Noname manuscript No.
(will be inserted by the editor)

Production Workload Management on Leadership Class Facilities

Do you have a subtitle? If so, write it here

First Author · Second Author

Received: date / Accepted: date

Abstract Insert your abstract here. Include keywords, PACS and mathematical subject classification numbers as needed.

Keywords First keyword · Second keyword · More

1 Introduction

Traditionally, the ATLAS experiment at LHC has utilized distributed resources as provided by the WLCG to support data distribution and enable the simulation of events. For example, the ATLAS experiment uses a geographically distributed grid of approximately 200,000 cores continuously (250,000 cores at peak), (over 1,000 million core-hours per year) to process, simulate, and analyze its data (today's total data volume of ATLAS is more than 300 PB). After the early success in discovering a new particle consistent with the long awaited Higgs boson, ATLAS is starting the precision measurements necessary for further discoveries that will become possible by much higher LHC collision energy and rates from Run2. The need for simulation and analysis will overwhelm the expected capacity of WLCG computing facilities unless the range and precision of physics studies will be curtailed.

Over the past few years, the ATLAS experiment has been investigating the implications of using high-performance computers – such as those found

Grants or other notes about the article that should go on the front page should be placed here. General acknowledgments should be placed at the end of the article.

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

at Oak Ridge leadership class facility (ORNL). This steady transition is a consequence of application requirements (e.g., greater than expected data production), technology trends and software complexity.

Our approach to the exascale involve the BigPanDA workload management system which is responsible for coordination of tasks, orchestration of resources and job submission and management. Historically, BigPanDA was used to for workload management across multiple distributed resources on the WLCG. We describe the changes to the BigPanDA software system needed to enable BigPanDA to utilize Titan. We will then describe how architectural, algorithmic and software changes have also been addressed by ATLAS computing.

We quantify the impact of this sustained and steady uptake of supercomputers via BigPanDA: For the latest 18 month period for which data is available, Big Panda has enabled the utilization of ∼400 Million Titan core hours (primarily via Backfill mechanisms 275M, but also through regular "front end" submission as part of the ALCC project 125M). This non-trivial amount of 400 million Titan core hours has resulted in 920 million events being analysed. Approximately 3-5% of all of ATLAS compute resources now provided by Titan; other DOE supercomputers provide non-trivial compute allocations. In spite of these impressive numbers, there is a need to further improve the uptake and utilization of supercomputing resources to improve the ATLAS prospects for Run 3.

In spite of these impressive numbers, there is a need to further improve the uptake and utilization of supercomputing resources to improve the ATLAS prospects for Run 3. The aim of this paper to (i) . . . (ii) . . . (iiii) . . . (iv) We will outline how we have steadily made the ATLAS project ready for the exascale era . . .

2 PanDA Workload Management System: Software System Overview

PanDA is a Workload Management System (WMS) [1] designed to support the execution of workloads in grid-like distributed computing environment via pilots [2]. Pilot-capable WMS enable high throughput of task execution via multi-level scheduling while supporting interoperability across multiple sites. This is particularly relevant for Large Hadron Collider (LHC) experiments, where millions of tasks are executed across multiple sites of the Worldwide LHC Computing Grid (WLCG) every month, analyzing and producing petabytes of data. The design of PanDA WMS started in 2005 to support ATLAS.

2.1 Design

PanDA's application model assumes tasks grouped into workloads. Tasks represent a set of homogeneous operations performed on datasets stored in a set

of input files. Tasks are decomposed into jobs, where each job consists of the task's operations performed on a partition of the task's data set. PanDA acts as a unified interface to distributed computing resources, enabling users of the ATLAS project to submit datasets for processing. Data can be processed by a single job or by one or more tasks created for each dataset, where each task is partitioned into one or more jobs. Jobs are distributed across the available resources for concurrent execution. So-called "production workflows" are sets of transformations of collected and simulated data into the formats required for user analysis. These workflows are rendered into a set of tasks and therefore executed as a set of jobs on diverse computing infrastructures.

PanDA's security model is based on separation among authentication, authorization, and accounting for both single users and groups of users. Both authentication and authorization are based on digital certificates and on the virtual organization abstraction [3]. Currently, PanDA's execution model is based on four main abstractions: task, job, global queue, and pilot. 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 task and job. PanDA has a global queue where all jobs are registered and one resource queue for each target computing resource. PanDA assigns specific sets of jobs from the global queue to the resource queues, depending on the jobs requirements, and each resource capability and availability. When pilots become available on the target compute resource, PanDA sends jobs on each pilot from the resource queue associated with that target compute source.

In PanDA's data model, each datum refers to a recorded or simulated measurement of a physical process. Data are stored in files that are grouped into datasets, with a many-to-many relationship between files and datasets. As with jobs, data have both attributes and states, and some of the attributes are shared among events and jobs. Raw, reconstruction, and simulation data are all assumed to be distributed across multiple storage facilities and managed by the ATLAS Distributed Data Management (DDM) [4]. When necessary, input files 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 past and present stages of the execution for each job and data file.

2.2 Implementation and Execution

The implementation of PanDA WMS consists of several interconnected subsystems, most of them built from off-the-shelf and Open Source components. Subsystems communicate via messaging using HTTP and dedicated APIs, and each subsystem is implemented by one or more modules. Databases are used

to store eventful entities like tasks, jobs, and input/output data and to store information about sites and resources.

Currently, PanDA's architecture has five main subsystems: JEDI/PanDA Server [5,6], PanDA Pilot [7], Schedconfig [8], AutoPyFactory [9] and PanDA Monitoring [10]. JEDI and PanDA Server process tasks into jobs and broker their distributed execution. PanDA Pilot is a pilot system that executes jobs on computing infrastructures, managing the stage-in and stage-out of their data. Schedconfig is an information system implemented within PanDA to store PanDA queue (resource) descriptions. Schedconfig synchronizes with the ATLAS Grid Information system (AGIS) [11] to obtain information about distributed resources. AutoPyFactory is the system used to submit pilots to grid sites. PanDA Monitoring is a web application to monitor the execution of workloads in a distributed computing environment. Other subsystems are used by some of ATLAS workflows (e.g., ATLAS Event Service [12] and ATLAS Production System [13]), but their discussion is omitted here because they are irrelevant to understanding how PanDA has been ported to supercomputers. For a full list of subsystems, see Ref. [14].

Figure 1 shows a diagrammatic representation of PanDA's main subsystems, highlighting the execution process of tasks while omitting monitoring details to improve readability. During the first data collection period at the LHC (LHC Run 1), PanDA required users to perform a static conversion between tasks and jobs; tasks were described as a set of jobs and then submitted to the PanDA Server. This introduced inefficiency both with usability and resource utilization. Ideally, users should conceive analyses in terms of one or more potentially related tasks, while the workload manager (i.e., PanDA) should partition tasks into jobs, depending on the amount of data (i.e., number of input files and events) that need to be processed and the task requirements.

PanDA registers three types of workloads for execution: a set of tasks submitted by the ATLAS Production system (Fig. 1:1); a single user task when a whole dataset is submitted for processing (Fig. 1:2); and a single job submitted by the user, usually with a single input file (Fig. 1:3).

The Job Generator component derives a set of jobs for each registered task based on the amount of input data (number of files) and the amount of processed data per job (number of events per job from task parameters). For each task, not all jobs are generated in one interaction. In this way, the job buffer is kept to a manageable size, enabling tuning of task parameters based on initial job results and, if needed, task prioritization (Fig. 1:4).

Once derived, jobs are collected into the Job Buffer, waiting to be bound to a specific resource for execution (Fig. 1:5). The Brokerage component pulls jobs from the Job Buffer (Fig. 1:6), binding each job to a computing resource based on job requirements, resource capability (Fig. 1:7 and (Fig. 1:8), and data availability (Fig. 1:9). Note that the Resource Configuration component is also called Schedconfig.

Once bound, jobs are passed to the Job Scheduler to be scheduled on the assigned resource and executed (Fig. 1:10). Before scheduling each job, the

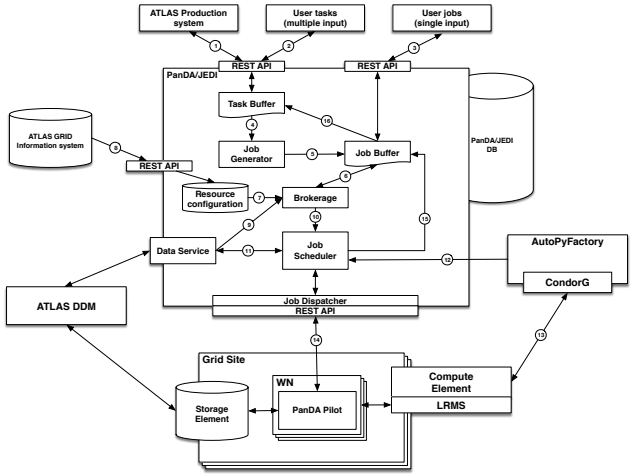


Fig. 1 PanDA WMS architecture. Numbers indicate the JEDI-based execution process described in section 2.2. Several subsystems, components, and architectural and communication details are abstracted to improve clarity.

Job Scheduler checks whether input data are available and, if needed, requests a transfer to the storage associated with the target resource (Fig. 1:11).

Meanwhile, AutoPyFactory defines PanDA Pilots on the basis of the number of jobs bound and ready to execute, and it submits these pilots to a Condor-G agent (Fig. 1:12). Condor-G schedules these pilots on the required sites (Fig. 1:13). Once active, pilots interact with the Job Dispatcher to pull jobs for execution (Fig. 1:14). Depending on task and job parameters, failed jobs can be rescheduled with a new Job ID (Fig. 1:15).

Once all the jobs of a task have been executed and, depending on the failure policy, all or most of the output data have been collected, tasks are marked as done. Thus, no more jobs will be generated for that task, and ATLAS Production System or single users will be informed about the completion of their tasks (Fig. 1:16). Tasks can also be marked as failed, depending on whether a user-defined threshold for number of failures has been exceeded.

2.3 Job State Definitions in PanDA

The life cycle of the job in the PanDA system is split into a series of sequentially changing states. Each state is literally coupled with the PanDA job status used by the different algorithms and monitoring. The status reflects the current step of the job processing since the time that the job was submitted to the system, transferred to the particular resource and finally executed. Figure 2 illustrates the life cycle of jobs submitted to PaNDA WMS.

Jobs are injected into the system by JEDI in ATLAS or by the PanDA client in the general case as a “job parameters” object with a “Pending” status.

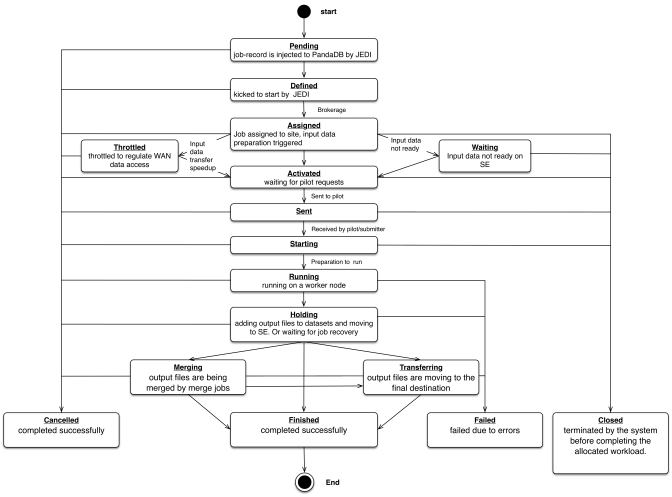


Fig. 2 This is a job state transitions model diagram for PanDA.

Initially, this object is represented by a string containing unsorted parameters. Next, the string is processed and the parameters of the job are sorted into dedicated database fields, and the status is changed to "Defined." After that, the job is processed through the brokerage algorithm and assigned to a particular resource (PanDA queue), and the status is changed to "Assigned". Then, the status is changed to "Waiting" while the PanDA server checks the availability of the input data and the required software at the resource before changing the job's status to "Activated". An activated job is ready to be dispatched to the next corresponding pilot. When the job is dispatched and taken by the pilot, the job's status is changed to "Sent". At this time, the handling of the job processing has not been delegated to the pilot, and thus, the next few job states correspond to the steps of the job processing on the assigned resource. When the job status changes to "Starting", the pilot is starting the job on a worker node or local batch system, after which the status becomes "Running" when the job begins running on a worker node. At this time, the progression of states is once again handled by the server instead of by the pilot. When the job finishes executing and output and log files are transferred, then the PanDA server is responsible to register the files in the file catalog. At the same time, the pilot returns the final status of the job to the server by communicating that the job either succeeded or failed. During this process, the job has a "Holding" status. The PanDA server check the output files regularly by using cron, and it assigns the final "Finished" or "Failed" status to the job. There are also some other statuses, the two most important of which are "Cancelled" for manually killed jobs and "Closed" for jobs which were terminated by the system before completion so they could be reassigned to other sites.

2.4 Brokerage Characterization

Resources (queues) presented in the database together with the wide set of static parameters such as walltime, CPU cores, memory, disk space etc. Same parameters can be provided within job definition to specify strict demands to the resource where the job can be executed. Both resources (queues) and jobs with parameters stored in the PanDA database.

Also PanDA server maintains in the DB the dynamic information for queues about the number of defined, activated and running jobs and also the pilots statistics - number of requests of different types like "get job" or "update job status".

PanDA Broker - key component of the BigPanDA workflow automation - is an intelligent module designed to prioritize and assign PanDA jobs (job passed the brokerage transitioning from "defined" to the "assigned" state) to available computing resources on the basis of job type, software availability, input data and its locality, real-time job statistics, available CPU and storage resources and etc. Users are able to specify explicitly the resource while job submission or they can rely on automated brokerage engine. Full power of the PanDA brokerage integrated with another distributed computing and data management tools (internal and external with respect to the PanDA) is actively used in ATLAS experiment. In this paper we will present and will benchmark the basic brokerage functionality.

The basic brokerage algorithm works the following way. It takes the lists of submitted jobs and available queues. Then each job is checked against each queue by set of parameters if the queue meets the jobs static demands like number of CPU core or the walltime. All queues passed the round are proceeding to the short list where for each queue Broker calculates the weight on the basis of current job statistics for given queue according to the formula (1). Job finally assigning to the queue with bigger weight. Weight calculation algorithm fo ATLAS is more complicated and taking into account clouds default weights, network bandwidth, sharing policies etc.

The basic brokerage algorithm works the following way. Having the list of the submitted jobs, each job is checked against available resources as shown in SELECT_CAND (Alg.). Available resources presented as the set of defined PanDA queues: res = $\text{queue}_1, \ldots, \text{queue}_n$. For each queue in the set (3) we checking if it's satisfying the parameters of job (4). Successfully passed queues are concatenating to the list of candidate-queues (5).

SATISFY_JOB function (Alg.) is used to check if the queue attributes can scope job parameters. Set of the job parameters defined as $\text{par}_1, \ldots, \text{par}_m$ represents the software/hardware demands to the resource like CPU core count, walltime, SW releases etc. Each of these parameters can be mapped to the set of queue attributes defined as $\text{atr}_1, \ldots, \text{atr}_n$, where $n \geq m$. So for each job parameter (2) we check if it can be satisfied with the corresponding queue attribute (3). Finally queue passes the test if it copes all the jobs parameters (5).

The procedure SATISFY_REQ (Alg.) is responsible to testing if the value of the job parameter is in the set of allowed values $\text{val}_1, \ldots, \text{val}_k$ of the queue attribute (2).

Listing 1 Caption

```
# Require: par; atr = (val_1, . . . , val_k)
# Ensure: True or False
# 1. procedure SATISFY REQ(par, atr)
# 2.     if par.value in atr then:
# 3.             return True
# 4.     return False

# This file corresponds to the \lstinputlisting example added by Matteo.
Require: par; atr = (val_1, ..., val_k)
```

Require: par; atr = ($\text{val}_1, \ldots, \text{val}_k$)
Ensure: True or False
1: procedure SATISFY_REQ(par, atr)
2: if par.value in atr then:
3: return True
4: return False

Require: job = {$\text{par}_1, \ldots, \text{par}_m$}; queue = {$\text{atr}_1, \ldots, \text{atr}_n$}
Ensure: True or False
1: procedure SATISFY_JOB(queue, job)
2: for all par in job do:
3: if SATISFY_REQ(par, atr)= False then
4: return False
5: return True

Require: job; res = ($\text{queue}_1, \ldots, \text{queue}_n$)
Ensure: cand
1: procedure SELECT_CAND(job, res)
2: cand ← NONE
3: for all queue in res do:
4: if SATISFY_JOB(queue, job) = True then
5: cand ∪ queue
6: return True

As it was shown SELECT_CAND procedure provides generates the short list of the candidates queues. SELECT_QUEUE (Alg.) taking the short list of the candidate-queues as the set $\text{queue}_1, \ldots, \text{queue}_n$. For each queue (4) Broker calculates the weight (5) on the basis of current job statistics for given queue according to the formula (1). Job finally assigning to the queue with bigger weight (6-7). Weight calculation algorithm fo ATLAS is more complicated and taking into account clouds default weights, network bandwidth, sharing policies etc

```
Require: cand = (queue_1, ..., queue_n)
Ensure: res_queue
1:  procedure SELECT_QUEUE(cand)
2:    res_queue ←queue_1
3:      max_weight ← 0
4:    for all queue in cand do:
5:    queue.weight ← WEIGHT_CALC(queue)
6:      if queue.weight > max_weight then
7:        res_queue← queue
8:      return res_queue
```

$$manyAssigned = \max(1, \min(2, \frac{assigned}{activated})),$$
$$weight = \frac{running + 1}{(activated + assigned + sharing + defined + 10) * manyAssigned} \quad (1)$$

Response time of the brokerage in general can be estimated as (2). Basically it's time the job transits from "defined" to assigned state.

$$T = \sum_{i=1}^{Q} \sum_{j=1}^{J} T_{ij} \quad (2)$$

In formula (2) Q is the number of available queues, J is the number of concurrently submitted jobs and T_{ij} is the time to process job j for queue i. The processing time includes the check if queue meet demands of the job. Then for successfully selected queues the weight is calculating and job assigning for the queue with bigger weight. Hence the time T can be presented as sum (3).

$$T = t_1 + t_2 + t3_3 + C \quad (3)$$

In formula 3, t_1 is the time to make checks if queue meet demands of the job, t_2 is the time for weight calculation and finally t_3 is the time spent to assign job to the resulted queue.

Under the assumption that all jobs can run on the same average number of queues N then we can transform equation as (4).

$$T = J * \left(\sum_{i=1}^{Q-N} t1_j + \sum_{j=1}^{N} (tmax + t2_j) + t3 \right) + C, t1 < tmax \quad (4)$$

Here N is the average number of queues which met all demands of each job. As shown in the SATISFY_JOB algorithm the function returns FALSE as soon as the first discrepancy in the job parameter and queue attributes is met. Hence for for all other Q-N queues the time to make checks t1 will be less than tmax.

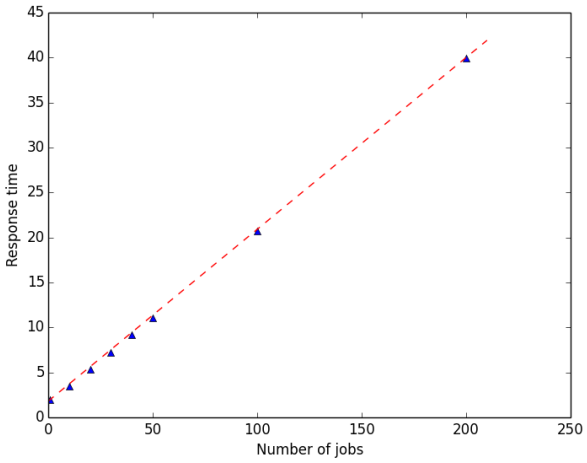


Fig. 3 Response time dependency on number of concurrently submitted jobs

Here N is the average number of queues which met all demands of each job. As shown in the SATISFY_JOB algorithm the function returns False as soon as the first discrepancy in the job parameter and queue attributes is met. Hence for for all other Q-N queues the time to make checks t1 will be less than tmax.

Again taking assumption that the times for different queues are equal we can streamline the equation like (5)

$$\begin{aligned} T =& J * ((Q - N) * t1 + N * (tmax + t2) + t3) + C \\ =& J * (Q * t1 + N * (tmax - t1 + t2) + t3) + C, \text{where } (tmax - t1) > 0 \end{aligned} \tag{5}$$

In order to estimate dependency of brokerage response time from the number of concurrently submitted jobs we deployed a dedicated test instance of PanDA server at ORNL. PanDA was configured to use ten testing queues. Two of the queus was configured to provide 8 CPU cores and eight remaining queues provide 2 cores. All other parameters are configured equal for all queues.

Job submission client was configured to generate and send to the server the lists of equal jobs where each job demands 4 CPU cores. PanDA testing-instance was adjusted to simulate the brokerage two queues will be selected as meeting the criteria of cores number. Then due to simulation of job statistics on that selected queues the jobs will be assigned to the queue with bigger weight. Brokerage time dependency on number of concurrently submitted jobs is shown in figure.

For this experiment we measured the response time for a jobs to transit from the “Defined” status to the “Activated”. As in the test environment the JEDI system wasn’t used and injection of the jobs was done using the simple

python client interaction with PanDA REST API the first stated of the job indicated in PanDA is "Defined" and corresponds to the creation time. Also during this measurements we used no-input jobs. Hence the status of the jobs progressed to the "Activated" immediately after "Defined". In general the time to check input files can be considered as constant for the constant number of input files. So omitting the "Assigned" state in this testing environment is acceptable.

3 Deploying PanDA Workload Management System on Titan

Consistent with its leadership-computing mission of enabling applications of size and complexity that cannot be readily performed using smaller facilities, the OLCF prioritizes the scheduling of large capability jobs (or "leadership-class" jobs). OLCF uses batch queue policy on the Titan systems to support the delivery of large capability-class jobs [15]. OLCF deploys Adaptive Computing's MOAB resource manager. MOAB resource manager supports features that allow it to directly integrate with Cray's Application Level Placement Scheduler (ALPS), a lower-level resource manager unique to Cray HPC clusters [16]. MOAB will schedule jobs in the queue in priority order, and priority jobs will be executed given the availability of required resources. As a DOE Leadership Computing Facility, the OLCF has a mandate that a large portion of Titan's usage come from large, leadership-class (aka capability) jobs. To ensure the OLCF user programs achieve this mission, OLCF policies strongly encourage through queue policy users to run jobs on Titan that are as large as their code will warrant. To that end, the OLCF implements queue policies that enable large jobs to be scheduled and run in a timely fashion [15]. As a result, leadership-class jobs advance to the high-priority jobs in the queue. If a priority job does not fit, i.e., required resources are not available, a resource reservation will be made for it in the future when availability can be assured. Those nodes are exclusively reserved for that job. When the job finishes, the reservation is destroyed, and those nodes are available for the next job. Reservations are simply the mechanism by which a job receives exclusive access to the resources necessary to run the job [16]. However, if policy desires a priority reservation to be made for more than one job, one can specify the creation of reservations for the top N priority jobs in the queue by increasing the keyword RESERVATIONDEPTH to be greater than one. The priority reservation(s) will be re-evaluated (and destroyed/re-created) every scheduling iteration in order to take advantage of updated information.

Beyond the creation of reservations for the top priority jobs, Moab now switches to backfill mode and continues down the job queue until it finds a job that will be able to start and won't disturb the priority reservations made for the highest priority queued jobs, specified by the value of RESERVATIONDEPTH. As time continues and the scheduling algorithm continues to iterate, Moab continues to evaluate the queue for the highest priority jobs. If the highest priority job found will not fit within the available resources, its reservation

is updated, but left where it is. Switching to "backfill mode", Moab searches for a job in the queue that will be able to start and complete without disturbing the priority reservations. If such jobs are started, they will run within backfill. If no such backfill jobs are present in the queue, then available compute resources will remain unutilized.

In describing how the PanDA Workload management system is deployed on Titan, we necessarily describe it integration with the Moab Workload management system. In so doing, two rather different approaches to interfacing the PanDA managed work on Titan are availed: "Batch Queue Mode" and "Backfill Mode". In "Batch Queue Mode", PanDA interacts with Titan's Moab scheduler in a static, non-adaptive manner to executing the work to be performed. In "Backfill Mode", PanDA dynamically shapes the size of the work deployed on Titan to capture resources that may otherwise go unused because the size of the backfill opportunity is otherwise too small or to brief in duration.

In doing so, we demonstrate how Titan is more efficiently utilized by the injection and mixing of small and short-lived tasks in backfill with regular payloads. Cycles otherwise unusable (or very difficult to use) are used for science, thus increasing the overall utilization on Titan without loss of overall quality-of-service. The conventional mix of jobs at OLCF cannot be effectively backfilled because of size, duration, and scheduling policies. Our approach is extensible to any HPC with "capability scheduling" policies.

3.1 PanDA integration with Titan

As we described in previously PanDA is a pilot based WMS. On the Grid pilot jobs are submitted to batch queues on compute sites and wait for the resource to become available. When a pilot job starts on a worker node it contacts the PanDA server to retrieve an actual payload and then, after necessary preparations, executes the payload as a sub process. The PanDA pilot is also responsible for a job's data management on a worker node and can perform data stage-in and stage-out operations.

Taking advantage of its modular and extensible design, the PanDA pilot code and logic has been enhanced with tools and methods relevant for work on HPCs. The pilot runs on Titan's data transfer nodes (DTNs) which allows it to communicate with the PanDA server, since DTNs have good (10 GB/s) connectivity to the Internet. The DTNs and the worker nodes on Titan use a shared file system which makes it possible for the pilot to stage-in input files that are required by the payload and stage-out produced output files at the end of the job. In other words, the pilot acts as a site edge service for Titan. Pilots are launched by a daemon-like script which runs in user space. The ATLAS Tier 1 computing center at Brookhaven National Laboratory is currently used for data transfer to and from Titan, but in principle that can be any ATLAS site. Figure 4 shows schematic view of PanDA interface with Titan. The pilot submits ATLAS payloads to the worker nodes using the local batch system

(Moab) via the SAGA (Simple API for Grid Applications) interface [17]. It also uses SAGA for monitoring and management of PanDA jobs running on Titan's worker nodes. One of the features of the described system is the ability to collect and use information about Titan status, e.g., free worker nodes in real time. The pilot can query the Moab scheduler about currently unused nodes on Titan, using the "showbf" command, and check if the free resource availability time and size are suitable for PanDA jobs, and conforms with Titan's batch queue policies. The pilot transmits this information to the PanDA server, and in response gets a list of jobs intended for submission on Titan. Then based on the job information, it transfers the necessary input data from the ATLAS Grid, and once all the necessary data is transferred the pilot submits jobs to Titan using an MPI wrapper.

The MPI wrappers are Python scripts that are typically workload specific since they are responsible for setup of the workload environment, organization of per-rank worker directories, rank-specific data management, optional input parameters modification, and cleanup on exit. When activated on worker nodes each copy of the wrapper script after completing the necessary preparations will start the actual payload as a subprocess and will wait until its completion. This approach allows for flexible execution of a wide spectrum of Grid-centric workloads on parallel computational platforms such as Titan.

Since ATLAS detector simulations are executed on Titan as discrete jobs submitted via MPI wrapper, parallel performance can scale nearly linearly, potentially limited only by shared file system performance (discussed below). Currently up to 20 pilots are deployed at a time, distributed evenly over 4 DTNs. Each pilot controls from 15 to 350 ATLAS simulation ranks per submission. This configuration is able to utilize up to 112,000 cores on Titan. We expect that these numbers will grow in the near future.

Figure 5 shows Titan core hours consumed per month by the ATLAS Geant4 simulations from January 2017 to September 2018. Please note that during this time our Director's Discretionary project ran 24/7 in pure backfill mode with lowest priority and no defined allocation. In 2017-2018 average resource utilization exceeded 10M core-hours per month and for February and March of 2018 reached 22M core-hours per month. We expect that average monthly utilization will grow due to further optimization of the workload management system.

4 Performance Characterization on Titan

In 2013, the PanDA team began working to incorporate the Titan supercomputer at Oak Ridge National Laboratory as a grid site for the Worldwide LHC Compute Grid, on behalf of the ATLAS Collaboration. The team has operated under several different project identifiers, including CSC108, HEP110, and HEP113. The HEP110 and HEP113 projects represent traditional ASCR Leadership Computing Challenge (ALCC) allocations, but the CSC108 project operates exclusively in what the team has colloquially referred to as "backfill

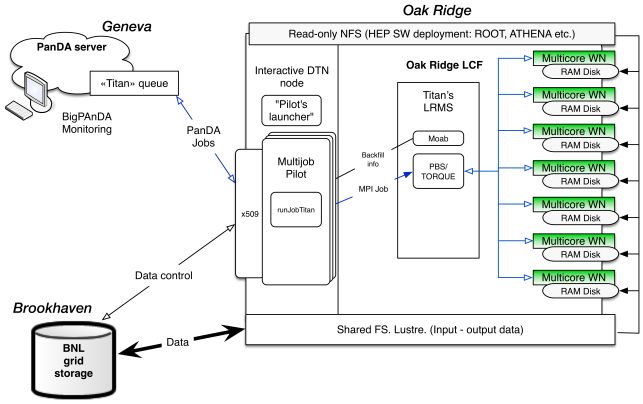


Fig. 4 Schematic view of PanDA WMS integration with Titan supercomputer at OLCF

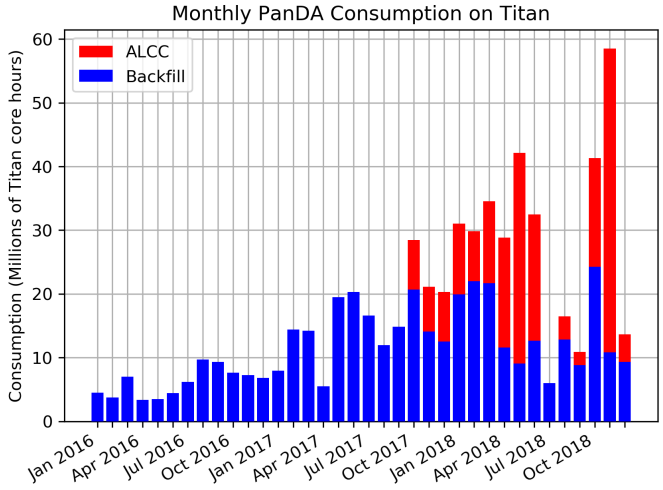


Fig. 5 This figure shows the monthly consumption of resources on Titan by the two methods used by PanDA.

mode", which is outlined in Section 3. The goal of CSC108 has been to consume idle resources on Titan which would otherwise have gone to waste, while making a good-faith effort not to disturb the rest of Titan's ecosystem. The focus of this section is to assess the degree to which CSC108 has accomplished this goal and especially to analyze the impact of the project on Titan.

Before proceeding, however, it is important to summarize the relevant policies at OLCF for running jobs on Titan. There are three queues on Titan: batch, debug, and killable. There is no actual backfill queue on Titan; instead, smaller jobs from each queue are scheduled into spaces that cannot be used by larger jobs. The batch queue is the default queue for submitted jobs, and it is the only queue considered in this analysis. Jobs submitted to the batch queue

Table 1 OLCF policies sort jobs into numbered bins based on the requested number of nodes, and each bin has its own set of constraints.

Bin	Requested Nodes	Maximum Wall Time	Aging Boost
1	11,250 - 18,688	24 hours	15 days
2	3,750 - 11,249	24 hours	5 days
3	313 - 3,749	12 hours	0
4	126 - 312	6 hours	0
5	1 - 125	2 hours	0

are grouped into five "bins" according to the number of requested nodes, and each bin has a maximum wall time. The definitions and rules for each bin are shown in Table 1. Jobs that request fewer nodes have correspondingly lesser maximum wall times. Nodes are assigned exclusively to one job at a time. Because Titan is a leadership class machine and priority is a function of wait time, the batch scheduler awards aging boosts to jobs in bins 1 and 2 in order to prioritize larger jobs over smaller ones. Once jobs in the batch queue begin to run, however, they will not be killed when new jobs arrive, regardless of their priority. Sometimes, jobs small enough to use currently idle resources on Titan are scheduled to run immediately, and this is what we refer to as "backfill opportunity". CSC108 takes advantage of the "showbf" command in order to query the Moab scheduler directly for currently available backfill opportunity and then tailors its own submissions to Titan accordingly. CSC108 has a special policy applied to guarantee that its jobs run with lesser priority than all others on Titan, to ensure that its jobs consume only backfill resources. Finally, "Titan core hours" are the billable units used at OLCF; they convert at a rate of 30 Titan core hours per 1 node hour.

As a convention, we will sometimes make use of a common scheduling metaphor in which jobs waiting to run on a computer are represented by rocks that are being used to fill a jar. Capability class jobs on Titan are the largest rocks, and the scheduler typically fills the remaining space around the largest rocks with smaller rocks. CSC108 attempts to fill whatever space remains, thanks to its having been given the lowest possible priority.

4.1 Compression study

Recall that the goal of CSC108 is to consume idle resources that will otherwise go to waste. As shown in Figure 5, CSC108 has successfully consumed hundreds of millions of core hours on Titan, and because there is no pre-emption of any kind on Titan, all resources that were consumed were guaranteed to have been idle. What remains to be shown is that these resources would otherwise have gone to waste. The simple bar plot shown in Figure 6 seems to show, at first glance, that utilization by CSC108 has increased at the same time that utilization by other projects has decreased, which is suggestive of competition for resources. As a simple test of the hypothesis that CSC108 has no effect on

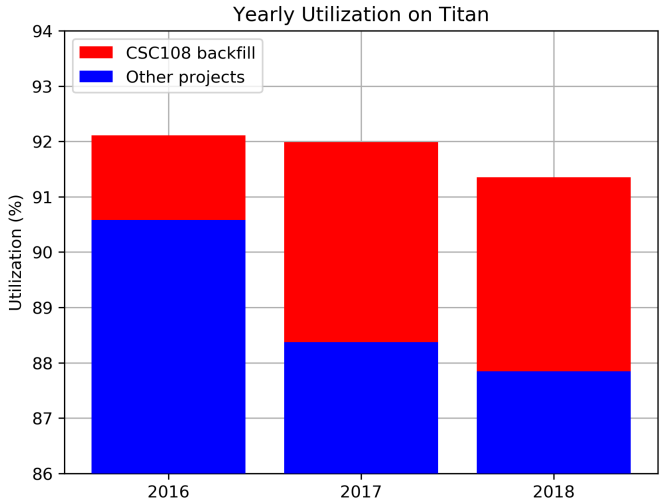


Fig. 6 This bar plot visualizes the yearly utilization of Titan core hours, separated into two categories: CSC108's utilization of Titan core hours through backfill opportunities (shown in red), and all other utilization of core hours on Titan (shown in blue).

Titan, three years' worth of job traces from Titan were "compressed", with and without CSC108's jobs, as a way to measure "displacement" due to CSC108.

The data used for this experiment are historical traces for all jobs on Titan during the years 2016, 2017, and 2018. These data were provided in anonymized form by OLCF so that job, project, and user identifiers were included only for the three PanDA projects on Titan. Otherwise, the job traces consist only of job submission time, start time, completion time, the number of requested nodes, and the amount of wall time requested. The experimental programs were written in Python using the well-known Matplotlib, NumPy, and SciPy libraries via Anaconda. Data were originally provided as text files but were imported into a SQLite database.

What we mean by "compressing" job traces is rescheduling the jobs that ran on Titan while preserving the original execution order, under the assumption of 100% availability (all nodes up and running all the time). Doing this once while including CSC108's jobs and once with CSC108's jobs removed allows effects on throughput and utilization to be estimated simply, using the same logic as measuring the displacement of rocks in a jar by measuring the volume in the jar with and without those rocks. Throughput was defined as jobs completed per day, and utilization was defined as the percentage of available core hours which were consumed.

The rescheduling algorithm is shown as pseudo-code in Listing 2.

Listing 2 Job traces are assumed to be sorted in order of original start time.

```
active_jobs = []
current_time = date("January 1, 2016")
```

Table 2 Compression study results

	Without CSC108	With CSC108	Percent change
Time to completion (days)	1021.2	1034.5	1.30
Throughput (jobs per day)	1324.93	1515.19	14.36
Utilization (percent)	92.36	94.15	1.94

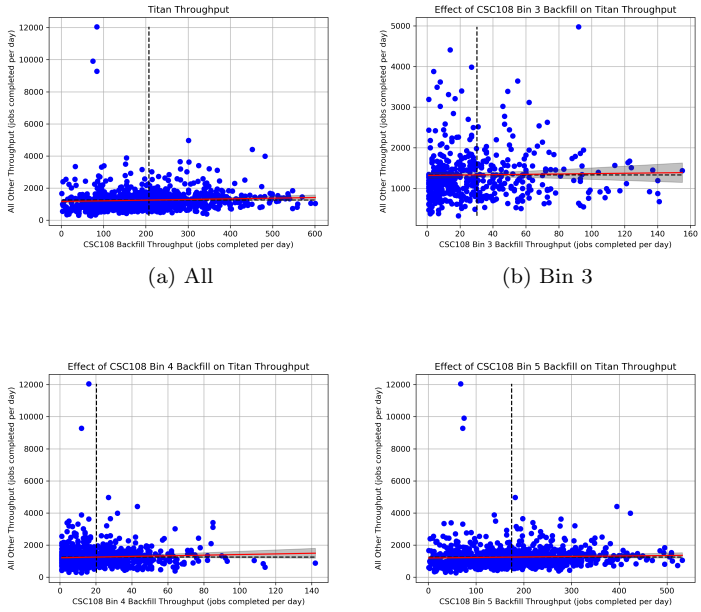
```
job_traces = original_traces.sort_by_start_time()
max_nodes = 18688

for job in job_traces:
    while count_active_nodes() + job["requested_nodes"] > max_nodes:
        current_time = get_next_completion_time() # from active_nodes
        evict_and_log_completed_jobs()
    active_jobs.append(job)

while len(active_jobs) > 0:
    current_time = get_next_completion_time()
    evict_and_log_completed_jobs()
```

The results, which are shown in Table 2, suggest that the hypothesis that CSC108 has no effect on Titan should be rejected. If CSC108 had no effect, then the percent changes in time to completion, throughput, and utilization would all have been zero. The 1.30% increase in the time to completion demonstrates that CSC108's jobs displaced other projects' jobs in this study, but the 1.94% change in utilization indicates that CSC108's jobs must have consumed resources which would otherwise have gone to waste. These results suggest that CSC108 has impacted Titan both positively and negatively in real life, and that the positive impact may justify the negative. More importantly, however, these results suggest that CSC108 has successfully consumed idle resources which would otherwise have gone to waste.

4.2 Simple linear relationships

Recall that the goal of CSC108 has been to consume idle resources on Titan which would otherwise have gone to waste, while also making a good faith effort not to disturb the rest of Titan's ecosystem. The results of Section 4.1 suggested that CSC108 has satisfied part of this goal, but that it may have disturbed Titan's ecosystem. In this section, we detail our explorations to understand the impact of CSC108 on Titan and search for simple linear relationships, especially direct and inverse relationships, by using linear regression.

The data used for this experiment include the same historical trace data used in Section 4.1, supplemented with daily availability data for Titan provided by OLCF for the same years, 2016-2018. The experimental programs were written in Python using the well-known Matplotlib, NumPy, SciPy, and



(a) All (b) Bin 3

(c) Bin 4 (d) Bin 5

Fig. 7 This figure demonstrates the relationship between CSC108 backfill throughput and throughput of other projects on Titan, in terms of jobs completed per day. Each blue point represents one day. Each red line is an Ordinary Least Squares (OLS) linear regression with parameters given in Table 3. Each shaded gray area represents a 95% confidence region. Each horizontal dotted black line represents the mean number of jobs completed on Titan every day by projects other than CSC108, and the vertical dotted black line represents the mean number of jobs completed every day by CSC108's use of backfill opportunity.

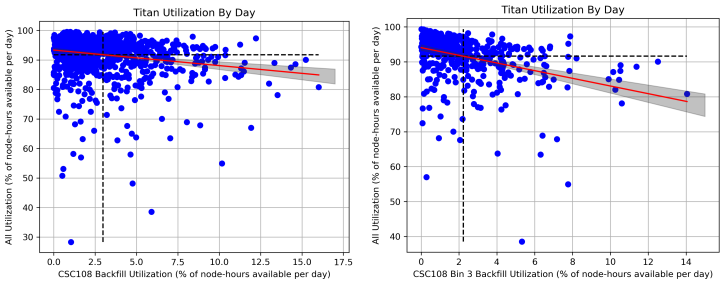
scikit-learn libraries via Anaconda. Data were originally provided as text files but were imported into a SQLite database. We plotted best-fit lines with ordinary least squares (OLS) linear regression, constructed 95% confidence regions around the lines, and used basic measures for goodness-of-fit such as R-squared.

There were two main ideas used here. The first idea was to look for a relationship between CSC108's throughput and the throughput of all other jobs on Titan, and second idea was to look for a relationship between CSC108's utilization as compared with overall utilization on Titan. Additionally, the same ideas were repeated to look for impacts due to different sizes of CSC108's jobs, using the bins defined in Table 1, because jobs are assigned priority differently by the scheduler on Titan in part due to the number of requested nodes and length of requested wall time.

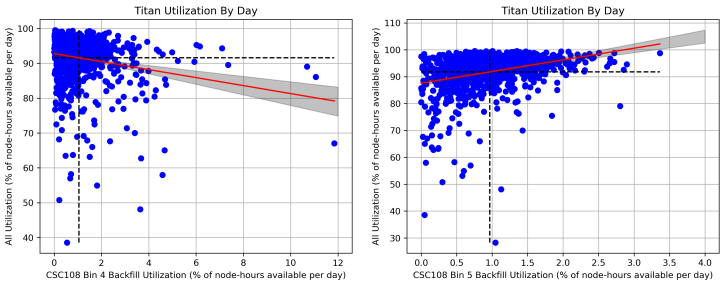
The plots shown in Figures 7a, 7b, 7c, and 7d visually suggest that CSC108 has little to no effect on other projects' throughputs, but the numbers in Table 3 show that linear relationships explain very little of the variability in

Table 3 The table contains the parameter values for the Ordinary Least Squares (OLS) linear regression models regarding throughput. The first column corresponds to the figure depicting the model, and the second column corresponds to the OLCF bin number, as defined in Table 1. The second and third columns correspond the coefficients β_1 and β_0 in the model $y = \beta_1 x + \beta_0$.

Figure	OLCF Bin	Slope β_1	Intercept β_0	R^2
7a	All	0.4106	1164.2561	0.0040
7b	3	0.4419	1322.0784	0.0005
7c	4	1.9819	1211.3384	0.0027
7d	5	0.3072	1195.6684	0.0018



(a) All (b) Bin 3



(c) Bin 4 (d) Bin 5

Fig. 8 This figure demonstrates the relationship between CSC108 backfill utilization and overall utilization on Titan, as percentages of available node-hours each day. Each blue point represents one day. Each red line is an Ordinary Least Squares (OLS) linear regression with parameters given in Table 4. Each shaded gray area represents a 95% confidence regions. Each horizontal dotted black line represents the mean utilization every day on Titan, and each vertical dotted black line represents the mean utilization of backfill opportunity every day by CSC108.

Table 4 The table contains the parameter values for the Ordinary Least Squares (OLS) linear regression models regarding utilization. The first column corresponds to the figure depicting the model, and the second column corresponds to the OLCF bin number, as defined in Table 1. The second and third columns correspond the coefficients β_1 and β_0 in the model $y = \beta_1 x + \beta_0$.

Figure	OLCF Bin	Slope β_1	Intercept β_0	R^2
8a	All	-0.5258	93.3404	0.0330
8b	3	-1.0977	94.0609	0.1359
8c	4	-1.1472	92.7870	0.0378
8d	5	4.3328	87.5839	0.1046

the data. The R^2 values, which represent goodness-of-fit on a scale of 0 to 1, are very close to 0, indicating poor fit.

Similar problems exist for the utilization results, but they raise one very interesting question. The plots shown in Figures 8a, 8b, and 8c all clearly suggest an inverse relationship, but 8d suggests a direct relationship, by virtue of its positive slope. Unfortunately, once again, the numbers show that linear relationships explain very little of the variability in the data, as shown in Table 4, because the R^2 values are very close to 0, on a scale of 0 to 1. This raises the question, what has caused the sign change? It can be tempting to assign blame and credit in such a case, such as to say that CSC108's consumption in bin 5 causes an increase in overall utilization, while its consumption in other bins decreases overall utilization. Here, however, we are only looking for relationships in the data, and the goodness-of-fit values are uniformly poor.

4.3 Blocking probability

Having struggled to find simple linear relationships using throughput and utilization as indicators, we next defined an event called a "block" and looked for its occurrence in the data. A block is said to occur when an eligible job in the batch queue waits due to insufficient resources on Titan for it to begin running; some other job(s) must be using the resources, and therefore the eligible job has been "blocked" by an already-running job. This event is interesting because it can indicate competition for resources even when there are large amounts of idle resources. It also serves as a way to symbolize the event when a user checks the system queue and sees that there are active jobs that are causing the user's job to wait. The goal was to try to detect CSC108's impact by focusing on times of great competition.

The data used for this experiment include the same historical trace data used in Section 4.1 and the daily availability data for Titan used in Section 4.2, but this time are supplemented with live snapshot data of the system queue for Titan. Snapshots were gathered by sampling live data from the Moab scheduler by polling with Python scripts launched by cron jobs on a data transfer node. These scripts recorded XML output from the "showbf" and "showq"

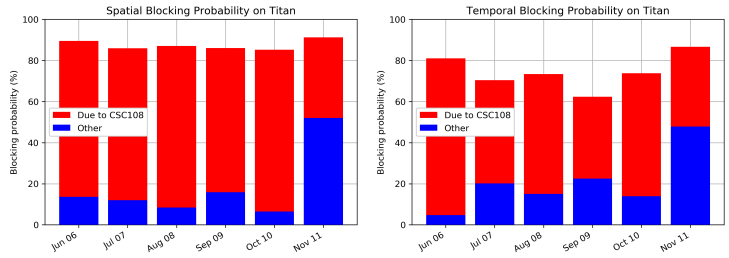
commands into files, and more cron jobs launched other Python scripts to import these files' sample data into SQLite. These tables contain data about the exact state of the queues at given times, including active jobs, blocked jobs, eligible jobs, recently completed jobs, and system information such as active nodes and available backfill opportunities. Then, experimental programs were written in Python using the same libraries and database as the previous sections.

Formally, the definitions for a block and a blocking probability follow. Let C_i be the abstract resources in use by CSC108 at the i^{th} sample point in time, and let U_i be the unused (idle) resources remaining on Titan. We then define a boolean B_i representing a "block" to be 1 if there exists at least one job at the i^{th} sample point which requests $(C_i + U_i)$ resources or less when C_i is non-zero; we define B_i to be zero otherwise. Summing B_i over all i gives a count of sample points at which a block occurred, and dividing that count by the number of total sample points yields a quantity we call a "blocking probability". The blocking probability is a rational number between 0 and 1.

Informally, blocking probability represents the proportion of samples in which a block occurred. The idea here is that when blocking probability increases, it indicates that the system is experiencing greater competition for its resources. Blocking probability does not predict the probability that a particular job will be blocked, but rather the probability that a given sample will contain a block.

To apply this abstract model to a real data set, we have initially defined the resources in one-dimensional "spatial" and "temporal" manners, by considering only jobs' requested numbers of nodes in the former and only jobs' requested wall times in the latter. An eligible job in the batch queue is said to be spatially blocked when the jobs number of requested nodes is too large to fit within the nodes available through backfill opportunity, so that the job must wait to run. Similarly, an eligible job in the batch queue is said to be temporally blocked when the jobs requested wall time is too long to fit within the duration available through backfill opportunity. Similarly, a job is said to be blocked "due to CSC108" if at least one job which was blocked would no longer be blocked if CSC108's jobs were removed. Thus, a job is only said to be blocked due to CSC108 if it requests resources with are greater than U_i but less than $(C_i + U_i)$. Figures 9a and 9b demonstrate how spatial and temporal blocking probabilities vary from month to month, and Figure 10 shows that the two quantities relate to each other in an intuitive way, namely, that time periods of greater spatial blocking often correspond to time periods of greater temporal blocking as well.

Three indicators of system performance were chosen this time, as well, to assess the impact of CSC108 on Titan: wait times, throughput, and utilization. In order to map wait time to a value that can be attributed to a day, wait time was defined in terms of an average wait time. Average wait time was defined as the total number of hours spent waiting during a given day, per job that appeared on that day. For example, a job which was submitted one day but which did not run until the next day would contribute part of its wait time



(a) Spatial blocking (b) Temporal blocking

Fig. 9 These plots depict the spatial and temporal blocking probabilities by month for samples in which CSC108 was actively utilizing backfill opportunity. The total height of the bars indicates the blocking probability for the month, which is the proportion of samples in which at least one eligible job was blocked. The red region indicates the percentage of samples in which at least one eligible job would no longer be blocked if CSC108's jobs were removed.

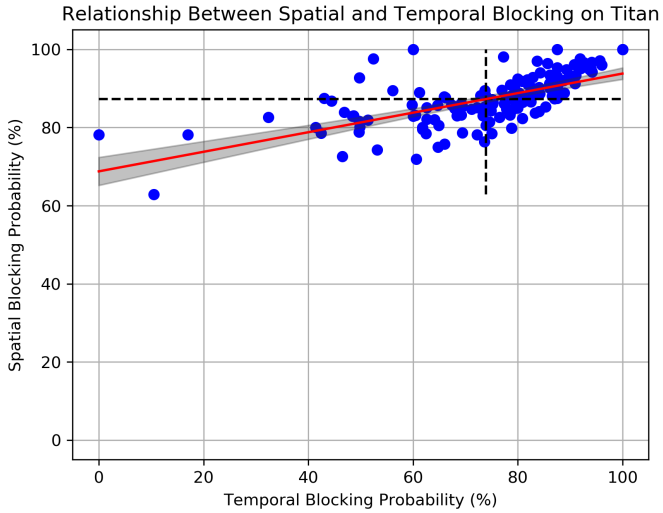
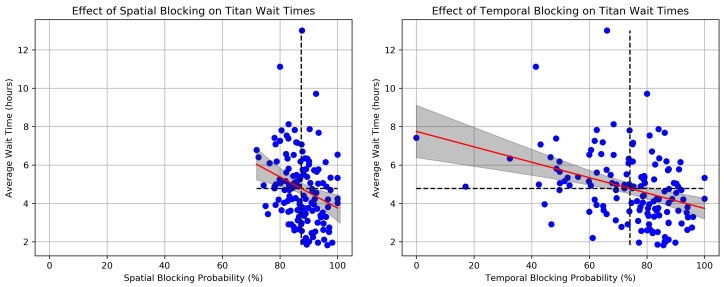
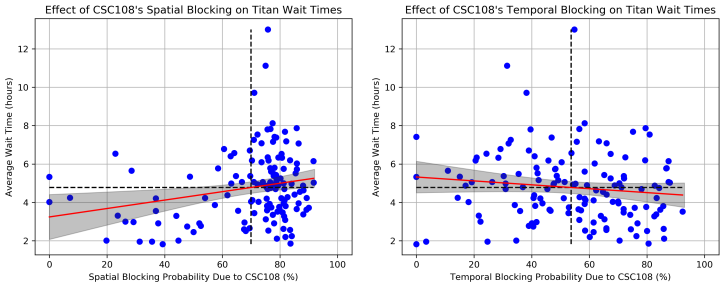


Fig. 10 This figure demonstrates the relationship between spatial and temporal blocking probabilities. Each blue point represents one day. The red line is an Ordinary Least Squares (OLS) linear regression ($y = \beta_1 x + \beta_0$) with a slope β_1 of 0.2503 and an intercept β_0 of 68.7731. The shaded gray areas represent 95% confidence regions. The horizontal dotted black line represents the mean spatial blocking probability for all points, and the vertical dotted black line represents the mean temporal blocking probability for all points. The R^2 value is 0.4410.

to the first day and the rest to the second day, and it would be considered to have appeared on both days. Throughput was defined as the number of jobs completed per day, as before. Utilization was also defined as before, as the percentage of core hours consumed out of the total core hours available.



(a) Spatial blocking (b) Temporal blocking



(c) Spatial blocking by CSC108 (d) Temporal blocking by CSC108

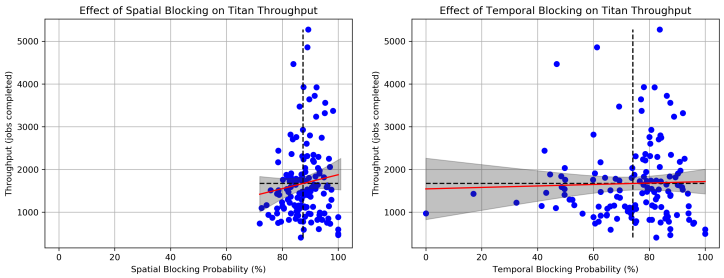
Fig. 11 These plots demonstrate the relationships between the average wait times on Titan and one-dimensional blocking probabilities. Each blue point represents one day. Each red line is an Ordinary Least Squares (OLS) linear regression with parameters given in Table 5. Each shaded gray area represents a 95% confidence region. Each horizontal dotted black line represents the mean wait times for all points in that plot, and each vertical dotted black line represents the mean blocking probability for all points in that plot.

Having established the two measures of blocking probability and their relationship to one another, we followed the same techniques used in Section 4.2 to create best-fit lines with 95% confidence intervals, to investigate the relationships between blocking probabilities and wait times experienced by jobs on Titan. Figures 11a and 11b illustrate the effects of spatial and temporal blocking probability on wait times, and Figures 11c and 11d show how CSC108's contribution to blocking impacts wait times. More specifically, in Figures 11c and 11d, the values used for the blocking probabilities correspond to the red regions in Figures 9a and 9b, which indicate the percentage of samples in which at least one eligible job would no longer be blocked if CSC108 freed its resources. The qualitative interpretation for the wait time plots is that, as competition for resources increases on Titan, average wait times decrease, but when competition with CSC108 for nodes increases, average wait times increase. Unfortunately, the goodness-of-fit values are again very poor.

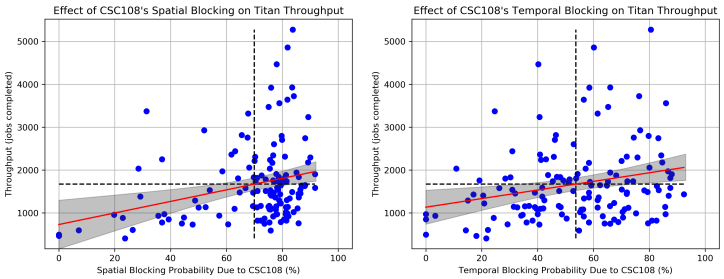
Figures 12a, 12b, 12c, and 12d are all in agreement that increasing competition corresponds to increasing throughput, in units of jobs completed per

Table 5 The table contains the parameter values for the Ordinary Least Squares (OLS) linear regression models regarding blocking probabilities and average wait times. The first column corresponds to the figure depicting the model, while the second and third columns correspond the coefficients β_1 and β_0 in the model $y = \beta_1 x + \beta_0$.

Figure	Slope β_1	Intercept β_0	R^2
11a	-0.0810	11.8610	0.0737
11b	-0.0401	7.7491	0.1265
11c	0.0219	3.2420	0.0509
11d	-0.0102	5.3217	0.0147



(a) Spatial blocking (b) Temporal blocking



(c) Spatial blocking by CSC108 (d) Temporal blocking by CSC108

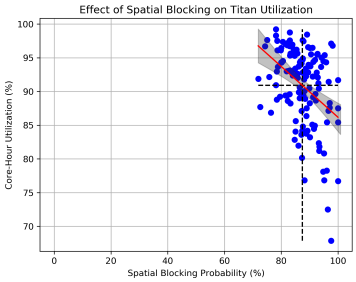
Fig. 12 These plots demonstrate the relationships between throughput on Titan and one-dimensional blocking probabilities. Each blue point represents one day. Each red line is an Ordinary Least Squares (OLS) linear regression with parameters given in Table 6. Each shaded gray area represents a 95% confidence region. Each horizontal dotted black line represents the mean wait times for all points in that plot, and each vertical dotted black line represents the mean blocking probability for all points in that plot.

day. The goodness-of-fit values are poor, however, as shown in Table 6, so these qualitative results may only be said to be suggestive.

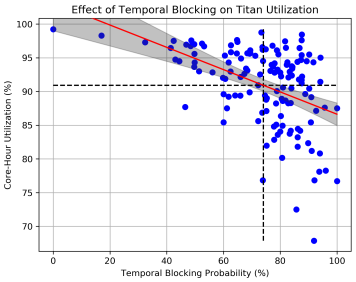
Finally, we searched for simple linear relationships between the different blocking probabilities and overall utilization on Titan. Figures 13a, 13b, 13c, and 13d do not "agree" like the throughput plots did, but three plots suggest an interpretation in which increasing competition, indicated by increasing blocking probability, corresponds to decreased utilization. The fourth plot,

Table 6 The table contains the parameter values for the Ordinary Least Squares (OLS) linear regression models regarding blocking probabilities and throughput. The first column corresponds to the figure depicting the model, while the second and third columns correspond the coefficients β_1 and β_0 in the model $y = \beta_1 x + \beta_0$.

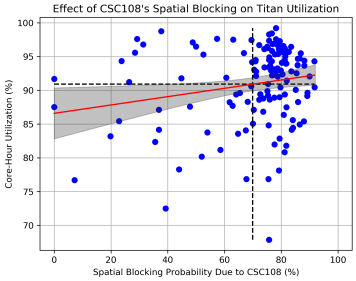
Figure	Slope β_1	Intercept β_0	R^2
12a	16.2402	252.3652	0.0122
12b	1.7196	1544.9669	0.0010
12c	13.4683	730.0687	0.0790
12d	10.0245	1134.0212	0.0587



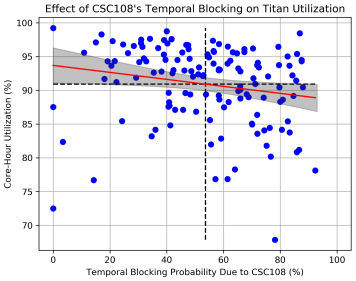
(a) Spatial blocking



(b) Temporal blocking



(c) Spatial blocking by CSC108



(d) Temporal blocking by CSC108

Fig. 13 These plots demonstrate the relationships between utilization on Titan and one-dimensional blocking probabilities. Each blue point represents one day. Each red line is an Ordinary Least Squares (OLS) linear regression with parameters given in Table 7. Each shaded gray area represents a 95% confidence region. Each horizontal dotted black line represents the mean wait times for all points in that plot, and each vertical dotted black line represents the mean blocking probability for all points in that plot.

which indicates competition with CSC108, relates increased competition to increased utilization. Once again, the goodness-of-fit values are poor, as shown in Table 7.

Thus, the use of blocking probability provided additional insight regarding the impact of CSC108 on Titan, but just like in Section 4.2, the best-fit lines

Table 7 The table contains the parameter values for the Ordinary Least Squares (OLS) linear regression models regarding blocking probabilities and utilization. The first column corresponds to the figure depicting the model, while the second and third columns correspond the coefficients β_1 and β_0 in the model $y = \beta_1 x + \beta_0$.

Figure	Slope β_1	Intercept β_0	R^2
13a	-0.3766	123.8332	0.1543
13b	-0.1654	103.1603	0.2084
13c	0.0617	86.5830	0.0391
13d	-0.0518	93.6845	0.0370

all displayed very poor goodness-of-fit, rendering the interpretations somewhat weak.

4.4 Summary

Recall that the goal of CSC108 has been to consume idle resources on Titan which would have otherwise gone to waste, while making a good-faith effort not to disturb the rest of Titan's ecosystem.

The results of the compression study in Section 4.1 suggested that CSC108 successfully accomplishes its goal of consuming idle resources which would otherwise have gone to waste, and also they suggested that CSC108 may have an impact on Titan. The results of searching for simple linear relationships in Section 4.2 between indicators like throughput and utilization provided additional insight, but the interpretations were weak statistically because of the poor goodness-of-fit values. Finally, blocking probability was used as a means to identify and analyze times of great competition for resources, but again the interpretations were weak due to poor goodness-of-fit.

These overall results underscore the difficulty of the main problem of this section, which was to identify and analyze the impact of the CSC108 project on Titan. The original hypothesis stated that CSC108 has no effect on Titan. We can see that it has had an impact on Titan in the form of consuming hundreds of millions of core hours. Results suggest that CSC108 negatively impacts Titan by increasing wait times, that CSC108 positively impacts Titan by increasing throughput, and that CSC108 positively impacts Titan by increasing utilization. Interestingly, the inability to find simple relationships by using blocking probability suggests that users' judging system performance by monitoring the batch queue is similarly incapable. In any case, the difficulty in confirming any impact may simply provide evidence that the CSC108 project has impacted Titan minimally, at least with respect to the indicators used.

Finally, we note here that the phenomenon of "draining" on Titan may play a role in some of the counterintuitive results, such as those depicted in Figures 8a, 8b, and 8c. Draining is technically a node state in the Moab scheduler, but it is used here colloquially to refer to the process by which the scheduler allows

busy nodes to finish executing workload before keeping them idle, in order to prepare for a capability class (bin 1) job. During this process, utilization would normally decrease monotonically, but because Titan has enabled backfill scheduling, these idle resources may actually be used for small, short jobs, provided that they will complete before those nodes will be needed for the large job. Because CSC108's consumption increases during times of increased backfill opportunity, the data will show that times of decreased utilization are correlated with increased utilization, unless CSC108 is able to consume all of the backfill opportunity. Even if the project had an infinite supply of new workloads to submit to Titan, it is limited to 20 concurrently executing jobs. Thus, CSC108 will appear to increase in utilization at the same time that all other utilization decreases, even though CSC108 is not actually displacing other projects' workloads. Future work will examine draining in greater detail, to determine if these times have an identifiable signature so that comparisons can be made, in much the same manner as followed in the blocking probability study. This will allow us to understand whether the results of Figure 8d imply that CSC108 should restrict its individual jobs to use only bin 5, for example.

5 Workload Management Beyond HEP

The objective of each subsection is to: (i) describe the science; and (ii) detail what customizations had to be done – either on PanDA or the Titan end to support the science driver. We will then conclude this section with a summary.

5.1 PanDA WMS beyond HEP

Traditionally computing in physics experiments at the basic level is usually independent processing of the input files to produce the output. This processing in referring in the paper as a job. Processing algorithm usually utilizes some experiment-specific software which may require parameterization and even additional configuration files. In the case if such a configuration file is specific for each job it can be defined in a job as another input file. Also experiment software may produce some additional files along with the primary output and they need to be stored. For instance PanDA pilot itself produces the tar-archive file containing the logs its own logs and the experiments software logs. Processing algorithm (referenced as "transformation script") responsible for the correct launching the experiment software and provide all necessary input information including the configuration and run parameters. PanDA job definition is only defines the launching command for the transformation script. This launching command is referring as a payload.

The following components are usually provided and controlled by the experiment groups outside from PanDA core components.

– Transformation scripts. User groups should define a complete set of the transformations scripts to cover all possible SW usage. In the case if the

same software is used and only the run parameters, configuration and input/output file names are changing, the single transformation script should be able to cope this.
- Input/output files conventions. The size of the input files often adjusted in a way to balance of the total processing walltime and flexibility in order to cope the failure risks. There is often case that the equal sized input files are required relatively equal processing time and produce equal sized output. Also input files are often named conventionally and grouped in the datasets by some attributes. PanDA job definition allows to provide name for the input/output datasets.

The real workflow for each scientific group provides a lot of additional requirements and constraints. A common example is a specific order of the jobs execution. Also implementation of the dedicated workflows demands an integration with existing experiment computing infrastructure or even development an additional components. This includes the issues with data management, user authentication, monitoring, workflow control and etc.

PanDA system may be the best solution for the new experiments and scientific groups by diversity of provided advantages. The main motivations for users are:

- Powerful workload management. Automation of the jobs handling, monitoring and logging.
- Streamlining the usage of the computing resources. Possibility for users to run their jobs on diversity of the computing resources. Local resource schedulers, and policies are transparent for the users.
- PanDA native data handling. PanDA provides a diverse set of the plugins to support data stage-in/-out from the remote storages and different data movement tools of different types.
- Close integration with OLCF. Being integrated with OLCF PanDA system also became attractive for many scientific groups already utilizing OLCF resources or those who wish to get use them.

Currently, there are few PanDA instances in use by different experiments and groups. In this paper, we have considered three instances. The original instance is installed at CERN, and it is used exclusively for the ATLAS experiment. Another instance is installed at OLCF, and it is dedicated to supporting projects on Titan, subject to OLCF policies. Finally, an instance on Amazon's EC2 cloud infrastructure provides access to multiple independent experiments from different disciplines, and it has the least restrictive security and usage policies.

5.2 PanDA instance at OLCF

In March 2017, we implemented a new PanDA server instance within OLCF operating under Red Hat OpenShift Origin [18] - a powerful container cluster management and orchestration system in order to serve various experiments

at Titan supercomputer. By running on-premise Red Hat OpenShift built on Kubernetes [19], the OLCF provides a container orchestration service that allows users to schedule and run their HPC middleware service containers while maintaining a high level of support for many diverse service workloads. The containers have direct access to all OLCF shared resources such as parallel filesystems and batch schedulers. With this PanDA instance, we implemented a set of demonstrations serving diverse scientific workflows including physics, biology studies of the genes and human brain, and molecular dynamics studies:

- Biology / Genomics. In collaboration with Center for Bioenergy Innovation at ORNL the PanDA based workflow for epistasis researches was established. Epistasis is the phenomenon where the effect of one gene is dependent on the presence of one or more "modifier genes", i.e. the genetic background. GBOOST [20] is a GPU-based tool for detecting gene-gene interactions in genome-wide case control studies, was used for initial tests.
- Molecular Dynamics. In collaboration with the Chemistry and Biochemistry department of the University of Texas Arlington, we implemented a test to try out PanDA to support the Molecular Dynamics study "Simulating Enzyme Catalysis, Conformational Change, and Ligand Binding/Release". The CHARMM (Chemistry at HARvard Macromolecular Mechanics) [21] a molecular simulation program was chosen as a basic payload tool. CHARMM design for hybrid MPI/OpenMP/GPU computing.
- IceCube. Together with experts from the IceCube experiment we implemented the demonstrator PanDA system. IceCube [22] is a particle detector at the South Pole that records the interactions of a nearly massless subatomic particle called the neutrino. Demonstrator includes the use of NuGen package (a modified version of ANIS [23] that works with IceCube software) - GPU application for atmospheric neutrinos are simulations packed in Singularity container and remote stage-in/-out the data from GridFTP [24] storage with GSI authentication.
- BlueBrain. In 2017, a R&D project was started between BigPanDA and the Blue Brain Project (BBP) [25] of the Ecole Polytechnique Federal de Lausanne (EPFL) located in Lausanne, Switzerland. This proof of concept project is aimed at demonstrating the efficient application of the BigPanDA system to support the complex scientific workflow of the BBP which relies on using a mix of desktop, cluster, and supercomputers to reconstruct and simulate accurate models of brain tissue. In the first phase of this joint project we supported the execution of BBP software on a variety of distributed computing systems powered by BigPanDA. The targeted systems for demonstration included: Intel x86-NVIDIA GPU based BBP clusters located in Geneva (47 TFlops) and Lugano (81 TFlops), BBP IBM BlueGene/Q supercomputer [26] (0.78 PFLops and 65 TB of DRAM memory) located in Lugano, the Titan Supercomputer with peak theoretical performance 27 PFlops operated by the Oak Ridge Leadership Computing Facility (OLCF), and Cloud based resources such as Amazon Cloud.

- LSST. A goal of LSST (Large Synoptic Survey Telescope) project is to conduct a 10-year survey of the sky that is expected to deliver 200 petabytes of data after it begins full science operations in 2022. The project will address some of the most pressing questions about the structure and evolution of the universe and the objects in it. It will require a large amount of simulations, which model the atmosphere, optics and camera to understand the collected data. For running LSST simulations with the PanDA WMS we have established a distributed testbed infrastructure that employs the resources of several sites on GridPP [27] and Open Science Grid (OSG) [28] as well as the Titan supercomputer at ORNL. In order to submit jobs to these sites we have used a PanDA server instance deployed on the Amazon AWS Cloud.
- LQCD. Lattice QCD (LQCD) [29] is a well-established non-perturbative approach to solving the quantum chromodynamics theory of quarks and gluons. Current LQCD payloads can be characterized as massively parallel, occupying thousands of nodes on leadership-class supercomputers.
  In 2017, as a part of SciDAC-4 funded project, a collaboration was formed between several US LQCD groups and BigPanDA team with the goal to adopt PanDA WMS for the needs of the SciDAC-4 LQCD computational program.
  LQCD payloads have been successfully tested on Titan as well as on other sites. Production campaigns were executed on BNL Institutional Cluster through a dedicated instance of Harvester installed on the front node of this site. During the period between April and June 2018 13 TB of input data were processed, producing output of 176 GB. LQCD jobs used around 15,000 GPU hours with average job duration around 12 hours.
- nEDM. Precision measurements of the properties of the neutron present an opportunity to search for violations of fundamental symmetries and to make critical tests of the validity of the Standard Model of electroweak interactions. These experiments have been pursued [30] with great energy and interest since the discovery of neutron in 1932. The goal of the nEDM [31] experiment at the Fundamental Neutron Physics Beamline at the Spallation Neutron Source (Oak Ridge National Laboratory) is to further improve the precision of this measurement by another factor of 100.

To isolate the workflows of different groups and experiments, dedicated queues were defined at the PanDA server. Presumably in next steps we will provide the security mechanisms that will provide the access to each queue for job submission and dispatching only for authorised users. Also, the PanDA server provides the tools to customise environment variables, system settings and workflow algorithms for different user groups. Also this split of the different groups workflows on the level of PanDA queues simplifies jobs monitoring via the web based PanDA tool.

In collaboration with the dedicated scientific groups representatives, we implemented the “transformation” scripts containing complete definition of the processing actions (set of specific software and general system commands)

Table 8 Please write your table caption here

Experiment	Payload	Jobs	Nodes	Walltime	Input data	Output data
Genomics	GBOOST	10	2	30 min	100 MB	300 MB
Molecular Dynamics	CHARMM	10	124	30-90 min	10 KB	2-6 GB
IceCube	NuGen	4500K	1	120 min	500 KB	10KB - 4GB
LSST/DESC	Phosim	20	2	600 min	700 MB	70 MB
LQCD	QDP++	10	8000	700 min	40 GB	150 MB
nEDM	GEANT	10	200	20 min	120 MB	20 MB

are has to be applied to the input data to produce the output. The transformation script then can be addressed by its name. Client tool provided to the users allows to submit jobs to the PanDA server with authentication based on grid certificates.

Responsible group representative also authorized to run pilots launcher daemon. Daemon launches the pilots. Number of parallel running pilots can be configured. Pilots are running and interacts with the PBS under user account and with Titan group privileges of the responsible representative.

The most important parameters of conducted tests are presented in the table

5.3 Summary

The overview of the successfully implemented demonstrations of diverse workflows implementation via PanDA shows that PanDA model can cope the challenges of the different experiments and user groups and also provide possibility for extensions beyond the core components set. The proof of concept was received from all considered experiments representatives and results that PanDA is considered as a possible solution. Preproduction utilization of PanDA is now under investigation with BlueBrain, IceCube, LSST, nEDM experiments, LQCD uses PanDA for Production.

Acknowledgements If you'd like to thank anyone, place your comments here.

References

1. C. Marco, C. Fabio, D. Alvise, G. Antonia, G. Francesco, M. Alessandro, M. Moreno, M. Salvatore, P. Luca, P. Francesco, in *International Conference on Grid and Pervasive Computing* (2009), pp. 256–268
2. M. Turilli, M. Santcroos, S. Jha, ACM Computing Surveys (accepted, in press), arXiv preprint arXiv:1508.04180v3 (2017)
3. I. Foster, C. Kesselman, S. Tuecke, International journal of high performance computing applications **15**(3), 200 (2001)
4. V. Garonne, G.A. Stewart, M. Lassnig, A. Molfetas, M. Barisits, T. Beermann, A. Nairz, L. Goossens, F.B. Megino, C. Serfon, et al., in *J. Phys.: Conf. Ser.*, vol. 396 (2012), vol. 396, p. 032045

5. T. Maeno, K. De, T. Wenaus, P. Nilsson, G. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, D. Smith, et al., in *J. Phys.: Conf. Ser.*, vol. 331 (2011), vol. 331, p. 072024
6. M. Borodin, K. De, J. Garcia, D. Golubkov, A. Klimentov, T. Maeno, A. Vaniachine, et al., in *J. Phys.: Conf. Ser.*, vol. 664 (2015), vol. 664, p. 062005
7. P. Nilsson, J. Caballero, K. De, T. Maeno, A. Stradling, T. Wenaus, A. Collaboration, et al., in *J. Phys.: Conf. Ser.*, vol. 331 (2011), vol. 331, p. 062040
8. P. Nilsson, M. Potekhin, T. Maeno, J. Caballero, K. De, T. Wenaus, PoS p. 027 (2008)
9. J. Caballero, J. Hover, P. Love, G. Stewart, in *J. Phys.: Conf. Ser.*, vol. 396 (2012), vol. 396, p. 032016
10. A. Klimentov, P. Nevski, M. Potekhin, T. Wenaus, in *J. Phys.: Conf. Ser.*, vol. 331 (2011), vol. 331, p. 072058
11. A. Anisenkov, A. Di Girolamo, A. Klimentov, D. Oleynik, A. Petrosyan, A. Collaboration, et al., in *Journal of Physics: Conference Series*, vol. 513 (IOP Publishing, 2014), vol. 513, p. 032001
12. P. Calafiura, K. De, W. Guan, T. Maeno, P. Nilsson, D. Oleynik, S. Panitkin, V. Tsulaia, P. Van Gemmeren, T. Wenaus, in *J. Phys.: Conf. Ser.*, vol. 664 (2015), vol. 664, p. 062065
13. M. Borodin, S. Padolski, T. Wenaus, T. Maeno, K. De, D. Golubkov, R. Mashinistov, F.H. Barreiro Megino, A. Klimentov, Atlas production system. Tech. rep., ATL-COM-SOFT-2016-021 (2016)
14. A.P. Team. The PanDA production and distributed analysis system (2017). URL `https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA`
15. OLCF. Titan scheduling policy (2018). URL `https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs`
16. M.A. Ezell, D.E. Maxwell, D. Beer, Proc. CUG
17. R. Team. RADICAL-SAGA software toolkit (2017). URL `https://github.com/radical-cybertools/radical-saga`
18. Red hat openshift web site. `https://www.openshift.com`. Accessed: 2019-01-18
19. Kubernetes web site. `https://kubernetes.io/`. Accessed: 2019-01-18
20. L. Sing Yung, C. Yang, X. Wan, W. Yu, **27**, 1309 (2011)
21. B.R. Brooks, C.L. Brooks, A.D. Mackerell, L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A.R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R.W. Pastor, C.B. Post, J.Z. Pu, M. Schaefer, B. Tidor, R.M. Venable, H.L. Woodcock, X. Wu, W. Yang, D.M. York, M. Karplus, Journal of Computational Chemistry **30**(10), 1545 (2009). DOI 10.1002/jcc.21287. URL `http://dx.doi.org/10.1002/jcc.21287`
22. F. Halzen, S.R. Klein, Rev. Sci. Instrum. **81**, 081101 (2010). DOI 10.1063/1.3480478
23. A. Gazizov, M.P. Kowalski, Comput. Phys. Commun. **172**, 203 (2005). DOI 10.1016/j.cpc.2005.03.113
24. W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing* (IEEE Computer Society, Washington, DC, USA, 2005), SC '05, pp. 54–. DOI 10.1109/SC.2005.72. URL `https://doi.org/10.1109/SC.2005.72`
25. H. Markram, **7**, 153 (2006)
26. H. Markram, Nat Rev Neurosci **7**(2), 153 (2006). DOI 10.1038/nrn1848. URL `http://dx.doi.org/10.1038/nrn1848`
27. T.G. Collaboration, P.J.W. Faulkner, L.S. Lowe, C.L.A. Tan, P.M. Watkins, D.S. Bailey, T.A. Barrass, N.H. Brook, R.J.H. Croft, M.P. Kelly, C.K. Mackay, S. Metson, O.J.E. Maroney, D.M. Newbold, F.F. Wilson, P.R. Hobson, A. Khan, P. Kyberd, J.J. Nebrensky, M. Bly, C. Brew, S. Burke, R. Byrom, J. Coles, L.A. Cornwall, A. Djaoui, L. Field, S.M. Fisher, G.T. Folkes, N.I. Geddes, J.C. Gordon, S.J.C. Hicks, J.G. Jensen, G. Johnson, D. Kant, D.P. Kelsey, G. Kuznetsov, J. Leake, R.P. Middleton, G.N. Patrick, G. Prassas, B.J. Saunders, D. Ross, R.A. Sansum, T. Shah, B. Strong, O. Synge, R. Tam, M. Thorpe, S. Traylen, J.F. Wheeler, N.G.H. White, A.J. Wilson, I. Antcheva, E. Artiaga, J. Beringer, I.G. Bird, J. Casey, A.J. Cass, R. Chytracek, M.V.G. Torreira, J. Generowicz, M. Girone, G. Govi, F. Harris, M. Heikkurinen, A. Horvath, E. Knezo,

M. Litmaath, M. Lubeck, J. Moscicki, I. Neilson, E. Poinsignon, W. Pokorski, A. Ribon, Z. Sekera, D.H. Smith, W.L. Tomlin, J.E. van Eldik, J. Wojcieszuk, F.M. Brochu, S. Das, K. Harrison, M. Hayes, J.C. Hill, C.G. Lester, M.J. Palmer, M.A. Parker, M. Nelson, M.R. Whalley, E.W.N. Glover, P. Anderson, P.J. Clark, A.D. Earl, A. Holt, A. Jackson, B. Joo, R.D. Kenway, C.M. Maynard, J. Perry, L. Smith, S. Thorn, A.S. Trew, W.H. Bell, M. Burgon-Lyon, D.G. Cameron, A.T. Doyle, A. Flavell, S.J. Hanlon, D.J. Martin, G. McCance, A.P. Millar, C. Nicholson, S.K. Paterson, A. Pickford, P. Soler, F. Speirs, R.S. Denis, A.S. Thompson, D. Britton, W. Cameron, D. Colling, G. Davies, P. Dornan, U. Egede, K. Georgiou, P. Lewis, B. MacEvoy, S. Marr, J. Martyniak, H. Tallini, S. Wakefield, R. Walker, I.A. Bertram, E. Bouhova-Thacker, D. Evans, R.C.W. Henderson, R.W.L. Jones, P. Love, S. Downing, M.P. George, A.C. Irving, C. McNeile, Z. Sroczynski, M. Tobin, A.J. Washbrook, R.J. Barlow, S. Dallison, G. Fairey, A. Forti, R.E. Hughes-Jones, M.A.S. Jones, S. Kaushal, R. Marshall, A. McNab, S. Salih, J.C. Werner, V. Bartsch, C. Cioffi, P. Gronbech, N. Harnew, J.F. Harris, B.T. Huffman, M. Leslie, I. McArthur, R. Newman, A. Soroko, I. Stokes-Rees, S. Stonjek, J. Tseng, D. Waters, G. Wilkinson, T.R. Arter, R.A. Cordenonsi, A.S. Datta, T. Hartin, S.L. Lloyd, A.J. Martin, S.E. Pearce, C.J. Williams, M. Gardner, S. George, B.J. Green, S. Johal, G. Rybkine, J.A. Strong, P. Teixeira-Dias, P. Hodgson, M. Robinson, D.R. Tovey, N.J.C. Spooner, C.R. Allton, W. Armour, P. Clarke, P. Mealor, D. Waters, B. Waugh, B. West, Journal of Physics G: Nuclear and Particle Physics **32**(1), N1 (2006). URL `http://stacks.iop.org/0954-3899/32/i=1/a=N01`

28. R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wrthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, Journal of Physics: Conference Series **78**(1), 012057 (2007). URL `http://stacks.iop.org/1742-6596/78/i=1/a=012057`
29. R. Babich, M.A. Clark, B. Joó, in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE Computer Society, Washington, DC, USA, 2010), SC '10, pp. 1–11. DOI 10.1109/SC.2010.40. URL `https://doi.org/10.1109/SC.2010.40`
30. A.D. Sakharov, Pisma Zh. Eksp. Teor. Fiz. **5**, 32 (1967). DOI 10.1070/PU1991v034n05ABEH002497. [Usp. Fiz. Nauk161,no.5,61(1991)]
31. S.K. Lamoreaux, R. Golub, Journal of Physics G: Nuclear and Particle Physics **36**(10), 104002 (2009). URL `http://stacks.iop.org/0954-3899/36/i=10/a=104002`