Global heterogeneous resource harvesting: the next-generation PanDA Pilot for ATLAS

A. Anisenkov[1,2], D. Drizhuk[3], W. Guan[4], M. Lassnig[5], P. Nilsson[6,*], D. Oleynik[7,8] on behalf of the ATLAS Collaboration

[1] Budker Institute of Nuclear Physics, Siberian Branch of Russian Academy of Sciences, 11, Akademika Lavrentieva prospect, Novosibirsk, 630090, Russia
[2] Novosibirsk State University, 2, Pirogova street, Novosibirsk, 630090, Russia
[3] National Research Centre Kurchatov Institute, 1, pl. Akademika Kurchatova, Moscow, 123182, Russia
[4] University of Wisconsin-Madison, Department of Physics, 4425 Chamberlin Hall Madison, WI 53706, USA
[5] CERN, European Laboratory for Particle Physics, CH-1211 Geneva 23
[6] Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973-5000, USA
[7] University of Texas at Arlington, 502 Yates Street, Arlington, TX 76019-0059, USA
[8] Joint Institute for Nuclear Research, Joliot-Curie 6, Dubna, 141980, Russia

[*]Corresponding author: Paul.Nilsson@cern.ch

Abstract. The Production and Distributed Analysis system (PanDA), used for workload management in the ATLAS Experiment at the LHC for over a decade, has in recent years expanded its reach to diverse new resource types such as HPCs, and innovative new workflows such as the Event Service. PanDA meets the heterogeneous resources it harvests in the PanDA Pilot, which has embarked on a next-generation reengineering to efficiently integrate and exploit the new platforms and workflows. The new modular architecture is the product of a year of design and prototyping in conjunction with the design of a completely new component, Harvester, that will mediate a richer flow of control and information between Pilot and PanDA. Harvester will enable more intelligent and dynamic matching between processing tasks and resources, with an initial focus on HPCs, simplifying the operator and user view of a PanDA site but internally leveraging deep information gathering on the resource to accrue detailed knowledge of a site's capabilities and dynamic state to inform the matchmaking. This paper will give an overview of the new Pilot architecture, how it will be used in and beyond ATLAS, its relation to Harvester, and the work ahead.

ATL-SOFT-PROC-2017-074
05 December 2017

1. Introduction

The PanDA [1] Pilot is responsible for executing payloads on the worker nodes on local resources, on grids or clouds, on (non-restrictive) High Performance Computers (HPCs) and on volunteer computers. It has been used in the ATLAS Experiment [2] at the LHC for well over a decade, and is in constant development. Over time, however, it has become more and more challenging to maintain the now 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 a solid code base will grow old and contain outdated and deprecated functions, at which point one has to make the decision whether to refactor the code significantly or to rewrite it. In our case, we had already spent years of partial refactoring when we 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 old code constructs in one go. Pilot 2 is a fresh start that provides more functionalities than the original Pilot, such as APIs for external usage, although some of its traditional responsibilities can be moved to a new member of the PanDA family, Harvester [3]. This new component is a service between the PanDA server and the Pilot especially designed for use on HPCs, and provides resource provisioning and workload shaping. It enables more intelligent and dynamic task matching with the resources, simplifying the operator and user view of a PanDA site but

internally utilizing various kinds of information about the resource and a site's capabilities to boost the performance of tasks as well as sites.

There are several functional overlaps between the Pilot and Harvester, such as payload setup and copy tools, that are implemented in the Pilot and imported and used by Harvester to avoid code duplication.

The Pilot 2 architecture follows a component-based approach which in comparison to the previous Pilot implementation allows to exploit an enhanced system flexibility, enables a clear workflow control, evolves the system according to modern functional use-cases and covers upcoming feature requests from new PanDA users.

2. Macroscopic system structure

The macroscopic system structure (Figure 1) shows the relations between the various Pilot components, how they communicate with each other and the flow of information. The main tasks or functions are sorted into several controller components, such as Job Control, Payload Control and Data Control. There is also a set of components with auxiliary functionalities, e.g. Setup, Information Service and Monitoring. There are two types of monitoring, one for internal use which monitors threads and one that is tied to the job and checks parameters that are relevant for the payload, e.g. size checks and that the payload is still producing output at a regular basis.

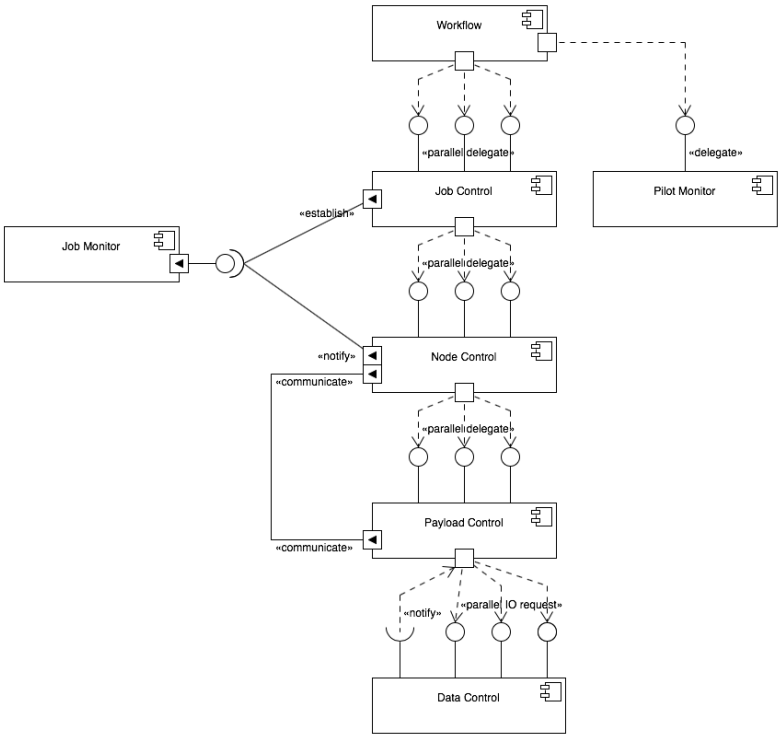

Figure 1. Macroscopic system structure.

Code that is specific to a PanDA Pilot user is handled by plug-in mechanisms and is stored in its own directories to keep it separated from the core Pilot code. New workflows can be defined in case the generic Pilot workflow is not sufficient, while still benefitting from core Pilot functionalities.

3. Parallel streaming data flow

Figure 2 shows the generic flow of the job objects and how they move through the internal queues in the different Pilot components. A job object is an entity that is containing all necessary information about the payload (e.g. software release version, parameters for payload setup, transfer type of input files, etc).

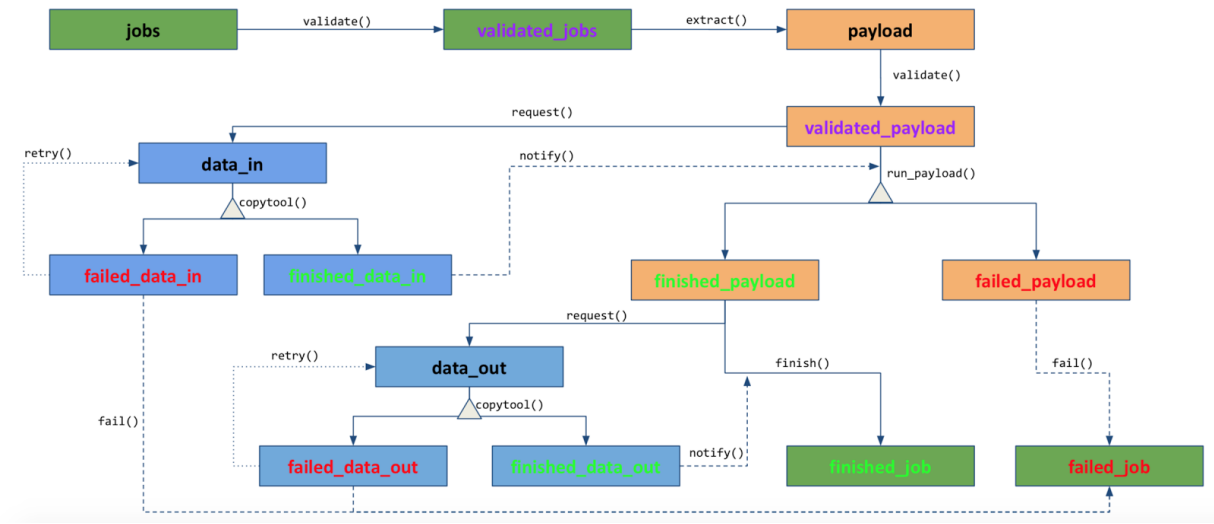

Figure 2. The generic flow of the job objects.

When a job has been verified, i.e. ready for execution, it is moved from the job queue in the Job Control to a validated job queue. The Payload Control takes it from there and executes it. When the payload has finished, any output is transferred by the Data Control to a relevant storage element (SE) along with the job log. As a last step, the Pilot informs the server about the final state of the job.

An Event Service [4] job is handled internally in a similar way to a 'normal' job but the workflow differs somewhat from the generic workflow (see Figure 3). While both job types have similar job definitions, there are differences especially with the stage-out process. In an Event Service job, input data is read directly from storage (also possible for normal jobs) and accessed via event ranges (unlike normal jobs). The output from a processed event range is immediately and asynchronously transferred to an object store, while a normal job only transfers the output at the end of the job to a normal SE.

An Event Service job may also use an additional service application, called the Prefetcher. This tool is an application that is launched by the Pilot. Its task is to asynchronously prefetch needed data from a remote location and store it locally, and is a key component of the Event Streaming Service (ESS) [4].

While this section has described workflows used by ATLAS, it should be stressed that Pilot 2 allows for other workflows to be implemented while still benefitting from core Pilot functionalities such as monitoring, copy tools, containerization of executable commands, etc.

4. Pilot APIs

Decoupling system logic into well-defined components with a corresponding common API helps to effectively consolidate system modules and in particular share code functionality, integrate and communicate with external applications like Harvester, as well as for dedicated testing.

Selected Pilot functionality is exposed to external users via the following APIs:

1. Data API: functions for performing stage-in and stage-out.
2. Communicator API: communication tools for external services.
3. Environment API: interface to the job execution environment on HPCs.
4. Services API: functions related to external service tools such as benchmarking and memory monitoring.

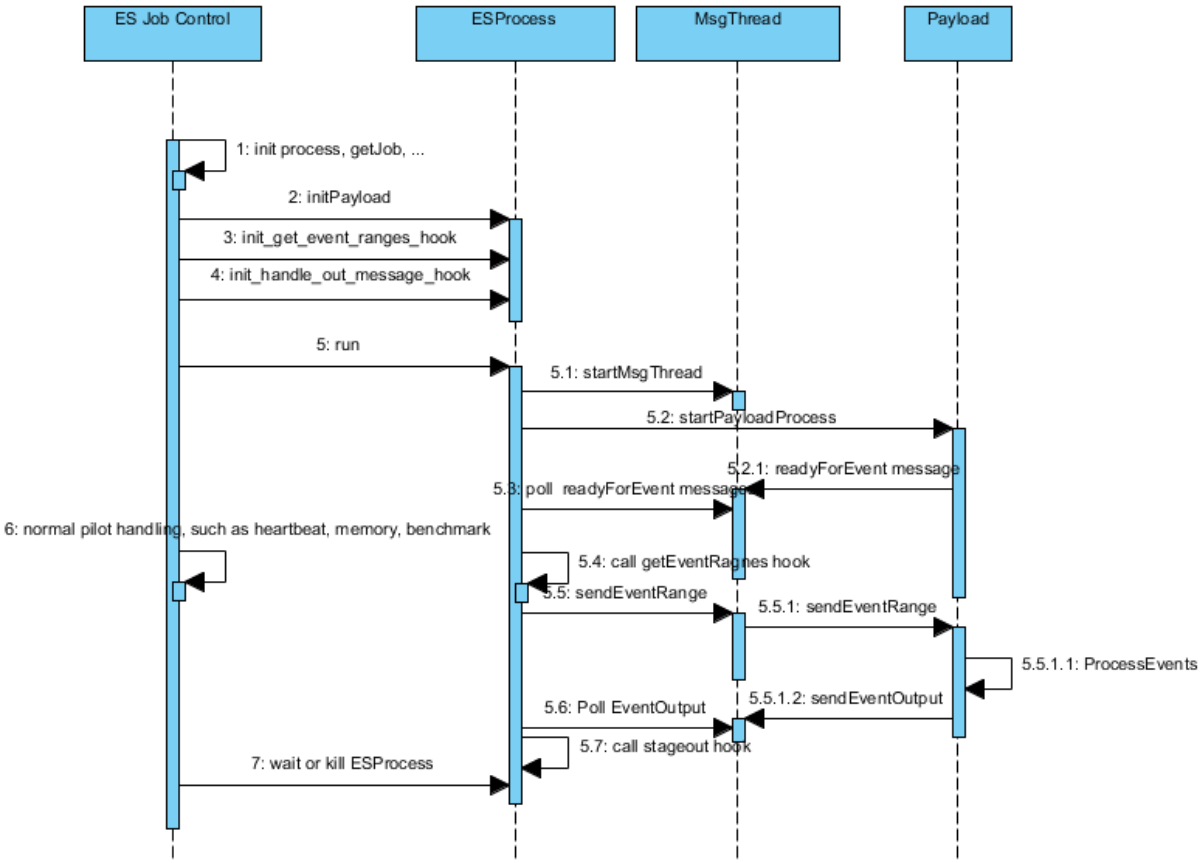

Figure 3. Sequence diagram for the Event Service workflow as used in ATLAS.

Server communications such as the getJob and updateJob commands can be issued using the Communicator API and may be used by both Pilot and Harvester. Furthermore, this API provides the interface to the Information Service component which has a set of low-level information providers to aggregate, prioritize, hide dependency to external storages and expose details for queues, sites, and storages, in a unified and structured way to all Pilot modules.

On HPCs, the Pilot is normally not running on the worker nodes but Harvester and the Yoda [5] tool can use the Pilot Environment API to determine how the payload should be set up. The Services API exposes benchmarking functions (currently an interface to the CERN Cloud Benchmark Suite [6]) that may be used by Harvester to add a benchmark step to the MPI rank 0 which would execute it before the actual payload. It could also be used on the resources where the Pilot is normally not used (such as on the ATLAS Tier-0) and also contains functions to interpret and forward the output of the benchmark tool to the relevant service. On normal grid sites, the current PanDA Pilot executes the benchmark suite during stage-in once per a hundred jobs and the results are gathered and studied on Elasticsearch [7].

The Services API also exposes functions that can be used for memory monitoring. Normally, the Pilot runs a memory monitoring tool in parallel with the payload to make sure the payload does not consume more resources than it is allowed to. The server is informed on each job update with current memory measurements during running which allows for real-time memory monitoring. On HPCs, Yoda can use the memory monitoring functions from the Services API to add a memory monitoring step to the MPI rank 0 which can execute the tool in parallel with the payload. The API contains functions for uploading the output to the relevant service.

5. Harvester-Pilot integration

The idea of the Harvester component is to facilitate the usage of PanDA on HPCs by moving away responsibilities from the Pilot, while still using some of the Pilot functionalities via APIs. Details about HPC queues and special workflows required for the HPC in question should not be handled directly by the Pilot. Instead, plug-ins to Harvester are being developed for each HPC. These plug-ins, as well as Harvester itself, will use relevant Pilot components to avoid code duplications; e.g. file transfers, job definition downloads and functions for communicating with the server. In case Harvester is not used, the Pilot communicates directly with the server.

5.1. The road ahead

The Pilot 2 project is now in its second year of existence and is currently in the implementation stage. Basic functionality (file transfers, payload execution and monitoring) is in place and the first of the Pilot APIs, the Data API, was delivered in early 2017 for external use by Harvester. Supplementary components, such as the Information Service and Payload components, are being developed and the general code base is expanding rapidly. Additional APIs are in late planning and will be implemented next, including the Services API.

6. Summary

The PanDA Pilot has been in use in ATLAS for well over a decade. It was decided in 2016 to replace it with an entirely new product, Pilot 2, to more efficiently meet the demands of ATLAS and beyond on grids, clouds, volunteer computers and HPCs. It is being developed in parallel with another PanDA product, Harvester, located between the PanDA server and the Pilot, and will primarily be used on HPCs. Both Pilot 2 and Harvester are in rapid development and are expected to be used in production in 2018.

Acknowledgements

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.

References

[1] ATLAS Collaboration, 2008 JINST 3 S08003

[2] F. H. Barreiro Megino et al., “PanDA for ATLAS distributed computing in the next decade”, Proceedings of 22nd International Conference on Computing in High Energy and Nuclear Physics, J. Phys. Conf. Ser. 898 (2017)

[3] Harvester: https://github.com/PanDAWMS/panda-harvester/wiki

[4] P. Calafiura et al, “The ATLAS Event Service: A New Approach to Event Processing”, Proceedings of 21st International Conference on Computing in High Energy and Nuclear Physics, J. Phys. Conf. Ser. 664 (2015)

[5] P. Calafiura et al, “Fine grained event processing on HPCs with the ATLAS Yoda system”, Proceedings of 21st International Conference on Computing in High Energy and Nuclear Physics, J. Phys. Conf. Ser. 664 (2015)

[6] Manfred Alef et al, “Benchmarking cloud resources for HEP”, Proceedings of 22nd International Conference on Computing in High Energy and Nuclear Physics, J. Phys. Conf. Ser. 898 (2017)

[7] Elasticsearch, https://www.elastic.co/products/elasticsearch