Managing the ATLAS Grid through Harvester

Fernando Harald Barreiro Megino[1,*], *Aleksandr* Alekseev[2], *Frank* Berghaus[3], *David* Cameron[4], *Kaushik* De[1], *Andrej* Filipcic[5], *Ivan* Glushkov[1], *FaHui* Lin[1], *Tadashi* Maeno[6], and *Nicolò* Magini[7] on behalf of the ATLAS Collaboration [†]

[1]University of Texas at Arlington, United States of America
[2]Tomsk Polytechnic University, Russia
[3]University of Victoria, Canada
[4]University of Oslo, Norway
[5]Jozef Stefan Institute, Slovenia
[6]Brookhaven National Laboratory, United States of America
[7]Iowa State University, United States of America

Abstract. ATLAS Computing Management has identified the migration of all computing resources to Harvester, PanDA's new workload submission engine, as a critical milestone for LHC Run 3 and 4. This contribution will focus on the Grid migration to Harvester. We have built a redundant architecture based on CERN IT's common offerings (e.g. Openstack Virtual Machines and Database on Demand) to run the necessary Harvester and HTCondor services, capable of sustaining the load of O(1M) workers on the Grid per day. We have reviewed the ATLAS Grid region by region and moved as much possible away from blind worker submission, where multiple queues (e.g. single core, multi core, high memory) compete for resources on a site. Instead we have migrated towards more intelligent models that use information and priorities from the central PanDA workload management system and stream the right number of workers of each category to a unified queue while keeping late binding to the jobs. We will also describe our enhanced monitoring and analytics framework. Worker and job information is synchronized with minimal delays to a CERN IT provided ElasticSearch repository, where we can interact with dashboards to follow submission progress, discover site issues (e.g. broken Compute Elements) or spot empty workers. The result is a much more efficient usage of the Grid resources with smart, built-in monitoring of resources.

[*] Corresponding author: barreiro [at] uta [dot] edu

[†] Copyright 2020 CERN for the benefit of the ATLAS Collaboration. Reproduction of this article or parts of it is allowed as specified in the CC-BY-4.0 license.

1 Introduction

The Worldwide LHC Computing Grid (WLCG) [1] is a highly heterogeneous federation of computing sites with different middleware and increasingly special resources, such as Cloud or High Performance Computing (HPC) resources. PanDA [2] is the Workload Management System (WMS) for the ATLAS experiment [3] at the Large Hadron Collider (LHC), managing all production and user jobs across the WLCG centers associated with the experiment. In order to exploit resources, PanDA is based on the Pilot paradigm [4], where Pilot jobs are submitted to the batch systems at sites. The Pilots retrieve the real payload from PanDA server and execute it.

Over the years numerous Pilot submission systems have been developed, frequently specializing on a certain subset of resources and having independent code bases. The Harvester project was born as an attempt to provide a universal Pilot submission system. At the same time, in the case of Grid resources, some improvements were needed to increase the stability and usage efficiency through a tighter integration with the PanDA Workload Management System allowing a more informed decision taking.

This contribution will focus on core Harvester design decisions and other significant aspects like new submission modes and monitoring. We will also show the process and results of migrating all Grid resources to Harvester.

2 Harvester design decisions

2.1 Lightweight vs High Performance execution modes

In order to be a universal service for any type of resource, Harvester needs to provide certain flexibility in terms of resource consumption and dependencies. In the case of HPCs, Harvester frequently needs to run on edge nodes with strict restrictions on installation and memory/CPU footprint of the application. In this case Harvester provides a lightweight mode, using a SQLite database and limiting the number of internally managed threads.

On the contrary for the Grid, central Harvester instances manage very large, costly infrastructures that need to be kept fully utilized. There are no important installation restrictions and it is preferable to host high performance services. In this case, Harvester typically uses a MySQL/MariaDB database and multiple Harvester processes can be started through frameworks like uWSGI [5].

2.2 Fast integration of new resources

To reduce the lead time until a new resource is exploited successfully, the Harvester code follows a plug-in approach: the code is split into a common core and a set of specific plug-ins that can be configured for each resource. Typically, new resources only require the implementation of resource-specific libraries for submission, monitoring and cleaning up workers that have finished. For most of the cases, these are python modules with lengths of a few hundred lines of code.

Up to date, plug-ins have been developed for HTCondor [6], ARC CE [7], Google Compute Engine [8], Kubernetes [9], SAGA [10], PBS [11], Cobalt [12] and Slurm [13].

Data management plugins can be also developed to handle input and output files. This is often required for HPCs, where the worker nodes have no external connectivity. However this is generally not required for the Grid, since the Pilot handles the data management.

2.3 Queue unification

ATLAS submits Pilots specifying the number of cores and memory for the jobs. In the past, sites provided multiple separate queues per job type:

- Analysis: usually single core payloads running with a non-production proxy security role
- Production: several different payloads summarized in Table 1.

Table 1. Requirements for the various production workloads

	CPUs	**Memory**
Single core	1	2 GB
Multi core	Depends on site, usually 8	2 GB/core
Single core, high memory	1	Depends on site, >2 GB
Multi core, high memory	Depends on site, usually 8	Depends on site, >2 GB/core

An average site for ATLAS would have an Analysis queue, a single core and a multi-core production queue. Larger sites would also provide high memory queues. Pilots would be submitted to these queues independently, causing random competition for the slots and making it impossible to establish any control over the ratios.

In order to follow the Global Shares [14] priorities of ATLAS, it is more desirable to unify all queues at a site into one single queue that can accept Pilots with different sizes. These unified queues can be managed using either the Pull mode or the new Pull UPS mode (see next subsection).

2.4 Submission modes supported by Harvester

Harvester supports the classical push and pull workflows, and has extended the pull workflow for unified queues as follows:

- **Push (early binding)**: the worker submitted to the batch queue is already assigned to a particular job and requests the CPU, memory, and potentially other requirements, of the specific job. The push workflow works natively with unified queues. The disadvantage is the early binding: once the worker is submitted, you can't control the queue time. If the queue time is long, a high priority job is stuck in the queue. Also, during the queueing time, a more important job might have appeared and needs to wait until the queue of previously submitted jobs has cleared.
- **Pull (late binding)**: To circumvent the drawbacks of the push workflow, the pull workflow submits workers without any pre-assigned job. The advantage is that the WMS can choose the most important job right at the time the worker starts running. However, all workers submitted to the queue are the same and thus this mode does not support natively on unified queues. A dumb pilot submitter and multiple queues with different requirements on the same site causes competition between them and ATLAS can't control the ratio of workers across the queues.
- **Pull UPS (Unified Pilot Streaming):** This mode is an extension of the traditional pull mode to support unified queues. All types of jobs are queued together in the same queue and it's the WMS system deciding the fraction of workers of each type to submit, depending on the global priorities. The WMS decision is published to Harvester as a command.

3 Harvester monitoring

3.1 Worker monitoring

Harvester reports the worker information up to the WMS, in our case PanDA. PanDA stores the worker information from the different Harvesters in one combined central Oracle table and also mirrors it to an ElasticSearch repository managed by CERN IT [15]. Our choice in monitoring follows the general trend to build dashboards on top of ElasticSearch, because of its speed and flexibility. We have built dashboards in Kibana [16] for expert users and more user-friendly dashboards in Grafana [17] for widespread use by site admins and shifters.

The dashboards allow the user to interact with the Harvester data at a detailed worker level and at high level overviews:

- The dashboards show a table with detailed worker information, which provide links to the logs and to the job submission file, to provide easy debugging of worker failures.
- The dashboards also provide several plots to see the submission evolution broken down by different criteria (Harvester instance, resource type, state, etc.). There are also plots to see the worker distribution across the Computing Elements of a site.

3.2 Service monitoring

Our service monitoring collects important information about the Harvester instances:

- Disk, CPU and memory usage
- Worker submission, update and completion rates

These metrics allow to easily identify when an instance is in trouble or misbehaving. We have an alerting system in place that sends a mail to the central Harvester team in case a metric has exceeded a threshold. The alerting system has proven very useful and warns us quickly about issues. This alerting system is critical in times of low operational manpower and shifters in ATLAS.

3.3 Site monitoring

We are also implementing monitoring to identify broken sites. There are views in place grouping broken sites and broken Computing Elements, based on high ratios of workers in bad states (failed or cancelled submission). We show also the error messages happening on those sites.

We are working on a monitoring to identify inactive sites, where the submission rate is lower than what would be expected. There can be different reasons for this, including misbehavior of Harvester submission rate calculation for the site. In the future we would like to automate actions, like issuing a reset of the worker submission calculation or submitting tickets to the site.

4 Migration of the Grid to Harvester

4.1 Central infrastructure

We have setup a central high-performance Harvester infrastructure for the whole Grid as summarized in Figure 1. It is based on CERN IT provided services. The databases are

hosted by the DB on Demand [18] project, while the servers are virtual machines from the OpenStack service.
We have distributed the Grid sites across three Harvester instances:

- Harvester A: US, CERN Tier 0, France, Russia, Canada, Netherlands
- Harvester B: CERN, Germany, UK, Taiwan, Italy, Spain
- Harvester + ARC Control Tower (ACT): ARC CE sites

Each instance consists of two active machines and one shared database for each instance. The two machines provide redundancy and are in different zones of the CERN computing center. If one machine is unavailable, the other machine can take over the full load.
Non-ARC computing elements rely on HTCondor as interface. There are five load-balanced HTCondor machines for the whole grid.

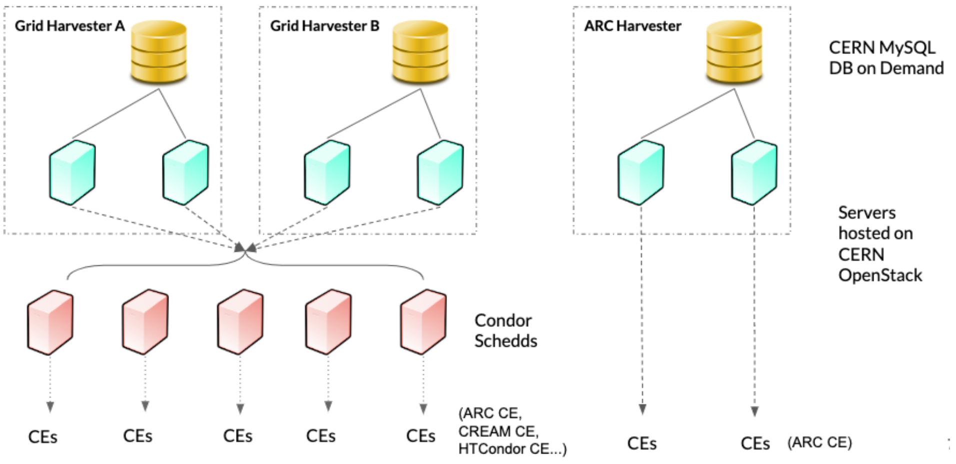

Fig. 1. Harvester infrastructure overview

4.2 Migration process

The whole grid was migrated from June 2018 to March 2019 (see Figure 2). During the migration process we watched stability and scaling of the services, improved the efficiency of the HTCondor interfaces, simplified the configuration of queues through the ATLAS Grid Information System (AGIS) [19] and improved some interactions with the database.

During the first period we migrated production queues to Harvester. Although production jobs occupy the better part of the grid slots, they are rather easy to handle from a Harvester load point of view: they run for multiple hours and typically occupy multiple cores. During the migration a major effort was devoted to unify production queues to the pull UPS mode explained in section 2.4.

Once the infrastructure was proven to be scalable and stable, we also migrated analysis queues to Harvester. Analysis jobs represent less than 20% of the grid slots, but they are short and run on a single core. From a Harvester load point of view, they are heavier and require a much higher worker submission rate.

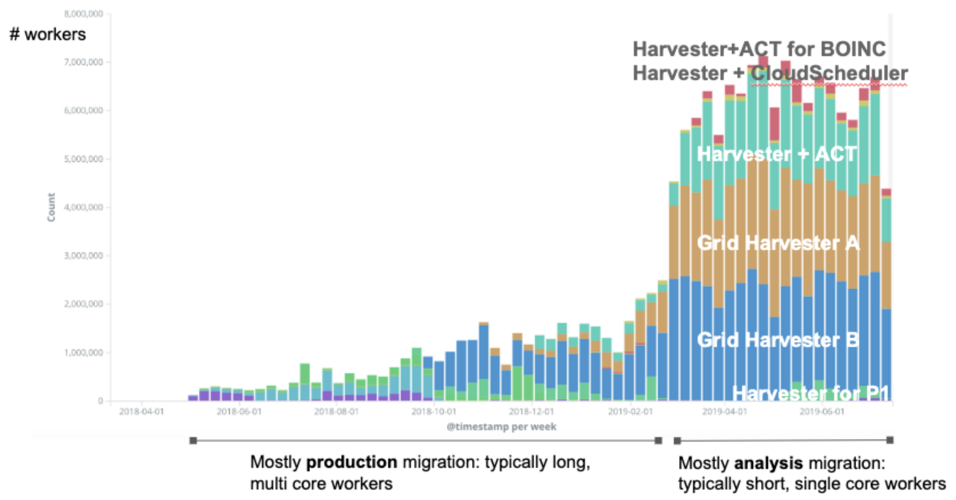

Fig. 2. Overview of the Grid migration progress to Harvester, in number of workers. This histogram is extracted from our Harvester worker monitoring.

5 Results

Harvester is a project that took 2 years from initial discussion to full roll out. It is contributing to a better usage of the ATLAS Grid. During the roll out of Harvester, we observed a significant increase of job slots (see Figure 3). While there are multiple potential contributions happening in parallel (e.g. 3-4% of pledge increase or increased usage of the ATLAS Tier 0), we believe that Harvester has made a significant contribution to the increase for two reasons:

- Harvester is more aggressive than previous Pilot factories in worker submission and competes better on shared sites across multiple experiments.
- Through the queue unification and the pull UPS submission, the worker submission is also more intelligent. It avoids uninformed competition between ATLAS queues at the same site and follows better the ATLAS job priorities.

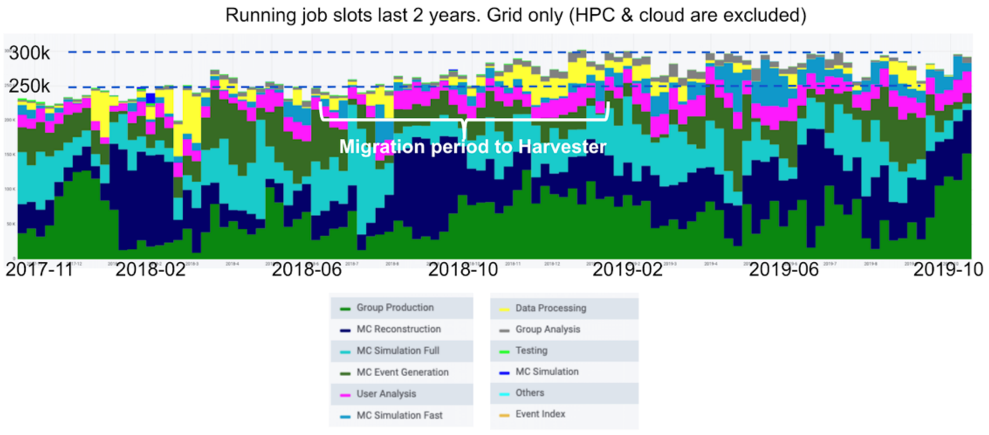

Fig. 3. Running job slots on Grid resources before and after the migration to Harvester

Unified queues for production are working very well and we are currently working on further unifying production and analysis queues, so that a standard Grid site only needs to provide one queue for all job types.

Lastly, we want to enhance automation of issues on the Grid to reduce the need of human operational effort.

Acknowledgements

This research was enabled in part by support provided by National Science Foundation (www.nsf.gov) and US Department of Energy (www.energy.gov/).

References

1. LHC Computing Grid: Technical Design Report, document LCG-TDR-001, CERN-LHCC-2005-024 (The LCG TDR Editorial Board) (2005)
2. T. Maeno et al. J. Phys. Conf. Ser. **898** 052002 (2017)
3. ATLAS Collaboration *J. Inst*. **3** S08003 (2008)
4. P. Nilsson et al. J. Phys. Conf. Ser. **513** 032071 (2014)
5. uWSGI https://uwsgi-docs.readthedocs.io/en/latest/
6. HTCondor https://research.cs.wisc.edu/htcondor/
7. ARC CE http://www.nordugrid.org/arc/ce/
8. Google Compute Engine https://cloud.google.com/compute/
9. Kubernetes https://kubernetes.io/
10. SAGA https://radical-cybertools.github.io/
11. PBS https://www.nas.nasa.gov/hecc/support/kb/121/
12. Cobalt https://trac.mcs.anl.gov/projects/cobalt
13. Slurm https://www.slurm.schedmd.com/
14. F. H. Barreiro Megino et al. EPJ Web Conf., **214** 03025 (2019)
15. P. Saiz et al. EPJ Web Conf., (to be published)
16. Kibana https://www.elastic.co/kibana
17. Grafana https://grafana.com/
18. R. Aparicio et al. J. Phys. Conf. Ser. **664** 042021 (2015)
19. A. Anisenkov et al. J. Phys. Conf. Ser. **664** 062001 (2015)