# Simplified pilot module development and testing within the ATLAS PanDA Pilot 2.0 Project

**D. Drizhuk[1,a], P. Nilsson[2], W. Guan[3]**
**on behalf of the ATLAS Collaboration**

[1] National Research Centre Kurchatov Institute, 1, pl. Akademika Kurchatova, Moscow, 123182, Russia

[2] Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973-5000, USA

[3] Department of Physics, University of Wisconsin-Madison, 4425 Chamberlin Hall Madison, WI 53706, USA

E-mail: [a] d.drizhuk@gmail.com

The Production and Distributed Analysis (PanDA) system has been developed to meet ATLAS production and analysis requirements for a data-driven workload management system capable of operating at the LHC data processing scale.

The PanDA pilot is one of the major components in the PanDA system. It runs on a worker node and takes care of setting up the environment, fetching and pushing data to storage, getting jobs from the PanDA server and executing them. The original PanDA Pilot was designed over 10 years ago and has since then grown organically. Large parts of the original pilot code base are now getting old and are difficult to maintain. Incremental changes and refactoring have been pushed to the limit, and the time is now right for a fresh start, informed by a decade of experience, with the PanDA Pilot 2.0 Project.

To create a testing environment for module development and automated unit and functional testing for next generation pilot tasks, a simple pilot version was developed. It resembles the basic workflow of pilot tasks used in production and provides a simple and clean template for module construction. The miniPilot has a simple structure and is easy to use for development, testing and debugging server-client interactions with new protocols and application interfaces. The unit and functional test system will be developed on top of the miniPilot, and will be used to run automatic tests.

This paper describes the miniPilot and the test system that will be used during the Pilot 2.0 Project.

Keywords: Distributed computing, pilot, testing framework, grid, PanDA

The work was supported in part by the Russian Ministry of Education and Science under contract # 14.Z50.31.0024

## The PanDA system

The PanDA system [Maeno, et al., 2012] is a data-driven production and analysis job and workload management system. Originally it was designed to support the ATLAS experiment [ATLAS Collaboration, 2008], one of the four major experiments at the CERN Large Hadron Collider. Recently it was extended to support other projects and experiments. Another recent extension brings the computational power of Leadership Computing Facilities and supercomputers like MIRA, NERSC and the third most powerful computer in the world, Titan. PanDA also supports a new data processing workflow, called Event Service [Calafiura, De K., …, 2015], which can be used both on normal grid sites as well as on supercomputers.

## The Pilot

The Pilot [Nilsson et al., 2013] is one of the fundamental PanDA components. It runs on the worker node where it sets up and configures the environment. It receives a job from the PanDA server, downloads all necessary data, starts the job and supports it through all of its lifetime. It monitors the job as well as the state of the worker node, and does logging. Finally, it uploads the results and cleans up the environment, and may start a new job.

As one of the original components of PanDA, the Pilot was developed by and for ATLAS on the Open Science Grid (OSG) and to support its workflows and storage systems. During the extension of the PanDA system to support other grids and experiments, the Pilot had to be refactored and extended. It is important to note that the Pilot was originally designed for the first release of the PanDA system and since then has been evolving through patches. Due to constant extensions and fixing, the code eventually become very complex and sometimes difficult to maintain, especially when adding new major features. Needless to say, since the core of the Pilot is now over a decade old, it contains some obsolete solutions and execution branches handling outdated use cases with portions of the code being redundant. At this point the code is very difficult to maintain, not to mention extending. The refactoring of this code is also a slow process due to high obscurity of existing code and many layers of hard-coded solutions and workarounds built on top of each other.

## Pilot 2.0

To solve these issues, the decision was made to build a new version from scratch. This new version is known as Pilot 2.0 and will be designed to be clean and simple, and easy to extend. It should follow modern development conventions, have full code and usage documentation, and use modern technologies. It should be easy to install, extend and use. Moreover, it must be experiment independent and easy to integrate, though it must support all the requirements, conditions and experiments present in the current version.

## Testsuite for Pilot 2.0

The Pilot 2.0 development is going to be distributed, and thus need to have an automated testing environment and strict rules for the development process enforced by the testing system. Having access to a testing system is not only practical but essential as it will contribute to making the overall code base robust. The testsuite also has a so called miniPilot that will be used by the Pilot developers during the design stage of the Pilot 2.0 project. The testing system and the miniPilot are described in the following sections.

## MiniPilot

A simplified version of the Pilot, called the miniPilot, was created to support several tasks primarily during the design stage of the Pilot 2.0 project. It is meant to be lightweight, easy to understand, clean, fully documented from the start and easy to extend. It implements the basic workflow of the older Pilot version, although without any special environment configuration and support. The miniPilot is intended to be used for just one type of task in a predefined environment. It should be a fully working Pilot but not necessarily support all the workflows and use cases of the main Pilot version. It should also meet PEP8 code conventions [van Rossum, Warsaw, …, 2001] and provide an example of the coding style.

During the implementation of the miniPilot, a few additional properties were realised; it largely became OS-independent (it does use the linux-specific Rucio data management system [Serfon, et al., 2016] as it is required by ATLAS, but may be simply changed), it uses a modern version of python and libraries, it minimises external calls, has proper logging and uses a new job description format. The number of supplementary functions was brought to a minimum to make the code clean and readable.

The workflow of this version is presented in figure 1.

The miniPilot may evolve into a version that could be used by adopting PanDA system users as an entry level Pilot for testing and development.

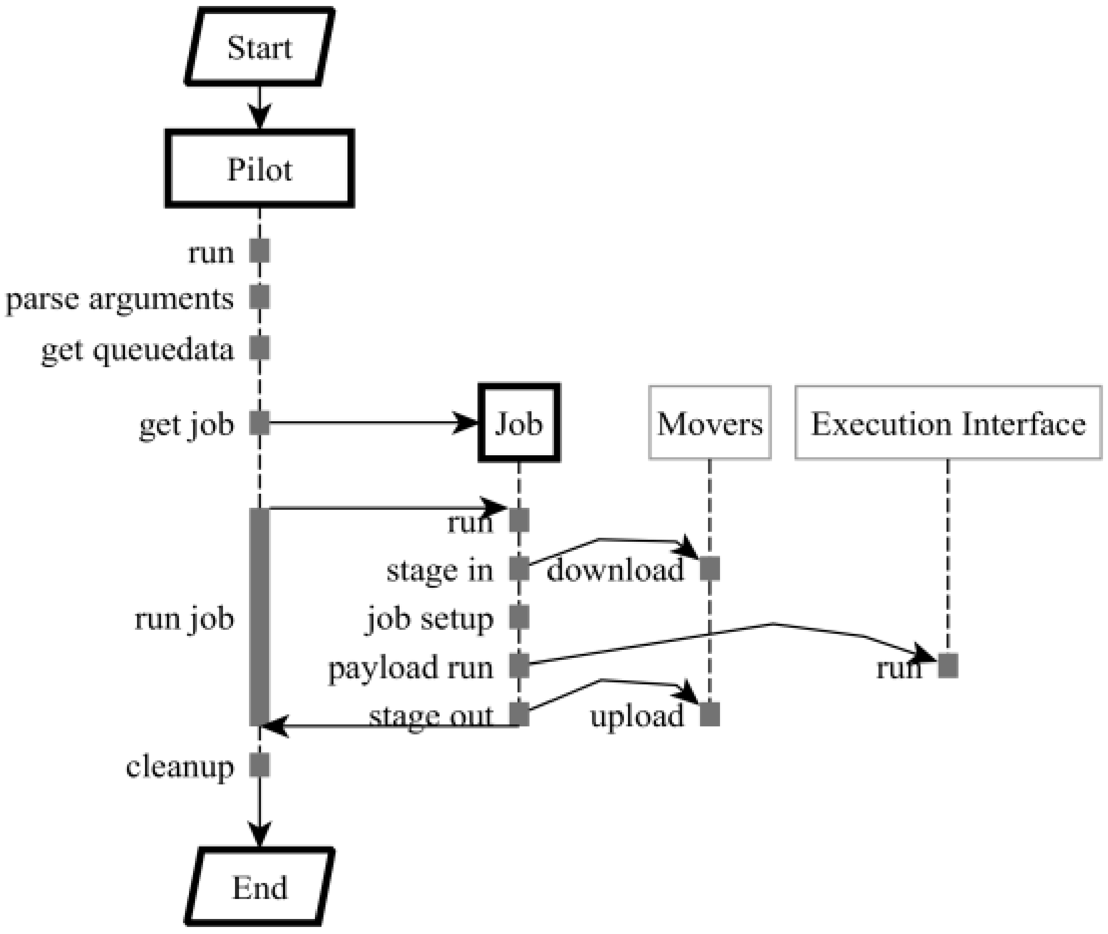

Figure 1. MiniPilot workflow

## Testing framework

A testing framework was created with the miniPilot as its initial customer. It provides code validation as well as unit and functional testing. Unit testing can be used to cover most workflows, necessary error handling, and interface development. It can be used to test critical functions in all Pilot modules. Basically, components can have their own unit tests and the testing system will execute them.

Since the Pilot is a component that depends on PanDA, the testing framework needs to have tests that are suitable and relevant. Because of this, a solution like the Travis CI [Travis project, 2016] will not work since it runs in a cloud without the possibility to setup PanDA-specific subsystems and interactions with the PanDA server.

The framework was developed using the Github API [Github API documentation, 2016]. When one submits a patch as a pull-request, the API triggers the testing system to test the patch. As the tests are performed, every log is submitted as a comment to the pull-request and is marked as valid or invalid. Later, the manager can decide whether to approve the merging of the patch into the current development branch or not. The workflow is depicted in figure 2.

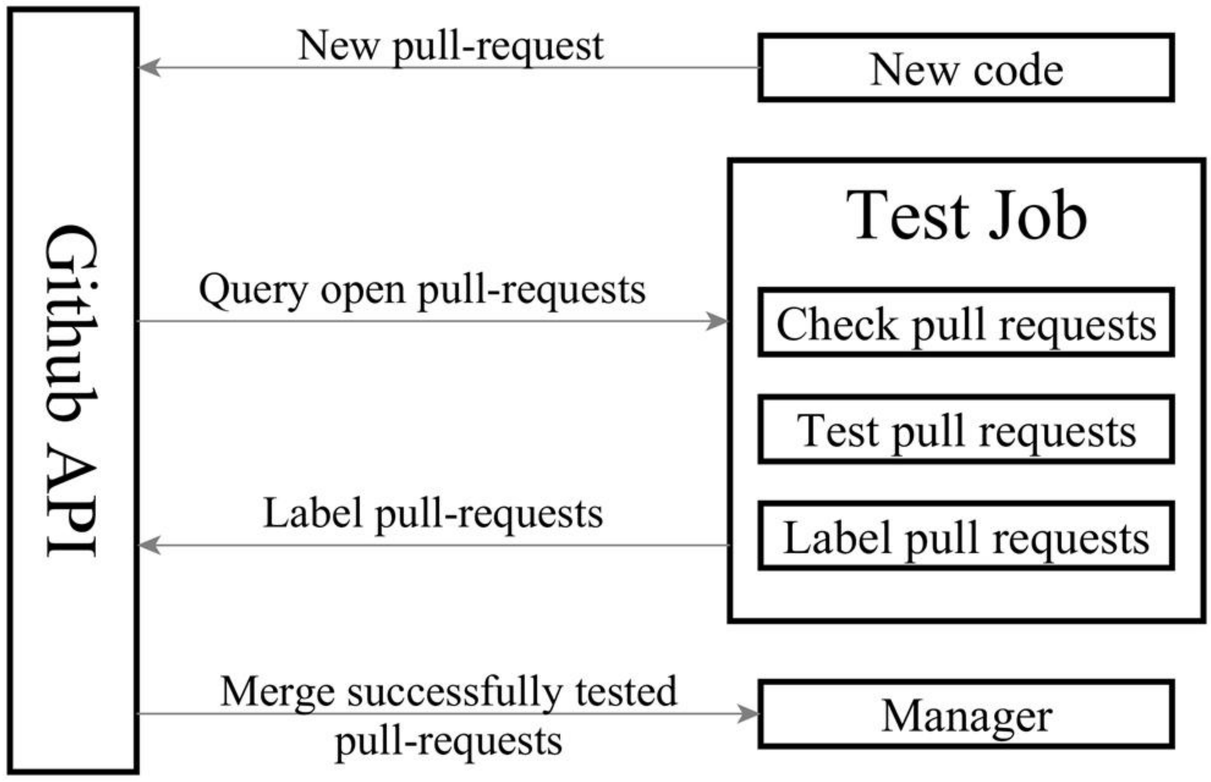

Figure 2. Test Framework workflow

## Conclusion

The basic workflow of the PanDA Pilot was simplified and implemented in a simple example program. This program was written according to agreed code conventions and meeting all requirements. Around this program, the test framework was designed with an automated testing platform connected to a repository.

The testsuite discussed in this proceeding will greatly benefit the development of the PanDA Pilot 2.0.

## References

*The ATLAS Collaboration.* The ATLAS Experiment at the CERN Large Hadron Collider // J. Inst. — 2008. — Vol. 3. — S08003.

*Maeno T. et al.* Evolution for the ATLAS PanDA production and distributed analysis system // J. Phys. Conf. Ser. — 2012. — Vol. 396. — P. 032071.

*Calafiura P., De K., Guan W. et al. on behalf of the ATLAS Collaboration.* The ATLAS Event Service: A new approach to event processing // J. Phys. Conf. Ser. — 2015. — Vol. 664, Issue 3. — P. 062065.

*Nilsson P. et al. on behalf of the ATLAS Collaboration.* Next Generation PanDA Pilot for ATLAS and Other Experiments // J. Phys. Conf. Ser. — 2013. — Vol. 513, Issue 3. — P. 032071.

*Van Rossum G., Warsaw B., Coghlan N.* Style Guide for Python Code // Python Enhancement Proposals. — 2001. — Vol. 8. [Electronic resource]: http://www.python.org/dev/peps/pep-0008/ (accessed 19.10.2016).

*Serfon C. et al., for the ATLAS Collaboration.* Rucio, the next-generation Data Management system in ATLAS // J. Nucl. Phys. B Proc. — 2016. — Vol. 273-275. — P. 969-975.

Travis project. [Electronic resource]. URL: https://travis-ci.org/ (accessed 19.10.2016).

Github API documentation. [Electronic resource]. URL: https://developer.github.com/v3/ (accessed 19.10.2016).