in HPC specific plugins and include: optimization of payload execution (using high-speed local storage for input data and working directory), infrastructure specific setup, etc. On Titan it is not possible to use the common procedure to setup the required environment on the computing node for ATLAS payloads, but it is possible with a special procedure, which should be performed instead of the common procedure and is optional for other HPCs.

Special treatment was given to the Pilot features that use local files to minimize the I/O of the shared file system. Input data and working directory should be placed on dedicated high-speed storage associated with the computing node (SSD, RAM disk, etc.) to minimize the impact on the shared file system. After the payload execution, the Pilot copies the results of execution to the shared file system, from where it will be staged out by the edge service (Harvester) at a later time.

The Pilot running the HPC workflow has been successfully validated on the Titan supercomputer.

5 Beyond ATLAS

The PanDA system is currently used by several experiments including ATLAS, COMPASS [14], LSST [15] and IceCube [16] as well as with software applications for Lattice QCD and Molecular Dynamics [17, 18]. One of the design goals of the Pilot 2 project was to facilitate Pilot development and usage by new users (such as experiments). Since the Pilot keeps user specific code in plugins, as well as being a component-based system, it is easy to support new workflows. In case the standard workflow is not relevant for the new user, an entirely new workflow may be implemented that will only use other relevant Pilot modules and functions.

6 Summary

The original PanDA Pilot used by ATLAS and other experiments has been rewritten. The new version, Pilot 2, is now in its final stages of development and has entered the commissioning stage. It has been designed especially with HPCs in mind, for which it has a dedicated workflow with plugins being prepared for several HPCs. In particular, it has been successfully validated on Titan.

The workflow mechanism represents a major difference to the previous Pilot version, as it makes it much easier to implement entirely new workflows that may use as much or as little as is relevant from the Pilot code stack. The new Pilot also has easy-to-use APIs for external use. E.g. the resource-facing Harvester service is relying on the Pilot Data API for file transfers.

Pilot 2 is currently being tested on all grids used by ATLAS as well as with Harvester on HPCs and will soon be used for running all production and user analysis jobs.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, under contract number DE-SC0012704, and was funded in part by the Russian Ministry of Education and Science under contract No 14.Z50.31.0024.
This research used resources of the National Energy Research Scientific Computing Center, a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.
This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.
This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

1. P. Nilsson et al., J. Phys.: Conf. Ser. **513** 032071 (2014)
2. ATLAS Collaboration, JINST **3** S08003 (2008)
3. F. H. Barreiro Megino et al., J. Phys.: Conf. Ser. **898** 052002 (2017)
4. P. Calafiura et al., J. Phys.: Conf. Series **664** 092025 (2015)
5. M. Ellert et al., Future Generation Computer Systems **23** n.2 219-240 (2007)
6. T. Maeno et al., To be published in *Proceedings of 23rd International Conference on Computing in High Energy and Nuclear Physics,* EPJ Web of Conferences
7. Titan (OLCF), https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan
8. K. De et al, J. Phys. Conf. Series **664** 062035 (2015)
9. Theta (ALCF), https://www.alcf.anl.gov/theta
10. Cori (NERSC), https://www.nersc.gov/users/computational-systems/cori

11. Edison (NERSC), http://www.nersc.gov/users/computational-systems/edison
12. KNL cluster (BNL), https://www.racf.bnl.gov/experiments/sdcc/knl-cluster
13. Scheduling policy at OLCF for Titan: https://www.olcf.ornl.gov/for-users/olcf-policy-guide
14. P. Abbon et al, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume **577**, Issue 3, Pages 455-518 (2007)
15. D. Sweeney, *Ground-based and Airborne Telescopes, Proceedings of the SPIE*, Volume **6267**, id. 626706 (2006)
16. R. Abbasi et al., Nuclear Instruments and Methods in Physics Research A. **601** (3): 294 (2009)
17. R. Babich et al., *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis,* Article No. 70 (2011)
18. The CHARMM molecular modeling software http://www.charmm.org