Converging High-Throughput and High-Performance Computing: A Case Study

Author 1
Institution 1
email1

Author 2
Institution 2
email2

Author 3
Institution 3
email3

Author 4
Institution 4
email4

Author 5
Institution 5
email5

Author 6
Institution 6
email6

ABSTRACT

Experiments at the Large Hadron Collider (LHC) face unprecedented computing challenges. Thousands of physicists analyze exabytes of data every year, using billions of computing hours on hundreds of computing sites worldwide. PanDA (Production and Distributed Analysis) is a workload management system (WMS) developed to meet the scale and complexity of LHC distributed computing for the ATLAS experiment. PanDA is the first exascale workload management system in HEP, executing millions of computing jobs per day, and processing over an exabyte of data in 2016. In this paper, we introduce the design and implementation of PanDA, describing its deployment on Titan, the third biggest supercomputer in the world. We analyze scalability, reliability and performance of PanDA on Titan, highlighting the challenges addressed by its architecture and implementation. We present preliminary results of experiments performed with the Next Generation Executer, a prototype we developed to meet new challenges of scale and resource heterogeneity.

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference format:
Author 1, Author 2, Author 3, Author 4, Author 5, and Author 6. 2017. Converging High-Throughput and High-Performance Computing: A Case Study. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, USA, Nov 2017 (SC2017)*, 15 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Large Hadron Collider (LHC) was created to explore the fundamental properties of matter for the next decades. Since LHC start-up in 2009, multiple experiments at LHC have collected and distributed hundreds of petabytes of data worldwide to hundreds of computer centers. Thousands of physicists analyze petascale data volumes daily. The detection of the Higgs Boson speaks to the success of the detector and experiment design, as well as the sophistication of computing systems devised to analyze the data.

Historically the computing systems consisted of the federation of a hundreds to thousands of distributed resources — ranging in scale from small to mid-size resource [13]. Although the workloads to be executed are independent, the management of the distribution of extremely- large workloads across many heterogeneous resources to ensure the effective utilization of resources and efficient execution of workloads presents non- trivial challenges.

Many software solutions have been developed in response to these challenges. One of the LHC experiments, the CMS experiment devised a solution based around the HTCondor [**?**] software ecosystem. The ATLAS [1] experiment, utilizes the Production and Distributed Analysis (PanDA) workload management system [17] (WMS) for distributed data processing and analysis. The CMS and ATLAS experiments represent arguably the largest production grade distributed computing solutions and have symbolized the paradigm of *high-throughput computing* viz., the effective execution of many independent workloads.

As the LHC prepares for the high-luminosity era (Run 3 in $\approx$ 2022), it is anticipated that the data volumes that will need analyzing will increase by 10X compared to the current phase (Run 2). The data will be larger in volume but will also require heterogeneous computational processes. In spite of the impressive scale of the ATLAS distributed computing system, demand for computing systems will significantly outstrip supply (availability).

There are multiple levels at which this problem needs to be addressed urgently, e.g., the utilization of emerging parallel architectures (e.g., platforms), algorithmic and advances in analytical methods (e.g., use of Machine Learning) and the ability to exploit different platforms (e.g., clouds and supercomputers).

This paper is a case study of how the ATLAS experiment has ”broken free” of the traditional computational approach of high-throughput computing on distributed resources to embrace new platforms, in particular high-performance computers. Specifically, we discuss the experience of integrating the PanDA workload management system with Titan — a DOE leadership computing facility and scaled to analyze up to ***SJ: XX% of the ATLAS experiment workload. We present the design of PanDA and show how localized substitution to use well established pilot-concepts allow enhanced support for heterogeneous workloads (such as molecular dynamics) and advanced execution modes.

This state-of-practice paper provides multiple contributions. (i) Documents the many design and operational considerations that have been taken to support the sustained, scalable and production usage of Titan for historically high-throughput workloads, (ii) Extensions to PanDA to support non-traditional heterogeneous workloads and execution modes, and (iii) As the community looks forward to designing the next generation of online analytical platforms [?] the lessons learnt from our project provide some guidance for how current and future experimental and observational systems can be integrated with supercomputers in production.

2 PANDA OVERVIEW

PanDA is a Workload Management System (WMS) designed to support the execution of distributed workloads and workflows via pilots. WMS is middleware for discovering and selecting resources, submitting tasks of workloads and workflows, and monitoring their execution [20]. Pilot is an abstraction that enables multi-level scheduling by decoupling resource acquisition from tasks scheduling [31]. When implemented, a pilot is scheduled on a site and, once active, tasks are scheduled to the pilot, not to the site's scheduler.

Pilot-enabled WMS enable high throughput of tasks execution while supporting interoperability across multiple sites. This is particularly relevant for LHC experiments, where millions of tasks are executed across multiple sites every month, analyzing and producing petabytes of data. The design of PanDA WMS started in 2005 and its implementation went into production for the LHC Run 1 on 2009. PanDA was then extended with new subsystems to be deployed on Run 2, on 2015.

2.1 Design

PanDA's application model assumes tasks, workloads and workflows. Tasks represent a set of operations performed on a set of events that are stored in one or more input files. Tasks are decomposed into jobs, where each job represents the task's set of operations and a partition of the task's events. Since 2005, a certain amount of parallelism has been progressively introduced for job execution [10] but, so far, no MPI jobs have been considered for production. Jobs are supposed to be relatively self-contained, capable of setting up their own execution environment or having a minimal set of common dependences.

PanDA's usage model is based on multitenancy of resources and the support of at least two types of HEP users: individual researchers and groups executing so called 'production' workflows. Users are free to submit tasks and workflows to the PanDA WMS at any point in time, directly or via dedicated application frameworks. Consistently, PanDA's security model is based on separation between authentication, authorization and accounting for both single users and group of users. Both authentication and authorization are based on digital certificates and the X.509 standard and on the virtual organization (VO) abstraction.

Currently, PanDA's execution model is based on five main abstractions: task, job, queue, pilot, and event. Both tasks and jobs are assumed to have attributes and states and to be queued into a global queue for execution. Prioritization and binding of jobs are assumed to depend on the attributes of each job. Pilot is used to indicate the abstraction of resource capabilities. Each job is thought to be bound to one pilot and executed on the site where the pilot has been instantiated.

In PanDA's data model, each event refers to the recorded or simulated measurement of a physical event. One or more events can be packaged into files or other data containers. As with jobs, data have both attributes and states, and some of the attributes are shared between events and jobs. Raw, reconstruction, and simulation data are assumed to be distributed across multiple storage facilities and managed by the ATLAS Distributed Data Management (DDM) [14]. When necessary, datasets required by each job are assumed to be replicated over the network, both for input and output data.

PanDA's design supports provenance and traceability for both jobs and data. Attributes enable provenance by linking jobs and data items, providing information like ownership or project affiliation. States enable traceability by providing information about the stage of the execution in which each job or data item is or has been. Some attributes are assumed to be immutable across execution and jobs and data items are assumed to be always in one and only one state.

2.2 Implementation and Execution

The implementation of PanDA consists of several interconnected subsystems, most of them built from off-the-shelf and Open Source components. Subsystems communicate via dedicated API or HTTP messaging, and each subsystem is implemented by one or more modules. Databases are used to store eventful entities like tasks, jobs and events, and to store information about sites, resources, logs, and accounting.

Currently, PanDA's architecture has five main subsystems: PanDA Server [19], AutoPyFactory [7], PanDA Pilot [21], JEDI [5], and PanDA Monitoring [16]. Other subsystems are used by some of ATLAS workflows (e.g., PanDA Event Service [8]) but, at the moment, they are not relevant to understand how PanDA has been ported to supercomputers. For a full list of subsystems see Ref. [28]. Figure 1 shows a diagrammatic representation of PanDA main subsystems, highlighting the execution process of tasks while omitting monitoring details to improve readability.

The relation between tasks and jobs can be one-to-one or one-to-many, and the conversion between the two can by static or dynamic. During the LHC Run 1, PanDA required users or applications to perform a static conversion between tasks and jobs: tasks were described as a set of jobs with a fixed number of events and then submitted to the PanDA Server.

This approach introduced inefficiency both with usability and resource utilization [6]. Ideally, users should not have to reason in terms of jobs: Users conceive analyses in terms of one or more, possibly related tasks; the 'job' abstraction is required by the execution middleware, i.e. PanDA. Further, a static partitioning of tasks into jobs does not take into account the heterogeneity and dynamicity of the resources of the pilots on which each job will be executed.

Another problem of static job sizing is that PanDA instantiates pilots on sites with different type of resources and different models of availability of those resources. An optimal sizing of each job should take into account these properties. For example, sites may offer cores with different speed, networking with different amount of bandwidth, and resources could be guaranteed to be available for

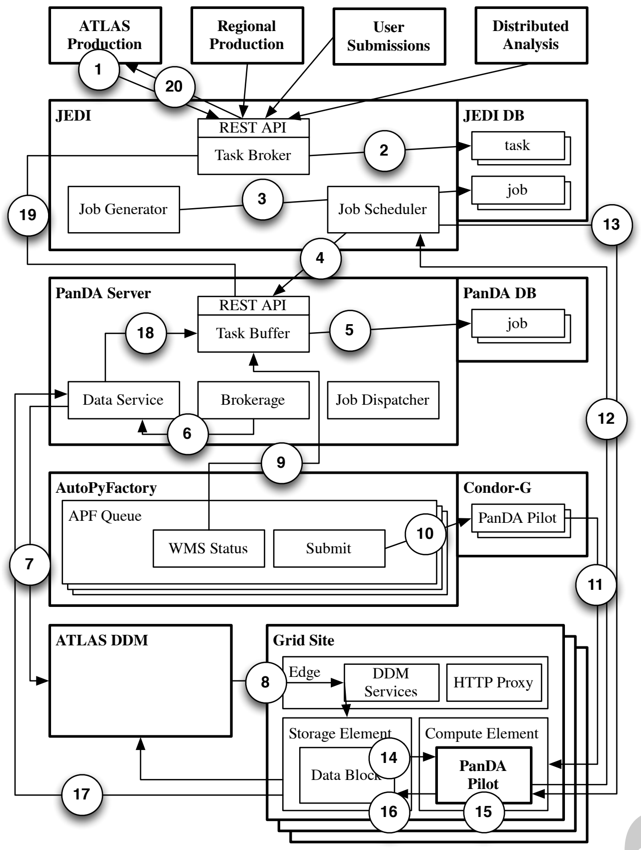

Figure 1: PanDA architecture with the subsystems and components relevant to the integration of PanDA with supercomputers. Numbers indicates the execution process based on JEDI: From the submission of tasks (1) to the retrieval of their output (20). The monitoring subsystem, the architectural details of PanDA Pilot and the communication among subsystem's components are abstracted to improve clarity.

a defined amount of time or could disappear at any point in time as with opportunistic models of resource provision.

JEDI was deployed for the LHC Run 2 to address these inefficiencies. Users or the application layer submits tasks descriptions to JEDI (Fig. 1:1) that stores them into a queue implemented by a database (Fig. 1:2). Tasks are partitioned into jobs of different size, depending on both static and dynamic information about available resources (Fig. 1:3). Jobs are bound to sites with resources that best match jobs' requirements, and submitted to the PanDA Server for execution (Fig. 1:4).

Once submitted to the PanDA Server, jobs are stored by the Task Buffer component into a global queue implemented as a database (Fig. 1:5). When jobs are submitted directly to the PanDA Server, the Brokerage component is used to bind jobs to available sites, depending on static information about the resources available for each site. Jobs submitted by JEDI are already bound to sites so no further brokerage is needed. Once jobs are bound to sites, the Brokerage module communicates to the Data Service module what data sets need to be made available on what site (Fig. 1:6). The Data Service communicates these requirements to the ATLAS DDM (Fig. 1:7) that, when needed, takes care of aggregating files into datasets and containers, replicating them on the required sites (Fig. 1:8).

The AutoPyFactory subsystem communicates with the Task Buffer component of the PanDA Server, acquiring information about jobs that are ready for execution on specific (type of) sites (Fig. 1:9). On the basis of this information, AutoPyFactory defines a set of suitable PanDA Pilots and concurrently submits them to a Condor-G broker (Fig. 1:10). This broker submits these pilots wrapped as jobs or VMs to the required sites (Fig. 1:11).

When a PanDA Pilot becomes available on a site, it calls the Job Dispatcher module of the PanDA Server, requesting for a job to execute (Fig. 1:12). The Job Dispatcher communicates with the Task Buffer, requesting for a job that is bound to the site of that pilot and ready to be executed. Task Buffer checks the global queue (i.e., the PanDA database) and if such a job is available, it passes the job handler to the Job Dispatcher. The Job Dispatcher dispatches the job to the PanDA Pilot (Fig. 1:13).

Upon receiving a job, a PanDA Pilot starts a monitoring process and forks a subprocess for the execution of the job's payload. The job is setup, input data are transferred from the designated staging in location (Fig. 1:14), and the job's payload is executed (Fig. 1:15). Once completed, output is transferred to the staging out location (Fig. 1:16).

The Data Service module of the PanDA Server tracks and collects the output generated by each job (Fig. 1:17), updating jobs' attributes via the Task Buffer module (Fig. 1:18). When the output of all the jobs of a task are retrieved, it is made available to the user via PanDA Server. When a task is submitted to JEDI, task is marked as done (Fig. 1:19) and the result of its execution is made available to the user by JEDI (Fig. 1:20).

3 DEPLOYING PANDA ON A LEADERSHIP-SCALE SYSTEM

The next LHC run for taking data (Run 3) will require more resources than the Worldwide LHC Computing Grid (WLCG) can provide. Currently, PanDA WMS uses more than 100,000 cores at over 100 Grid sites, with a peak performance of 0.3 petaFLOPS. This capacity will be sufficient for the planned analysis and data processing, but it will be insufficient for the Monte Carlo production workflow and any extra activity. To alleviate these challenges, ATLAS is engaged in a program to expand the current computing model to include additional resources such as the opportunistic use of supercomputers as well as commercial and academic clouds.

3.1 Use of Supercomputers with PanDA

Modern supercomputers have been designed mainly to support parallel computation that requires runtime communication. Job execution is parallelized across multiple cores, each core calculating a small part of the problem and communicating with other cores via a message passing interface (MPI). Accordingly, supercomputers have large number of worker nodes, connected through a high-speed, low-latency dedicated network. Each worker node has multicore CPUs, usually augmented with parallel Graphics Processing Units (GPUs) or other types of specialized coprocessors.

PanDA WMS has been designed to support distributed Grid computing. Executing ATLAS workloads or workflows involves concurrent and/or sequential runs of possibly large amount of jobs, each requiring no or minimal parallelization and no runtime communication. Thus, computing infrastructure like WLCG have been designed to aggregate large amount of computing resources across multiple sites. While each site may deploy runtime message-passing capabilities, usually these are not used to perform distributed computations.

There are at least two approaches to enable PanDA WMS to execute ATLAS workloads or workflows on supercomputers: (i) using the subset of resources and capabilities shared by both supercomputers and WLCG; (ii) reconciling the parallel and distributed computing paradigms by means of dedicated abstractions. The former is a pragmatic approach that enables the execution of specific workloads by prototyping single-point solutions. The latter is a principled approach, better suited for a production-grade solution, capable of supporting general-purpose workloads and workflows on both supercomputers and Grid infrastructures. These two approaches are not mutually exclusive: developing a single-point solution gives the opportunity to better understand the problem space, supporting the creation of abstractions for the integration of computing paradigms.

This section illustrates the design and architecture of a job broker prototyped by the PanDA team. This broker supports execution of part of the ATLAS production Monte Carlo workflow on Titan, a leadership-class supercomputer managed by the Oak Ridge Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory (ORNL). After an analysis of the results obtained and the lessons learned, the following section introduces the design and first experimental characterization of a next generation executor (NGE). NGE is designed to abstract resources and capabilities, enabling the concurrent execution of both parallel and distributed computing on generic high performance computing (HPC) machines.

3.2 Interfacing PanDA with Titan

The Titan supercomputer, current number three on the Top 500 list [12], is a Cray XK7 system with 18,688 worker nodes and a total of 299,008 CPU cores. Each worker node has an AMD Opteron 6274 16-core CPU, a Nvidia Tesla K20X GPU, 32 GB of RAM and no local storage, though a 16 GB RAM disk can be set up. Work nodes use Cray's Gemini interconnect for inter-node MPI messaging. Titan is served by the Spider II [22], a Lustre filesystem with 32 PB of disk storage, and by a 29 PB HPSS tape storage system. Titan's worker nodes run Compute Node Linux, a run time environment based on SUSE Linux Enterprise Server.

Titan's users submit jobs to Titan's PBS scheduler by logging into login or data transfer nodes (DTNs). Titan's authentication and authorization model is based on two-factor authentication with a RSA SecurID key, generated every 30 seconds. Login nodes and DTNs have out/inbound wide area network connectivity while worker nodes have only local network access. Fair-share and allocation policies are in place both for the PBS batch system and shared file systems.

Titan's architecture, configuration and policies poses several challenges to the integration with PanDA. The deployment model of PanDA Pilot is unfeasible on Titan: PanDA Pilot requires to contact the Job Dispatcher of the PanDA Server to pull jobs to execute but this is not possible on Titan because worker nodes do not offer outbound network connectivity. Further, Titan does not support PanDA's security model based on certificates and virtual organizations, making the PanDA's approach to identity management also unfeasible. While DTNs offer wide area network data transfer, an integration with ATLAS DDM is beyond the functional and administrative scope of the current prototyping phase. Finally, the specific characteristics of the execution environment, especially the absence of local storage on the worker nodes and modules tailored to Compute Node Linux, require reengineering of ATLAS application frameworks.

Currently, very few HEP applications could benefit from Titan's GPUs but some computationally-intensive and non memory-intensive tasks of ATLAS workflows can be offloaded from the Grid to Titan's large amount of cores. Further, when HEP tasks can be partitioned to independent jobs, Titan worker nodes can be used to execute up to 16 concurrent payloads, one for each available core. Given these constraints and challenges, the type of task most suitable for execution at the moment on Titan is Monte Carlo detector simulation. This type of task is mostly computational-intensive, requiring less than 2GB of RAM at runtime and with small input data requirements. Detector simulation tasks in ATLAS account for $\approx$ 60% of all the jobs on WLCG, making them a primary candidate for offloading.

Detector simulation is part of the ATLAS production Monte Carlo (MC) workflow (also known as MC production chain) [11, 24, 25]. The MC workflow consists of four main stages: event generation, detector simulation, digitization, and reconstruction. Event generation creates sets of particle four-momenta via different generators, e.g., PYTHIA [27], HERWIG [9] and many others. The detector simulator is called Geant4 [3] and simulates the interaction of these particles with the sensitive material of the ATLAS detector. Each interaction creates a so-called hit and all hits are collected and passed on for digitalization where hits are further process to mimic the readout of the detector. Finally, reconstruction operates local pattern recognition, creating high-level objects like particles and jets.

3.3 PanDA Broker on Titan

The lack of wide area network connectivity on Titan's worker nodes is the most relevant challenge for integrating PanDA WMS and Titan. Without connectivity, Panda Pilots cannot be scheduled on worker nodes because they would not be able to communicate with PanDA Server and therefore pull and execute jobs. This makes impossible to port PanDA Pilot to Titan while maintaining the defining feature of the pilot abstraction: decoupling resource acquisition from workload execution via multi-stage scheduling.

The unavailability of pilots is a potential drawback when executing distributed workloads like MC detector simulation. Pilots are used to increase the throughput of distributed workloads: while pilots have to wait in the supercomputer's queue, once scheduled, they can pull and execute jobs independently from the system's queue. Jobs can be concurrently executed on every core available to the pilot, and multiple generations of concurrent executions can be performed until the pilot's walltime is exhausted. This is particularly relevant for machines like Titan where queue policies privilege parallel jobs on the base of the number of worker nodes they request: the higher the number of nodes, the shorter the amount of queue time (modulo fair-share and allocation policies).

Titan's backfill functionality offers the opportunity to avoid the overhead of queue wait times without using pilot abstraction. Backfill availability is the number of worker nodes that cannot be used for a certain amount of time by any of the jobs already queued on Titan: All queued jobs are either too large or their walltime is too long. At any point in time, Titan's Moab scheduler can be queried for backfill availability. Based upon the result of this query, a job can be shaped to request no more than the backfill availability. As such, when submitted this job is usually scheduled immediately, spending almost no time in the queue.

Compared to pilots, backfill has the disadvantage of limiting the amount of work nodes that can be requested. Pilots are normal jobs: they can request as many worker nodes for as much time as a queue can offer. On the contrary, jobs sized on the basis of backfill information availability ***SJ: should be resource availability or backfill queue information? ***MT: AFAIH, there is no dedicated queue to backfill depend on the number of worker nodes that cannot be given to any other job in the Titan's queue.

Usually, backfill availability is a fraction of the total capacity of the queue but the size of Titan mitigates this limitation. Every year, about 10% of Titan's capacity remains unused, corresponding to an average of 30,000 unused cores. This equals approximately 270M core hours per year, roughly 30% of the overall capacity of WLCG.

***MT: Please feel free to add details about backfill as needed. ***MT: Please note: I know the following departs for the current name given to the PanDA subsystem installed on Titan. I think there is a good reason to call it a broker instead of a pilot, and I think I explained it in the previous paragraphs. Please take this just as a suggestion, something I would like to discuss in our meeting. ***SP: It's not a pilot in conventional sense. I call it an agent. It just happened that we were able to use full PanDA pilot's code base to serve our purposes on Titan. that's why, by inertia, we still call it a pilot. ***MT: Thank you. We would have a problem calling it 'agent' as we use that term to name the pilot of NGE. Would PanDA Broker work? ***SP: OK broker it is for this paper.

Given the communication requirements of PanDA Pilots and the unused capacity of Titan, PanDA pilot was repurposed to serve as a job broker on the DTN nodes of Titan. Maintaining the core modules of PanDA Pilot and its stand-alone architecture, this prototype called 'PanDA Broker' implements functionalities to: (i) interrogate Titan about backfill availability; (ii) pull MC jobs and events; (iii) wrap the payload of ATLAS jobs into MPI scripts; (iv) submitting MPI scripts to Titan's PBS batch system and monitor their execution; and (v) staging input/output files. Backfill querying, payload wrapping, and scripts submission required a new implementation while pulling ATLAS job and events, and file staging were inherited from PanDA Pilot.

Backfill querying is performed via a dedicated Moab scheduler command while a tailored Python MPI script is used to execute the payload of ATLAS jobs. This MPI script enables the execution of

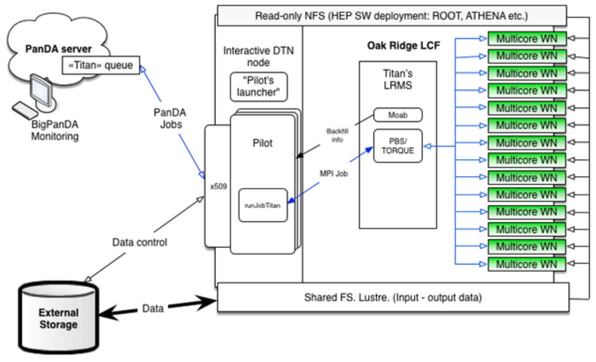

Figure 2: Schematic view of the PanDA Broker.

unmodified Grid-centric, ATLAS jobs on Titan. Typically, a MPI script is workload-specific as it sets up the execution environment for a specific payload. This involves organization of worker directories, data management, optional input parameters modification, and cleanup on exit. Upon submission, a copy of the MPI script runs on every available worker node, starting the execution of the ATLAS job's payload in a subprocess and waits until its completion.

MPI scripts are submitted to Titan's PBS batch system via RADICAL-SAGA [29], a Python module, compliant with the OGF GFD.90 SAGA specification [15]. The Simple API for Grid Applications (SAGA) offers a unified interface to diverse job schedulers and file transferring services. In this way, SAGA provides an interoperability layer that lowers the complexity of using distributed infrastructures. Behind the API façade, RADICAL-SAGA implements a adaptor architecture: each adaptor interface the SAGA API with different middleware systems and services, including the PBS batch scheduler of Titan.

The data staging capabilities of the PanDA Broker are implemented via a file system that is shared among DTNs and worker nodes. The input files with the events of the ATLAS jobs are downloaded on the shared filesystem via the ATLAS DDM service. The MPI script setup process includes making the location of these files available to the payload of the ATLAS's jobs. The PanDA Broker can locate the payload's output files on the shared filesystem and transfer them from Titan to any computing center used by ATLAS.

Once deployed on Titan, every PanDA Broker supports the execution of MC detector simulation in 8 steps: (i) PanDA Broker queries Titan's Moab scheduler about current backfill availability; (ii) the PanDA Broker queries the Job Dispatcher module of the PanDA server for ATLAS jobs that have been bound to Titan by JEDI; (iii) Upon receiving the descriptions of those jobs, PanDA Broker pulls their input files from the ATLAS DDM service to the DTN; (iv) the PanDA Broker creates an MPI script, wrapping enough ATLAS jobs' payload to fit backfill availability; (v) the PanDA Broker submits the MPI script to the Titan's PBS batch system via RADICAL-SAGA; (vi) upon execution on the worker node(s), the MPI script creates configures and executes one AthenaMP for each work node available; (vi) AthenaMP spawns 1 event simulation process on each available core (16); (vii) during execution, PanDA Broker monitors execution progress and sends PanDA Server "heart beats" for each job.; and (viii) upon completion of each simulation, the PanDA Broker locates the output on the shared filesystem, transfer it to designated computing centre, and performs cleanup.

4 ANALYSIS AND DISCUSSION

Currently, ATLAS deploys 20 instances of PanDA Broker on 4 Titan's DTNs, 5 instances per DTN. Each broker submits and manages the execution of 15 to 300 ATLAS jobs, one job for each worker node, and a theoretical maximum concurrent use of 96,000 cores. Since November 2015, PanDA Brokers have operated only in backfill mode, without a defined time allocation, and running at the lowest priority on Titan.

We evaluate the efficiency, scalability and reliability of the PanDA Brokers along two dimensions: (i) the amount of backfill availability utilized by the PanDA Brokers; and (ii) the amount of runtime spent performing detector simulations. We based these evaluations on measuring the number of cores utilized by ATLAS on Titan, the overall backfill availability, the number of detector simulations performed on all the resources available to ATLAS and on Titan, and the failure rate of these simulations. All the measurements were performed between January 2016 and February 2017, hereafter called 'experiment time window'.

4.1 Utilization and Efficiency of PanDA Broker on Titan

Figure 3 (gray bars) shows the number of core-hours used by ATLAS on Titan during the experiment time window. ATLAS consumed a total of 73.8M core-hours, for an average of ≈7M core-hours a month, with a minimum of 3.3M core-hours in April 2016 and a maximum 14.8M core-hours in February 2017.

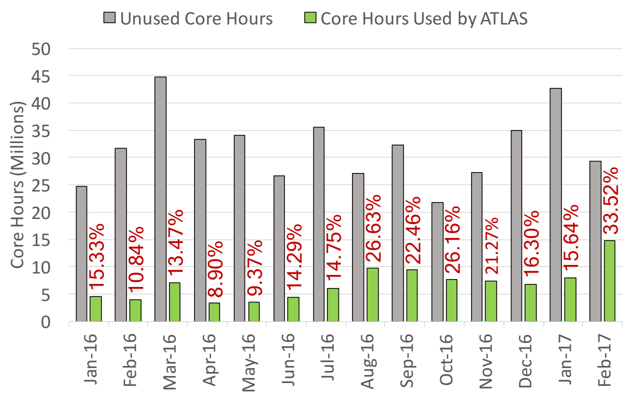

Figure 3: Gray bars: Number of unused cores of Titan's backfill availability; green bars: Number of cores of backfill availability used by ATLAS via PanDA Brokers on Titan; red labels: Efficiency of PanDA Brokers on Titan defined as percentage of backfill availability used every month of the experiment window.

Figure 3 (green bars) shows backfill utilization during the experiment time window. Efficiency (Figure 3, red labels) is defined as the fraction of Titan's cores available via backfill that was utilized by the PanDA Brokers. The brokers reached 18% average efficiency, with a minimum 8.9% efficiency on April 2016 and a maximum 33.5% efficiency on February 2017. The number of total backfill cores available was 38.1M in April 2016, and 33.1M in February 2017. This shows that the measured difference in efficiency did not depend on a comparable difference in total backfill availability.

***SJ: I would plot the efficiency on the Y2 axis for both [diagrams]. ***MT: Done but then the first diagram ended up plotting a subset of the data of the second diagram. I used only the second diagram. ***SJ: I would suggest similar style of X-axis as previous figure. Difficult to parse which is April if we are going to call out a specific month. Also consistency in style is generally good to keep cognitive burden low. ***MT: Done.

During the experiment time window, about 2.25M detector simulation jobs were completed on Titan, for a total of 225M events processed. This is equivalent to 0.9% of all the 250M detector simulations performed by ATLAS in the same period of time, and 3.5% of the 6.6B events processed by those jobs. Comparatively, Titan contributed 3.9% of the total of around 200K cores available to ATLAS on Grid, Cloud, and HPC infrastructures together. These figures confirms the relevance of supercomputers' resource contribution to the LHC Run 3, especially when accounting for the amount of unused backfill availability and the rate of improvement of PanDA efficiency.

On February 2017, PanDA Brokers used almost twice as much backfill availability than in any other month. No relevant code update was made during that period and logs indicated that the brokers were able to respond more promptly to backfill availability. This is likely due to hardware upgrades on the DTNs. The absence of continuous monitoring of those nodes does not allow to quantify bottlenecks but spot measurements of their load indicate that a faster CPU and better networking were likely responsible for the improved performance.

Investigations showed an average CPU load of 3.6% on the upgraded DTNs. As such, further hardware upgrades seem unlikely to improve significantly the performance of PanDA Brokers. Nonetheless, the current load suggests that the number of brokers per DTN could be increased. This would enable the submission of a larger number of concurrent jobs to Titan's PBS queue, allowing for PanDA to consume a higher percentage of backfill availability.

Every detector simulation executed on Titan process 100 events. This number of events is consistent with the physics of the use case and with the average duration of backfill availability. The duration of a detector simulation is a function of the number of events: the more events, the longer the simulation. Not all events take the same time to be simulated: Figure 4 shows a distribution of the simulation time of 1 event from ≈2 to ≈40 minutes, with a mean of ≈14 minutes. Considering that each worker node process up to 16 events concurrently, 100 events takes around 2 hours to process.

Figure 5 shows the distribution of backfill availability on Titan as a function of number of nodes and the time of their availability (i.e., walltime). We recorded these data by polling Titan's Moab scheduler at regular intervals during the experiment window time, while developing and deploying PanDA Brokers. The mean number

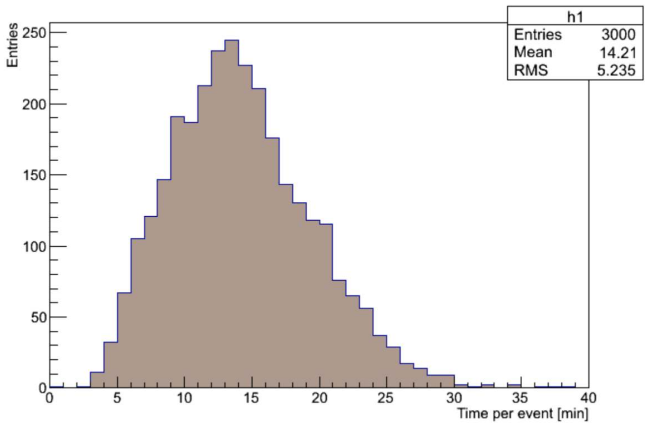

Figure 4: Distribution of the time taken by a Geant4 detector simulation to simulate one event. 3000 Events; 30 Titan worker nodes; 16 simulations per node; 100 events per node. Average time per event ≈14 min. Broad distribution from ≈2 to ≈40 minutes.

of nodes was 680, and their mean walltime was 680 minutes. Detector simulations of 100 events, enable to use down to 1 node for 1/2 of the mean walltime of backfill availability. As such, it offers a good compromise for PanDA Broker efficiency.

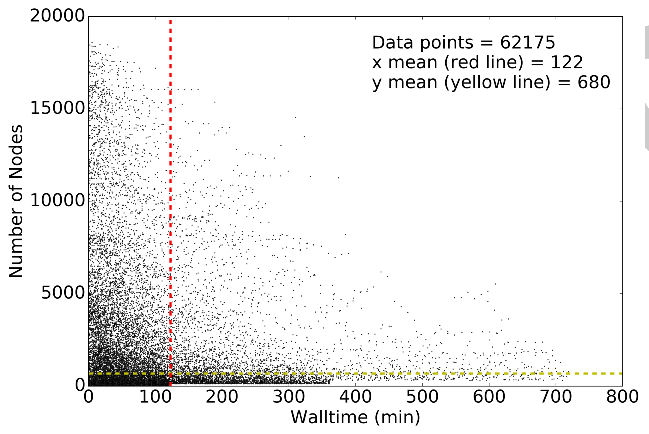

Figure 5: Backfill availability measures as walltime in minutes vs number of work nodes during the experiment window time. Availability was measured 62175 times: mean number of work nodes available 680 (yellow line); mean walltime available 122 minutes (red line).

Usually, detector simulations performed on the Grid process 10 times the number of events than those on Titan. This explains the difference between the percentage of detector simulations (9%) and of events computed by those simulations (3.9%) on Titan. PanDA Broker could fit the number of events to the walltime of the backfill availability on the base of the distribution of Figure 5. That specific number of event could then be pulled from the PanDA Event service [8] and given as input to one or more simulations. Once packaged into the MPI script submitted to titan's PBS batch system, these simulations would better fit backfill availability, increasing the efficiency of PanDA Brokers.

The transition from a homogeneous to a heterogeneous number of events per detector simulation has implications for the application layer. An even number of events across simulations makes it easier to partition, track and package events across simulations, especially when they are performed on both the Grid and Titan. A homogeneous number of events also helps to keep the size and duration of other stages of the MC workflow (§3.2) more uniform. Further analysis is needed to evaluate the trade offs between increased efficiency of resource utilization and the complexity that would be introduced at the application layer.

Currently, each PanDA Broker creates, submits, and monitor a single MPI PBS script at a time. This design is inherited from PanDA Pilot where a single process is spawn at a time to execute the payload. As a consequence, the utilization of a larger portion of Titan's backfill availability depends on the the number of concurrent PanDA Brokers instantiated on the DTNs: When all the 20 PanDA Brokers have submitted a MPI PBS Job, further backfill availability cannot be used.

Based on the current data, increasing PanDA Brokers efficiency on Titan would require around 60 concurrent brokers. Further testing and better monitoring capabilities are required to establish whether the DTN capabilities can support this load, especially when considering the volume of files staged in and out the DTNs. Alternatively, a design of a PanDA Broker capable of managing multiple MPI scripts at a time is being evaluated.

4.2 Efficiency, Scalability, and Reliability of Detector Simulation on Titan

We use two main parameters to measure the performance of the detector simulation jobs submitted to Titan: (i) the time taken to setup AthenaMP [2], the ATLAS software framework integrating the GEANT4 simulation toolkit [3]; and (ii) the distribution of the time taken by the Geant4 toolkit to simulate a certain number of events.

AthenaMP has an initialization and configuration stage. At initialization time, AthenaMP is assembled from a large number of shared libraries, depending on the type of payload that will have to be computed. Once initialized, every algorithm and service of AthenaMP is configured by a set of Python scripts. Both these operations result in a large number of read operations on the filesystem, including those required to access of small python scripts.

***MT: TODO: specify that this is work in progress (see Sergey's comment) and see how to move it to the data performance subsection Initially, all the shared libraries of AthenaMP and the python scripts for the configuration stage were stored on the Spider 2 Lustre file system. However, the I/O patterns of the initialization and configuration stages degraded the performance of the filesystem (Figure **??** ***MT: We may want to aggregate some of the diagrams about lustre's experiments here). Since Spider 2 is a center-wide file system, this resulted in performance degradation for all OLCF resources and users. ***MT: I am afraid the details about the trace are too specific given the space constraints of a SC submission. Please feel free to uncomment it if you disagree.

After careful characterization of the impact of ATLAS jobs on Lustre, the performance degradation was addressed by moving the AthenaMP distribution to a read-only NFS directory, shared among DTNs and worker nodes. This eliminated the problem of metadata contention, improving metadata read performance from ≈6,300 seconds on Lustre to ≈1,500 seconds on NFS. Further, at configuration time, the input files of each detector simulation job were stored to a ramdisk on the work nodes. This offered a marked improvement of reading time: from 1,320 seconds on Lustre to 40 seconds on NFS.

The AMD Opteron 6274 CPU used on Titan has 16 cores, divided into 8 compute units. Each compute units has 1 floating point (FP) scheduler shared between 2 cores. When using 16 cores for FP-intensive calculations, each pair of cores competes for a single FP scheduler. This creates the overhead shown in Figure 6: the mean runtime per event for 8 concurrent simulations computing 50 events is 10.8 minutes, while for 16 simulations is 14.25 minutes (consistent with Figure 4). Despite an inefficiency of almost 30%, Titan's allocation policy based on number of worker nodes used instead of number of cores does not justify the use of 1/2 of the cores available.

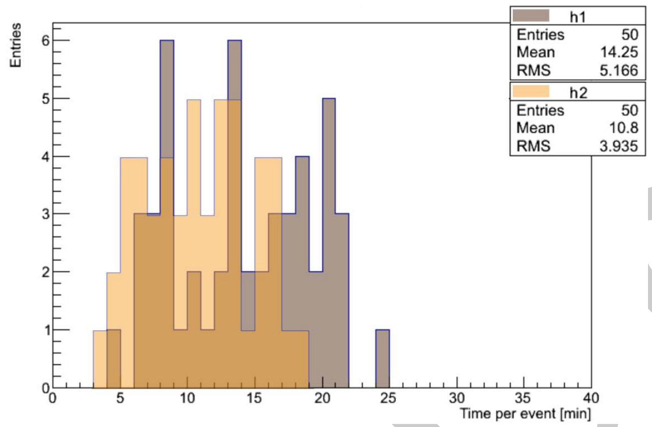

Figure 6: Comparison between distributions of the time taken by a Geant4 detector simulation to simulate one event when placing 2 simulations (h1) or 1 simulation (h2) per CPU. 2 simulation use 16 cores per node, 1 simulation 8, 1 per compute unit. 50 Events; 1 Titan worker nodes; 16 work threads per node; 100 events per node.

The performance analysis of Titan's AMD CPUs for detector simulations helps also to compare Titan and Grid site performances. Usually, Grid sites exposes resources with heterogeneous CPU architectures and a maximum of 8 (virtual) cores per worker node, while Titan's offer an homogeneous 16 cores architecture. We used the rate of events processes per minute as a measure of the efficiency of executing the same detector simulation on both Titan or Grid sites. Figure 7 compares the efficiencies of Titan to the BNL and SIGNET Grid sites, normalized for 8 cores. Effective performance per-core at Titan is ≈0.57 event per minute, roughly 1/2 of BNL and 1/3 of SIGNET performances.

The differences in performance between Titan and the BNL and SIGNET Grid sites are due to the FP scheduler competition and

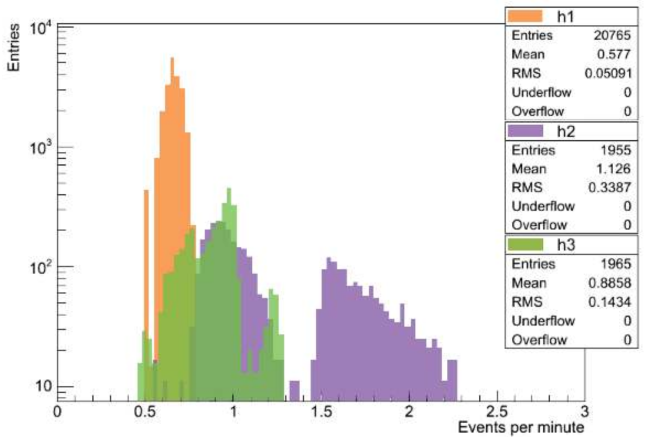

Figure 7: Comparison of the event rate for finished jobs at Titan (h1), BNL (h2) and GRIDNET (h3). Titan histogram shows 1/2 of the measured values to account for the core difference between Titan (16) and the Grid sites (8). Performance per core: BNL 1.126; SIGNET 0.885; Titan 0.577.

the availability of newer processors. The CPUs at the Grid sites have one FP scheduler per core and are on average newer than the CPU of Titan. The heterogeneity of the Grid sites' CPUs explain the higher performance variance compared to the performance consistency measured on Titan.

The current design and architecture of the PanDA Broker is proving to be as reliable as PanDA Pilot when used on the WLCG. Between Jan 2016 and Feb 2017, the overall failure rate of all the ATLAS detector simulation jobs was 14%, while the failure rate of jobs submitted to Titan was a comparable 13.6%. PanDA Brokers were responsible for around the 19% of the failures, compared to the 29% of failures produced by the JobDispatcher module of the PanDA Server, and the 13% failures produced by the Geant4 toolkit. The current failure rate of the PanDA Brokers confirms the benefits of reusing most of the code base of the PanDA Pilot for implementing the PanDA Broker. It also shows that adopting third-party libraries like RADICAL-SAGA did not have a measurable adverse effect on reliability.

4.3 PanDA I/O Impact at OLCF

To better understand the I/O impact of ATLAS PanDA project on Titan supercomputing environment we analyzed 1,175 jobs ran on the week of 10/25/2016, for a total of 174 hours. Table 1 shows the overall statistical breakdown of the observed file I/O impact of ATLAS at OLCF.

ATLAS jobs use between 1 and 300 worker nodes, and 35 on average. 75% of the ATLAS jobs consume less than 25 nodes and 92% less than 100. During the 174 hours of data collection, 6.75 ATLAS jobs were executed on average per hour, each job running for an average of 1.74 hours. Every job read less than 250 GB and wrote less than 75 GB of data and, on average, each job read 20 GB and wrote 6 GB of data.

The I/O of the ATLAS job executed on Titan show an interesting pattern. Table 1 indicates that the amount of data read per ATLAS

	Num. Nodes	Duration (s)	Read (GB)	Written (GB)	GB Read/nodes	GB Written/nodes	*open*()	*close*()
Min	1	1,932	0.01	0.03	0.00037	0.02485	1,368	349
Max	300	7,452	241.06	71.71	0.81670	0.23903	1,260,185	294,908
Average	35.66	6,280.82	20.36	6.87	0.38354	0.16794	146,459.37	34,155.74
Std. Dev.	55.33	520.99	43.90	12.33	0.19379	0.03376	231,346.55	53,799.08

Table 1: The Statistical breakdown of the I/O impact of 1,175 PanDA jobs executed at OLCF for the week of 10/25/16

worker node is less than 400 MB on average, while the amount of data written per node is less than 170 MB on average. This correlates with our finding that ATLAS PanDA jobs are read heavy. However,

ATLAS PanDA jobs are read heavy: On average, the amount of data read per worker node is less than 400 MB, while the amount of data written is less than 170 MB. This correlates with our finding that . However, the distributions of read and written data are quite different: The read operation distribution per job shows a long tail, ranging from 12.5 GB to 250 GB, while the written amount of data has a very narrow distribution. ***MT: Should we say why?

The metadata I/O breakdown shows that ATLAS jobs yield 23 file *open*() operations per second (not including file *stat*() operations) and 5 file *close*() operations per second, with similar distributions. On average, the maximum number of file *open*() operations per job is ≈170/s and the maximum number of file *close*() operations is ≈39/s. For the 1,175 ATLAS PanDA jobs observed, the total number of file *open*() operations is 172,089,760 and the total number of file *close*() operations is 40,132,992. The difference between these two values is still under investigation: One possible explanation is that ATLAS PanDA jobs don't call a file *close*() operation per every file *open*() issued.

Overall, based on our experiments with ATLAS jobs, it can be safely concluded that the file and metadata I/O load of ATLAS PanDA project on the OLCF Titan supercomputing environment and the Spider 2 file system is not detrimental to the center operations. At the current scale of the project, the overall impact of ATLAS operations on Titan is minimal.

5 PANDA: THE NEXT GENERATION EXECUTOR

***AA: I think that the names of the states should be removed. They are meaningless without the figures. I left them just in case we decide to reintroduce the figures. ***MT: Done.

***SJ: admittedly i haven't read earlier section in question carefully, but i'm a bit confused about the use of "wide area". wide area is typically used for cross data center ***MT: Every PanDA Pilot has to communicate with PanDA Server. PanDA Server runs on dedicated resources, outside OLCF. Titan's worker nodes cannot communicate with PanDA Server because their connectivity is limited to Titan's network. Do you have a suggestion for an alternative to "wide area"?

As seen in §3, PanDA Broker was deployed on DTNs because of the absence of wide area network connectivity on Titan's worker nodes. The lack of pilot capabilities impacts both the efficiency and the flexibility of PanDA's execution process. Pilots could help to improve efficiency by increasing throughput and by enabling better use of backfill utilization. Further, pilots makes easier to support heterogeneous workloads.

As discussed in §4, the absence of pilots imposes the static coupling between MPI scripts submitted to the PBS batch system and detector simulations. This makes impossible to schedule multiple generations of workload on the same PBS job: Once a number of detector simulations are packaged into a PBS job and this job is queued on Titan, no further simulations can be added to that job. New simulations have to be packaged into a new PBS job that needs to be submitted to Titan on the base of the backfill availability of that moment.

The support of multiple workload generations would enable a more efficient use of the backfill availability of walltime. Currently, when a set of simulations ends, the PBS job also ends, independent of whether more wall-time would still be available. With a pilot, more simulations could be executed so to utilize all the available wall-time, while avoiding further job packaging and submission overheads.

Multiple generations would also help with relaxing two assumptions of the current execution model: knowing the number of simulations before submitting the MPI script, and having a fixed amount of events per simulation (100 at the moment). Pilots would enable the scheduling of simulations independently from whether they were available at the moment of submitting the pilot. Further, simulations with a varying number of events could be scheduled on a pilot, depending on the amount of remaining walltime and the distribution of execution time per event, as shown in §4.2, Figure 4. This capabilities would be particularly useful to increase the PanDA Broker efficiency for availabilities with a wide delta between the number of cores and walltime.

Pilots can offer a payload-independent scheduling interface while hiding the mechanics of coordination and communication among multiple worker nodes. This could eliminate the need for packaging payload into MPI scripts within the broker, greatly simplifying the submission process. This simplification would also enable the submission of different types of payload, without having to develop a specific PBS script for each payload. The submission process would also be MPI-independent, as MPI is used for coordination among multiple worker nodes, not by the payload.

***MT: NOTE: I would speak about about backfill/not backfill in the discussion of NGE, when speaking about its generality towards resources, i.e., unified submission and scheduling process across different resources and multiple resources. Part of this generality is being able to submit to whatever batch system is supported by SAGA (including Titan's PBS and all its queues) and, in case, to multiple resources at the same time.

***MT: NOTE: NGE is not able to use backfill on Titan as we do not have specific functionalities to interrogate the Moab scheduler about backfill availability. This is why NGE should be presented as the pilot for PanDA Broker and not as an alternative to PanDA

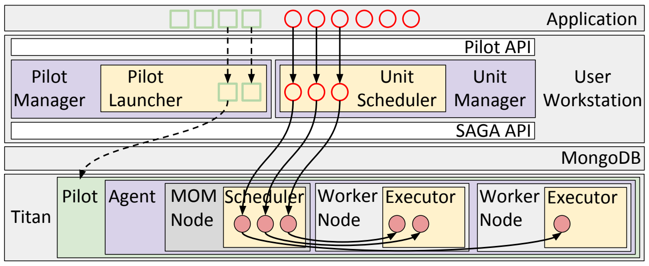

Figure 8: NGE Architecture as deployed on Titan. The Pilot-Manager and the UnitManager reside on one of Titan's DTN while the Agent is executed on Titan's worker nodes. Boxes color coding: gray for entities external to NGE, white for APIs, purple for NGE's modules, green for pilots, yellow for module's components.

Broker. Further, NGE alone would not be able to speak directly to PanDA Server.

5.1 Implementation

The implementation of pilot capabilities within the current PanDA Broker require quantification of the effective benefits that it could yield and, on the base of this analysis, a dedicated engineering effort. We developed a prototype of a pilot system capable of executing on Titan to study experimentally the quantitative and qualitative benefits that it could bring to PanDA. We called this prototype Next Generation Executor (NGE).

NGE is a runtime system designed to execute heterogeneous and dynamic workloads on diverse resources. Fig. 8 illustrates its current architecture as deployed on Titan: the two management modules represent a simplified version of the PanDA Broker while the agent module is the pilot submitted to Titan and executed on its worker nodes. The communication between PanDA Broker and PanDA Server is abstracted away as not immediately useful to evaluate the performance and capabilities of a pilot on Titan.

NGE exposes an API for the application layer to describe workloads (Fig. 8, green squares) and pilots (Fig. 8, red circles), and to instantiate a PilotManager and a UnitManager. The PilotManager submits pilots to Titan's PBS batch system via SAGA API (Fig. 8, dash arrow), as done by PanDA Broker to submit MPI scripts. Once scheduled, the pilot's Agent is bootstrapped on Titan's MOM node and worker nodes, and the UnitManager schedules units to the Agent's Scheduler (Fig. 8, solid arrow). The Agent's Scheduler schedules the units on one or more Agent's Executor for execution. The Agent's executors can manage one or more worker nodes, depending on performance evaluations.

The UnitManager and the Agent communicate via a database that is instantiated on one of Titan's DTN so to be reachable by both modules. A similar approach would be used to enable PanDA Broker to schedule pilots instead of MPI scripts. ***MT: Rewrite this after checking it with Andre.

The NGE Agent uses the *Open Run-Time Environment (ORTE)* for communication and coordination of units execution. ORTE is a spin-off from the Open-MPI project and is a critical component of the OpenMPI implementation. It was developed to support distributed high-performance computing applications operating in a heterogeneous environment. The system transparently provides support for interprocess communication, resource discovery and allocation, and process launching across a variety of platforms. ORTE provides a mechanism similar to the Pilot concept - it allows the user to create a *"dynamic virtual machine"* (DVM) that spans multiple nodes. ORTE provides libraries to enable the submission, monitoring and managing of tasks, avoiding filesystem bottlenecks and race conditions with network sockets. As a consequence, it is able to minimize the system overhead while submitting tasks.

5.2 Experiments

In this section we present experiments to determine the performances of the NGE and to show how it provides a minimal overhead while introducing new functionalities.

Experiments run instances of AthenaMP by using NGE pilots, simulating a pre-determined number of events in the ATLAS detector. ***MT: are they 100 events? ***AA: 'Not always.' We present three groups of experiments in which we test the NGE for weak scalability, weak scalability with multiple generation, and strong scalability.

We measured the execution time of the pilots and of the AthenaMP executed within them, collecting timestamps at all stages of the execution. Experiments have been performed by submitting NGE's pilots to TITAN's batch queue. Because of TITAN's policies, the turnaround time of each run of our experiments is dominated by queue time. Since we are interested only in the performances of the NGE, we removed queue time from our statistics.

All the experiments have been performed by configuring AthenaMP to use all the 16 cores of TITAN's worker nodes.

5.2.1 Weak scalability. In this experiment we run as many AthenaMP instances (also referred as tasks from now-on) as the number of nodes controlled by the pilot. Each AthenaMP simulates 100 events, requiring $\sim$ 70 minutes on average.

Tasks do not wait within the NGE Agent's queue since one node is available to each AthenaMP instance. Delays in tasks execution are consequence of three other factors: (i) the bootstrapping of the pilot on the nodes; (ii) the UnitManager, as defined in Section 5.1, that has to dispatch tasks to the agent; and (iii) the time that the agent requires to spawn all the tasks on the nodes.

We tested pilots with 250, 500, 1000 and 2000 worker nodes and 2 hours walltime. Figure 9 depicts the average pilot duration, the average execution time of AthenaMP, and the pilot's overhead as function of the pilot size.

We observe that, despite some fluctuations due to external factors (e.g., Titan's shared filesystem and the shared database used by the NGE), the average execution time of AthenaMP ranges between 4200 and 4800 seconds. We also observe that in all the cases the gap between AthenaMP execution times and the pilot durations is minimal, although it slightly increases with the pilot size. We notice that NGE's overhead does not grow linearly with the number of units.

5.2.2 Weak scalability with multiple generation . This experiment is similar to the one presented above but, in this case, we want to test also the impact of submitting multiple generations of

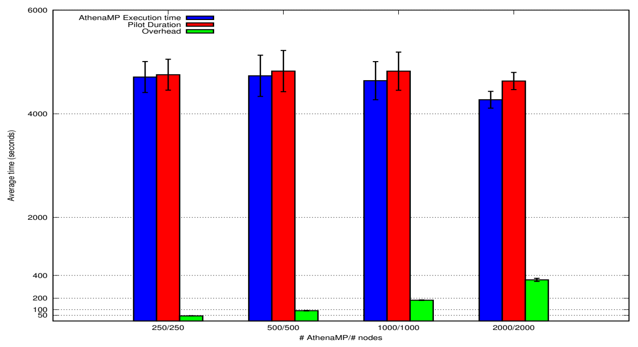

Figure 9: Weak scalability: average pilot duration, average duration of a single AthenaMP execution, and pilot's overhead for different pilot sizes (250, 500, 1000 and 2000 worker nodes).

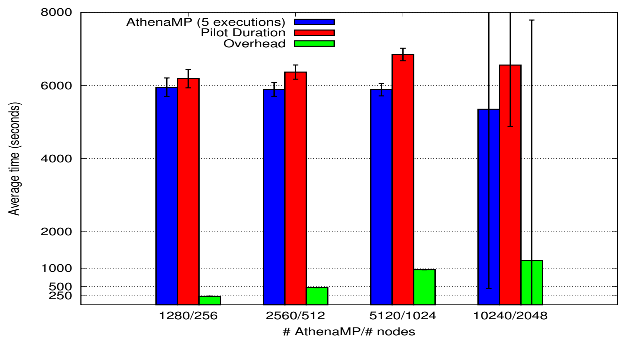

Figure 10: Weak scalability with multiple generations: average pilot duration, average duration of five sequential AthenaMP executions, and pilot's overhead for different pilot sizes (256, 512, 1024 and 2048 nodes).

AthenaMP to the same pilot. ***MT: we used 'generations' at the beginning of 6, I am assuming the reader understands the term by now. Please change if you disagree. ***AA: I agree In this way, we stress the pilot's components, as new tasks are scheduled for execution on the Agent while other tasks are still running.

***MT: we do we need to keep it below 2 hours? ***AA: You are right. It was misleadind. I Changed. Check if you like it. We performed the experiment by running five AthenaMP instances per node. Since we are focusing on the overhead generated by the scheduling and bootstrap of AthenaMP instances, we also reduced the number of events simulated by each AthenaMP to sixteen in such a way that the running time of each AthenaMP is, on average, $\sim$ 20 minutes. This choice has been made to avoid the wasting of Titan's core hours but it does not affect the aim of the experiment.

We tested pilots with 256, 512, 1024 and 2048 worker nodes and 3 ***MT: 2? hours walltime. Figure 10 depicts the average pilot duration, the average execution time of five sequential generations ***MT: generations? ***AA: Changed of AthenaMP, and the corresponding overhead. We observe that the difference between the two durations is more marked than in the previous experiments ***MT: mostly due to the increased overhead? ***AA: Well, I guess that's plain. . Despite this, we can notice that the growth of the overhead is consistent with the increment of the number of tasks per node for pilots with 256, 512 and 1024 worker nodes, and much less than linear for the pilot with 2048 worker nodes ***AA: Although that histogram is not trustworty because it is generate with only two runs.

5.2.3 *Strong scalability.* The last experiments study strong scalability by running the same amount of tasks for different pilot sizes. We used 2048 AthenaMP instances and pilots with 256, 512, 1024 and 2048 nodes. Thus, the number of AthenaMP generations is equal to eight times the size of the smallest pilot and corresponds to the size of the largest pilot. As a consequence, the number of consecutive generations of AthenaMP decreases with the pilot size by generating different dynamics within the pilots. This experiment aim to show that the pilot overhead is not affected by the concurrency level within the pilot but it depends only on the number of tasks. ***MT: Not sure I understand this sentence ***AA: My fault. It should have been seven and null or 8 and 1. It depends on how we consider the word sequential. anyway, to avoid confusion I changed the sentence. . Each AthenaMP instance simulates sixteen events as in the previous experiment.

Figure 11 shows the average pilot duration and the average execution time of possibly sequential AthenaMP instances. We notice that the difference between the pilot duration and the AthenaMP execution times is almost constant for all the pilot sizes, although the overall duration of the pilot decreases linearly with the pilot size.

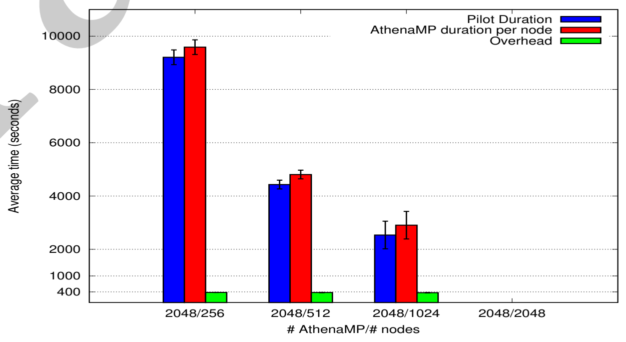

Figure 11: Strong scalability: average pilot duration, average duration of sequential AthenaMP executions, and pilot's overhead for different pilot sizes (256, 512, 1024 and 2048 nodes).

6 RELATED WORK

***SJ: In this section we should talk less about PanDA and more about other WMS – highlighting the fact that (i) there isn't a comprehensive systems (design, architecture and execution properties) discussion of other WMS so difficult to compare ***MT: Better? ***SJ: (ii) real comparison should be to the Gliden-in WMS ecosystem and use this as opportunity to narrate the reasons for the divergence. ***MT: From our meetings with Sergey: Before Run 1, ATLAS management become uncomfortable with the pace at which other WMS were being developed, loosing trust in their ability to deliver production-grade capabilities. ATLAS USA decided to go solo and developed their own solution. PanDA proven to be able to deliver the usual pilot-based WMS capabilities at the right scale. For that, it was chosen over other solutions. ***SJ: Also might this section be moved towards the rear, as this is a practice paper and not a traditional research paper? ***MT: Done.

Several pilot-enabled WMS were developed for the LHC experiments: AliEn [4] for ALICE; DIRAC [23] for LHCb; GlideinWMS [26] for CMS; and PanDA [18] for ATLAS. These systems implement similar design and architectural principles: centralization of task and resource management, and of monitoring and accounting; distribution of task execution across multiple sites; unification of the application interface; hiding of resource heterogeneity; and collection of static and sometimes dynamic information about resources.

AliEn, DIRAC, GlideinWMS and PanDA all share a similar design with two types of components: the management ones facing the application layer and centralizing the capabilities required to acquire tasks' descriptions and matching them to resource capabilities; and resource components used to acquire compute and data resources and information about their capabilities. Architecturally, the management components include one or more queue and a scheduler that coordinates with the resource modules via push/pull protocols. All resource components include middleware-specific APIs to request for resources, and a pilot capable of pulling tasks from the management modules and executing them on its resources.

AliEn, DIRAC, GlideinWMS and PanDA also have similar implementations. These WMS were initially implemented to use Grid resources, using one or more components to the Condor software ecosystem [30] and, as with GlideinWMS, contributing to its development. Accordingly, all LHC WMS implemented Grid-like authentication and authorization systems and adopted a computational model based on distributing a large amount of single/few-cores tasks across hundreds of sites ***MT: Is this true? .

All the experiments at LHC produces and process large amount of data both from actual collisions in the accelerator and from their simulations. Dedicated, multi-tiered data systems have been built to store, replicate, and distributed these data. All LHC WMS interface with these systems to move data to the sites where related compute tasks are executed or to schedule compute tasks where (large amount of) data are already stored.

7 CONCLUSION

The PanDA system was developed to meet the scale and complexity of LHC distributed computing for the ATLAS experiment. In the process, the old batch job paradigm of computing in HEP was discarded in favor of a far more flexible and scalable model. The success of PanDA at the LHC is leading to widespread adoption and testing by other experiments. PanDA is the first exascale workload management system in HEP, already operating at a million computing jobs per day, and processing over an exabyte of data in 2013. Next LHC run will pose massive computing challenges. With a doubling of the beam energy and luminosity as well as an increased need for simulates data, the data volume is expected to increase with a factor 5–6 or more. Storing and processing this amount of data is a challenge that cannot be resolved with the currently existing computing resources in ATLAS. To resolve this challenge, ATLAS is turning to commercial as well as academic Cloud services and HPCs via the PanDA system. Also the work underway is enabling the use of PanDA by new scientific collaborations and communities as a means of leveraging extreme scale computing resources with a low barrier of entry. The technology base provided by the PanDA system will enhance the usage of a variety of high-performance computing resources available to basic research.

ACKNOWLEDGEMENTS

The authors would like to thank... This research used resources of the Oak Ridge Leadership Computing Facility, supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

REFERENCES

[1] G. Aad and others. 2008. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST* 3 (2008), S08003. DOI:https://doi.org/10.1088/1748-0221/3/08/S08003

[2] Georges Aad, B Abbott, J Abdallah, AA Abdelalim, A Abdesselam, O Abdinov, B Abi, M Abolins, H Abramowicz, H Abreu, and others. 2010. The ATLAS simulation infrastructure. *The European Physical Journal C* 70, 3 (2010), 823–874.

[3] Sea Agostinelli, John Allison, K al Amako, J Apostolakis, H Araujo, P Arce, M Asai, D Axen, S Banerjee, G Barrand, and others. 2003. GEANT4—a simulation toolkit. *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506, 3 (2003), 250–303.

[4] S Bagnasco, L Betev, P Buncic, F Carminati, F Furano, A Grigoras, C Grigoras, P Mendez-Lorenzo, A J Peters, and P Saiz. 2010. The ALICE workload management system: Status before the real data taking. *J. Phys.: Conf. Ser.* 219 (2010), 062004. 6 p. https://cds.cern.ch/record/1353182

[5] Mikhail Borodin, K De, J Garcia, Dmitry Golubkov, A Klimentov, T Maeno, A Vaniachine, and others. 2015. Scaling up ATLAS production system for the LHC Run 2 and beyond: project ProdSys2. In *Journal of Physics: Conference Series*, Vol. 664. IOP Publishing, Bristol, UK, 062005. Issue 6.

[6] Mikhail Borodin, Kaushik De, Jose Garcia Navarro, Dmitry Golubkov, Alexei Klimentov, Tadashi Maeno, David South, and others. 2015. Unified System for Processing Real and Simulated Data in the ATLAS Experiment. (2015). arXiv:1508.07174

[7] J Caballero, J Hover, P Love, and GA Stewart. 2012. AutoPyFactory: a scalable flexible pilot factory implementation. In *Journal of Physics: Conference Series*, Vol. 396. IOP Publishing, Bristol, UK, 032016. Issue 3.

[8] Paolo Calafiura, K De, W Guan, T Maeno, P Nilsson, D Oleynik, S Panitkin, V Tsulaia, P Van Gemmeren, and T Wenaus. 2015. The ATLAS Event Service: A new approach to event processing. In *Journal of Physics: Conference Series*, Vol. 664. IOP Publishing, Bristol, UK, 062065. Issue 6.

[9] Gennaro Corcella, Ian G Knowles, Giuseppe Marchesini, Stefano Moretti, Kosuke Odagiri, Peter Richardson, Michael H Seymour, and Bryan R Webber. 2001. HERWIG 6: an event generator for hadron emission reactions with interfering gluons (including supersymmetric processes). *Journal of High Energy Physics* 2001, 01 (2001), 010.

[10] D Crooks, P Calafiura, R Harrington, M Jha, T Maeno, S Purdie, H Severini, S Skipsey, V Tsulaia, R Walker, and others. 2012. Multi-core job submission and grid resource scheduling for ATLAS AthenaMP. In *Journal of Physics: Conference Series*, Vol. 396. IOP Publishing, Bristol, UK, 032115. Issue 3.

[11] J De Favereau, C Delaere, P Demin, A Giammanco, V Lemaître, A Mertens, and M Selvaggi. 2013. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. (2013). arXiv:1307.6346

[12] Jack Dongarra, Hans Meuer, and Erich Strohmaier. 2016. Top500 Supercomputing Sites. http://www.top500.org. (2016).

[13] Ian Foster and Carl Kesselman. 2003. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, Amsterdam, Netherlands.

[14] Vincent Garonne, Graeme A Stewart, Mario Lassnig, Angelos Molfetas, Martin Barisits, Thomas Beermann, Armin Nairz, Luc Goossens, Fernando Barreiro Megino, Cedric Serfon, and others. 2012. The atlas distributed data management project: Past and future. In *Journal of Physics: Conference Series*, Vol. 396. IOP Publishing, Bristol, UK, 032045. Issue 3.

[15] T Goodale, S Jha, H Kaiser, T Kielmann, P al Kleijer, A Merzky, J Shalf, and C Smith. 2008. A Simple API for Grid Applications (SAGA). OGF Document Series 90. (2008).

[16] A Klimentov, P Nevski, M Potekhin, and T Wenaus. 2011. The ATLAS PanDA Monitoring System and its Evolution. In *Journal of Physics: Conference Series*, Vol. 331. IOP Publishing, Bristol, UK, 072058. Issue 7.

[17] T Maeno. 2011. Overview of ATLAS PanDA workload management. *J. Phys.: Conf. Ser* 331, 7 (2011), 072024. http://stacks.iop.org/1742-6596/331/i=7/a=072024

[18] T Maeno, K De, A Klimentov, P Nilsson, D Oleynik, S Panitkin, A Petrosyan, J Schovancova, A Vaniachine, T Wenaus, and others. 2014. Evolution of the ATLAS PanDA workload management system for exascale computational science. In *Journal of Physics: Conference Series*, Vol. 513. IOP Publishing, Bristol, UK, 032062. Issue 3.

[19] Tadashi Maeno, K De, T Wenaus, P Nilsson, GA Stewart, R Walker, A Stradling, J Caballero, M Potekhin, D Smith, and others. 2011. Overview of atlas panda workload management. In *Journal of Physics: Conference Series*, Vol. 331. IOP Publishing, Bristol, UK, 072024. Issue 7.

[20] Cecchi Marco, Capannini Fabio, Dorigo Alvise, Ghiselli Antonia, Giacomini Francesco, Maraschini Alessandro, Marzolla Moreno, Monforte Salvatore, Petronzio Luca, and Prelz Francesco. 2009. The glite workload management system. In *International Conference on Grid and Pervasive Computing*. Springer Publishing, New York City, NY, USA, 256–268.

[21] P Nilsson, J Caballero, K De, T Maeno, A Stradling, T Wenaus, Atlas Collaboration, and others. 2011. The ATLAS PanDA pilot in operation. In *Journal of Physics: Conference Series*, Vol. 331. IOP Publishing, Bristol, UK, 062040. Issue 6.

[22] Sarp Oral, David A Dillow, Douglas Fuller, Jason Hill, Dustin Leverman, Sudharshan S Vazhkudai, Feiyi Wang, Youngjae Kim, James Rogers, James Simmons, and others. 2013. OLCF's 1 TB/s, next-generation lustre file system. (2013).

[23] Stuart Paterson, Joel Closier, and the Lhcb Dirac Team. 2010. Performance of combined production and analysis WMS in DIRAC. *Journal of Physics: Conference Series* 219, 7 (2010), 072015. http://stacks.iop.org/1742-6596/219/i=7/a=072015

[24] A Rimoldi, A Dell'Acqua, A di Simone, M Gallas, A Nairz, J Boudreau, V Tsulaia, and D Costanzo. 2006. Atlas Detector Simulation: Status and Outlook. In *Astroparticle, Particle and Space Physics, Detectors and Medical Physics Applications*, Vol. 1. World Scientific Publishing, Singapore, 551–555.

[25] Elmar Ritsch. 2014. *ATLAS Detector Simulation in the Integrated Simulation Framework applied to the W Boson Mass Measurement.* Ph.D. Dissertation. Innsbruck U.

[26] Igor Sfiligoi. 2008. glideinWMS—a generic pilot-based workload management system. In *Journal of Physics: Conference Series*, Vol. 119. IOP Publishing, Bristol, UK, 062044. Issue 6.

[27] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. 2006. PYTHIA 6.4 physics and manual. *Journal of High Energy Physics* 2006, 05 (2006), 026.

[28] ATLAS PanDA Team. 2017. The PanDA Production and Distributed Analysis System. (2017). Retrieved March 25, 2017 from https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA

[29] RADICAL Team. 2017. RADICAL-SAGA Software Toolkit. (2017). Retrieved March 21, 2017 from https://github.com/radical-cybertools/radical-saga

[30] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: the Condor experience. *Concurrency and computation: practice and experience* 17, 2-4 (2005), 323–356.

[31] Matteo Turilli, Mark Santcroos, and Shantenu Jha. 2015. A Comprehensive Perspective on Pilot-Job Systems. (2015). arXiv:1508.04180

A ARTIFACT DESCRIPTION — CONVERGING HIGH-THROUGHPUT AND HIGH-PERFORMANCE COMPUTING: A CASE STUDY

A.1 Abstract

Experiments at the Large Hadron Collider (LHC) face unprecedented computing challenges. Thousands of physicists analyze exabytes of data every year, using billions of computing hours on hundreds of computing sites worldwide. PanDA (Production and Distributed Analysis) is a workload management system (WMS) developed to meet the scale and complexity of LHC distributed computing for the ATLAS experiment. PanDA is the first exascale workload management system in HEP, executing millions of computing jobs per day, and processing over an exabyte of data in 2016. In this paper, we introduce the design and implementation of PanDA, describing its deployment on Titan, the third biggest supercomputer in the world. We analyze scalability, reliability and performance of PanDA on Titan, highlighting the challenges addressed by its architecture and implementation. We present preliminary results of experiments performed with the Next Generation Executer, a prototype we developed to meet new challenges of scale and resource heterogeneity.

A.2 Description

A.2.1 Check-list (artifact meta information).

- **Program: ATLAS Monte Carlo Workflow, Geant4, AthenaMP, GROMACS**
- **Data set: Available at** https://github.com/ATLAS-Titan/PanDA-WMS-paper/tree/master/data
- **Run-time environment: PanDA Workload Management System, RADICAL-Pilot Pilot System (NGE)**
- **Hardware: OLCF Titan Cray XK7**
- **Output: Available at** https://github.com/ATLAS-Titan/PanDA-WMS-paper/tree/master/data
- **Experiment workflow: Raw data acquisition, data wrangling and filtering, plotting, analysis**
- **Publicly available?: Yes**

A.2.2 How software can be obtained (if available).

- PanDA Workload Management System: https://github.com/PanDAWMS.
- RADICAL-Pilot Pilot System (NGE): https://github.com/radical-cybertools.

A.2.3 Hardware dependencies. Access and allocation on OLCF Titan Cray XK7, workstation with at least 8GB of RAM.

A.2.4 Software dependencies. Python, jupyter, pandas, matplotlib, gnuplot, excel.

A.2.5 Datasets.

Section 5.1 – Figure 3
Section 5.1 – Figure 4
Section 5.1 – Figure 5 https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5
Section 5.1 – Figure 6
Section 5.2 – Figure 7
Section 5.2 – Figure 8
Section 5.3 – Figure 9
Section 6.3 – Figure 14
Section 6.3 – Figure 15
Section 6.3 – Figure 16
Section 6.3 – Figure 17
Section 6.3 – Figure 18
Section 6.3 – Figure 19
Section 6.3 – Figure 20
Section 6.3 – Figure 21

A.3 Installation

Section 5.1 – Figure 3
Section 5.1 – Figure 4
Section 5.1 – Figure 5
Section 5.1 – Figure 6
Section 5.2 – Figure 7

Section 5.2 – Figure 8
Section 5.3 – Figure 9
Section 6.3 – Figure 14
Section 6.3 – Figure 15
Section 6.3 – Figure 16
Section 6.3 – Figure 17
Section 6.3 – Figure 18
Section 6.3 – Figure 19
Section 6.3 – Figure 20
Section 6.3 – Figure 21

A.4 Experiment workflow

Section 5.1 – Figure 3
Section 5.1 – Figure 4
Section 5.1 – Figure 5
Section 5.1 – Figure 6
Section 5.2 – Figure 7
Section 5.2 – Figure 8
Section 5.3 – Figure 9
Section 6.3 – Figure 14
Section 6.3 – Figure 15
Section 6.3 – Figure 16
Section 6.3 – Figure 17
Section 6.3 – Figure 18
Section 6.3 – Figure 19
Section 6.3 – Figure 20
Section 6.3 – Figure 21

A.5 Evaluation and expected result

Section 5.1 – Figure 3
Section 5.1 – Figure 4
Section 5.1 – Figure 5
Section 5.1 – Figure 6
Section 5.2 – Figure 7
Section 5.2 – Figure 8
Section 5.3 – Figure 9
Section 6.3 – Figure 14
Section 6.3 – Figure 15
Section 6.3 – Figure 16
Section 6.3 – Figure 17
Section 6.3 – Figure 18
Section 6.3 – Figure 19
Section 6.3 – Figure 20
Section 6.3 – Figure 21

B OLD – REMOVE AFTER COMPLETING ABOVE

Reproducibility initiative appendices: Artifact Description (AD) and Computational Results Analysis (CRA). Description of the organization of the appendix.

B.1 Section 5.1 - Figure 3

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.2 Section 5.1 - Figure 4

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.3 Section 5.1 - Figure 5

Correlation between backfill availability of cores and walltime on Titan during the experiment time window.

- Raw data: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5/titan_backfill_availability.txt, as provided by the PanDA Brokers logs on Titan.
- Wrangling: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5/titan_backfill_availability.csv
- Analysis and Plotting: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5/titan_backfill_availability.ipynb

Replicability:

(1) Get a log file including regular polling of backfill availability from one or more PanDA Brokers on titan.
(2) Wrangle the ascii file with the find/replace regular expressions listed at: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5/titan_backfill_availability.ipynb, cell #1.
(3) Load the Jupyter workbook from a python virtual environment with the module Jupyter installed and all the modules listed at: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_5/titan_backfill_availability.ipynb, cell #2.
(4) Execute each cell of the Jupyter notebook to replicate Figure 5.

B.4 Section 5.2 - Figure 6

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.5 Section 5.2 - Figure 7

Artifacts description.

- Experiment code: repository address (when applicable)

- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.6 Section 5.3 - Figure 8

Panda failures on Titan by exit code during the experiment window.

- Raw data: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_8/panda-broker-failures_jan2016-feb2017.csv
- Wrangling: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_8/panda-broker-failures_jan2016-feb2017.xlsx (Pivot Data & Wrangled Data tabs)
- Analysis: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_8/panda-broker-failures_jan2016-feb2017.xlsx (Aggregated Data tab)
- Plotting: https://github.com/ATLAS-Titan/PanDA-WMS-paper/blob/master/data/figure_8/panda-broker-failures_jan2016-feb2017.xlsx (Plots tab)

Replicability:

(1) Download raw data download in csv format from the ATLAS jobs dashboard. Download link: http://dashb-atlas-job.cern.ch/dashboard/request.py/terminatedjobsstatuscsv?sites=All%20T3210&sitesCat=All%20Countries&resourcetype=All&pandares=ORNL_Titan_MCORE&activities=simul&sitesSort=7&sitesCatSort=0&start=2016-01-01&end=2017-02-28&timeRange=daily&sortBy=16&granularity=Monthly&generic=0&series=30&type=abcb
(2) Load the csv file into Excel (or any other software/language).
(3) Create a pivot table with failure types as columns and months as rows.
(4) Aggregate data of error codes of the same type by merge columns and adding the values of their raws according to this table:
(5) Plot the resulting table as a stacked area diagram.

B.7 Section 6.3 - Figure 14

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.8 Section 6.3 - Figure 15

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.9 Section 6.3 - Figure 16

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)

B.10 Section 6.3 - Figure 17

Artifacts description.

- Experiment code: repository address (when applicable)
- Raw data: repository address
- Wrangling: repository address
- Analysis code: repository address
- Plotting code: repository address

Replicability:

(1)