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

Daniel Drizhuk, Paul Nilsson, Wen Guan
on behalf of the ATLAS Collaboration

The PanDA System

- PanDA is the Production and Distributed Analysis system used and developed by the ATLAS Experiment
- Designed to meet ATLAS requirements for a data-driven workload management system for production and distributed analysis processing capable of operating at LHC data processing scale
- PanDA has been extended to support experiments other than ATLAS (e.g. AMS)
- It has also recently been extended to support Leadership Computing Facilities (e.g. Titan) and new workflows (Event Service)

PanDA Pilot

- Runs on the WN
- System configuration
- Late job acquisition
- Environment setup
- File staging in/out
- Job setup & starting
- Monitoring & logging

- PandA Pilot was originally developed for ATLAS, but has been modified to support other experiments
- Other parties and experiments are interested in using the PanDA system
- The pilot code is complex due to constant development and it can be very time-consuming to add new functionality
- Some of the code base is getting old and is difficult to maintain
- Refactoring is a slow process that has already been going on for years and does not always have highest priority

→ Pilot needs a good new start

Pilot 2.0

- Proper code conventions
- Proper use of developed and debugged libraries/code
- Up to date interpreters
- Full code and usage documentation
- Easy installation and extension
- Supercomputer support
- Experiment independent workflow
- Easy experiment integration

Testsuite for pilot module development

- Pilot 2.0 development is a distributed process
- One needs a clean and very basic version of Pilot to test new modules
  - If one develops main pilot modules, eg movers
  - If one integrates pilot to another experiment
  - If one adds a new computing sites class
  - If one develops new protocols and data views
  - And much more
- This suite is essentially necessary for pilot developers

Values of minipilot

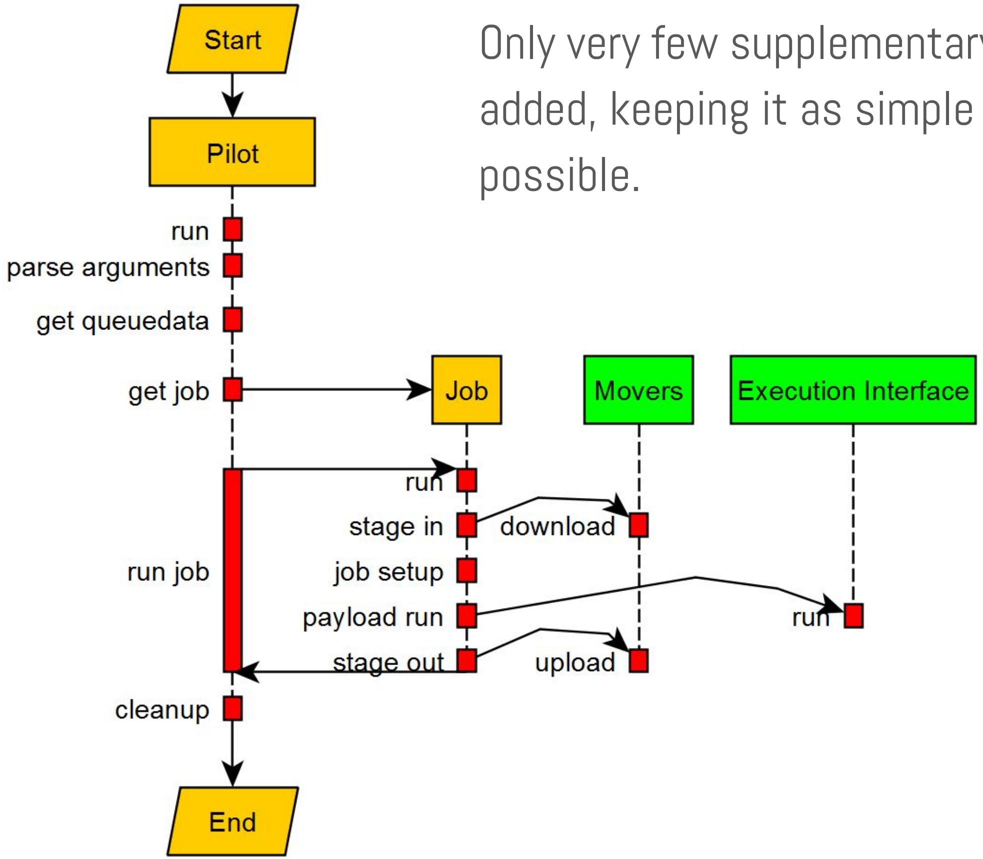
Basic

- Clean
- Documented
- Simple
- Implements basic pilot workflow
- Easy to extend

Additional (due to implementation):

- OS independent
- Python 2.7
- Use of Libraries
- Proper logging
- New job descriptions

Workflow



Only very few supplementary functions added, keeping it as simple as possible.

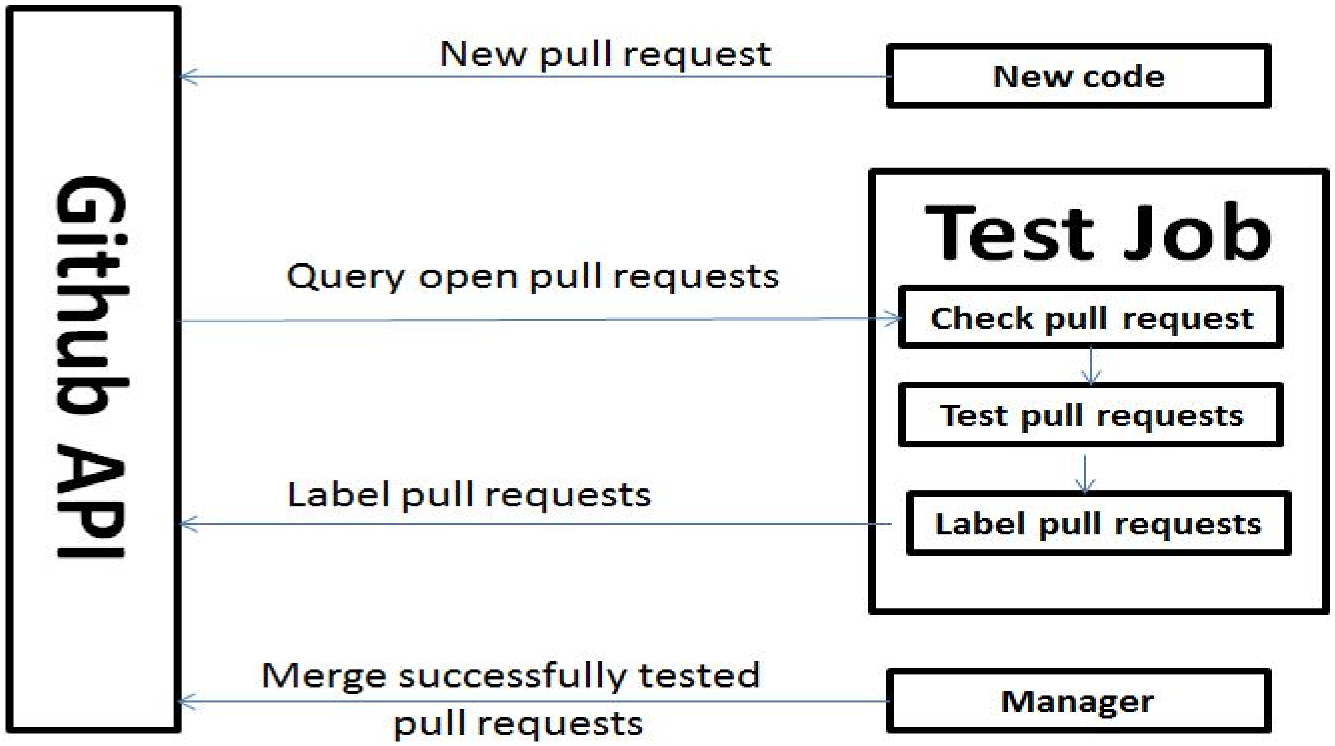
Test Framework

- Pattern Validation
  - Avoid typing errors
  - Keep code clean
- Developing unit test functions in pilot
  - Unit test includes successful workflow
  - Unit tests also include workflows to test pilot's ability to handle errors such as stagein failure.
  - Unit test must cover component interface
  - Unit test can cover important function in components

Test Framework(Cont)

- All new codes are submitted with a github pull request
- Automatic Test workflow
  - Test job can be in Jenkins
  - Test job uses github api
  - Test job queries ‘open’ but not tested pull requests
  - Test job pulls the code for a pull request
  - Test job runs the unit test function
  - Test job labels tested pull request with “OK” or “ERROR”, and adds the unit test output to pull request’ s body.
- Manager merges successfully tested pull requests

Test Framework(Cont)



Conclusion

The minimal testsuite for Pilot developers has been created

The basic Pilot workflow has been tested

New coding and development conventions for Pilots have been established

Automatic testing platform has been created

Pilot 2.0 has now a clean runway and a green light