EPJ Web of Conferences **214**, 03024 (2019) https://doi.org/10.1051/epjconf/201921403024
CHEP 2018

Running IceCube GPU simulations on Titan

Vladimir Brik[1,*], *David* Schultz[1], and *Gonzalo* Merino[1]

[1]Wisconsin IceCube Particle Astrophysics Center, University of Wisconsin-Madison, 222 W Washington Ave, Suite 500, Madison, WI 53703, USA

Abstract. Here we report IceCube's first experiences of running GPU simulations on the Titan supercomputer. This undertaking was non-trivial because Titan is designed for High Performance Computing (HPC) workloads, whereas IceCube's workloads fall under the High Throughput Computing (HTC) category. In particular: (i) Titan's design, policies, and tools are geared heavily toward large MPI applications, while IceCube's workloads consist of large numbers of relatively small independent jobs, (ii) Titan compute nodes run Cray Linux, which is not directly compatible with IceCube software, and (iii) Titan compute nodes cannot access outside networks, making it impossible to access IceCube's CVMFS repositories and workload management systems. This report examines our experience of packaging our application in Singularity containers and using HTCondor as the second-level scheduler on the Titan supercomputer.

1 Introduction

The IceCube Neutrino Observatory is a neutrino detector located at the South Pole [1]. The IceCube Collaboration studies the nearly massless subatomic particles called neutrinos of extra-terrestrial origin. Our research requires a great deal of computational resources, in particular resources that provide GPU computing capabilities.

Titan is a Cray XK7 supercomputer at Oak Ridge Leadership Computing Facility (ORLCF) [2]. It has been ranked #7 supercomputer in the world by TOP500 [3] as of June 2018, and it consists of 18,688 compute nodes each equipped with 16-core AMD Opteron CPU, 32GB RAM, and a Kepler K20X GPU. Titan uses TORQUE [4], Moab [5], and ALPS [6] for cluster management and operation.

IceCube has been granted a test allocation of 1 million "Titan-hours" (about 33 thousand GPU hours) under the "ORLCF director's discretionary allocation" program. Running IceCube jobs on Titan presented an unusual challenge for us, primarily because (1) worker nodes have no access to the Internet, (2) worker nodes' software and hardware are atypical, and (3) the system as a whole is designed and optimized for running large MPI jobs.

IceCube GPU simulation workloads generally consist of large numbers of independent single-GPU processes whose running times vary widely and cannot be predicted in advance. Such workloads are not a natural fit for Titan, whose Portable Batch System (PBS) scheduling policy is designed to strongly prefer large MPI jobs. Native scheduling tools on Titan are not designed to efficiently handle large numbers of tasks of highly-variable and unpredictable durations. Therefore, we decided to use HTCondor [7] as the second-level scheduler.

*Corresponding author: vbrik@icecube.wisc.edu

Instead of recompiling IceCube's software stack for Titan's Cray OS, which would have been a daunting task, we decided to use Singularity [8] containers to solve both the problem of software compatibility and software distribution.

2 Workload

A set of GPU-based photon propagation simulations were chosen as the workload for Titan. The simulations were part of an effort to collect information about the detector systematics, in particular understanding the effect of the so-called "hole ice", which is the column of ice that formed in the drill holes during the construction of the detector, on the particle energy and direction measured by IceCube. One of the end goals is to generate pre-computed models that will save GPU compute time in the future.

From the technical perspective, the chosen workload was a good fit for Titan because (1) all input files could be pre-generated and manually uploaded to Titan, (2) the workflow was simple, with all simulations being self-contained and straightforward to execute, (3) computational requirements were a good fit for our capabilities on Titan.

A "simulation" in this context is a single command consisting of a script and its arguments that would use a single GPU to perform computations. All simulation commands followed a simple pattern and could be generated in advance. The simulation scripts performed GPU simulations using standard IceCube software and data available in Open Science Grid's [9] CVMFS [10] repository and some custom code.

Run times for all simulations followed a similar long-tailed distribution, with majority of simulations finishing relatively quickly, and a significant number of long simulations. Simulation run times could not be accurately estimated beforehand. The actual distribution of the run times is discussed in more detail in Section 4.

3 Approach

The main challenges of running the simulation workload on Titan can be classified across several dimensions: Creating an execution environment suitable for execution of simulation scripts. Making IceCube's software and data available to the simulation scripts. Scheduling and managing execution of variable-duration simulations inside fixed-duration Titan jobs. Transferring input and output data to and out of Titan.

At a high level, we took the following approach. Transfer of input and output files was done manually. A Singularity container was constructed, which could execute the simulation scripts and run HTCondor daemons. Then, we iterated the following process:

1. A PBS job would be submitted and would initiate the "orchestration scripts".
2. The orchestration scripts would use the container to set up an HTCondor pool, whose state (journal, history, etc.) was stored on the Titan's shared Lustre file system [11].
3. On the central manager/submitter of the pool, the orchestration scripts would either submit jobs using a pre-generated submit file, and/or load HTCondor state from Lustre.
4. Simulations would be executed by HTCondor inside its container.
5. If the PBS job terminated before (almost) all simulations completed, the process would be repeated from Step 1, except the with a smaller PBS job (to reflect the reduced number of remaining simulations), and without providing an HTCondor submit file (since HTCondor state of the pool's previous incarnation would be loaded from Lustre).

6. As the final step of the campaign, the simulations whose output files were missing or corrupt were re-run.

Since our workload was comprised of a large number of independent simulations, we arbitrarily divided them into groups, and applied the above process to each group. This allowed us to take an iterative approach to debugging and fixing problems, and limited the amount of wasted allocation time a single malfunction could inflict. The trade-off was increased human overhead and more complicated log analytics.

Details of the approach undertaken are presented in the following subsections.

3.1 Singularity Container

A custom Singularity container image was created to execute simulation scripts and run HTCondor on Titan. The container was comprised of:

1. Scientific Linux 6.9 base packages.
2. A 36GB subset of IceCube's Open Science Grid CVMFS repository.
3. The custom code required by the simulation scripts.
4. Titan-specific Singularity configuration [12].
5. HTCondor 8.7.6 with a small custom configuration file.
6. Miscellaneous Titan-specific fixes and workarounds.
7. Tools for monitoring and debugging.

We have chosen to package IceCube software and data inside the container due to concerns that putting large number of small files that may be accessed simultaneously by hundreds of worker nodes may cause problems for Titan's shared Lustre file system.

Most of HTCondor configuration was stored on the shared file system. This way, tweaks to HTCondor configuration did not require changes to the container, which saved a lot of time.

3.2 Workload Execution

The workload was executed in an HTCondor pool created inside a PBS job ("reservation").

The following procedure was used to actually run simulations on Titan. Note that simulations were executed iteratively, in "stages", between a few hundred and 25,000 simulations per stage, with several PBS jobs per stage.

1. An HTCondor submit file was generated for the simulations to be executed in the given stage (one HTCondor job per simulation).
2. A PBS job was submitted. It was sized (in terms of number of nodes and walltime limit) roughly to be able to complete the set of simulations in the stage. The script that submitted the PBS reservation took as arguments (1) a path to the "pool directory" on the Lustre shared file system where HTCondor state files, logs, as well as files used for debugging, monitoring, and accounting were kept, and (2), optionally, a path to the HTCondor submit file.

3. Once the PBS job was started, it would run the main PBS script that orchestrated the setup of an HTCondor pool inside the PBS reservation, optionally submit HTCondor jobs to the pool, and implement a mechanism for shutdown of the pool. The simulations would then execute in the HTCondor pool until the PBS reservation was terminated, the pool ran out of jobs, or it was commanded to shutdown, or a problem was detected.

The main HTCondor pool orchestration script used multiple invocations of Titan's aprun command to start HTCondor in containers on available execute nodes. Up to 21 apruns were started, depending on the size of the PBS job. Although up to 50 apruns are permitted on a Titan service node, a process count `ulimit` of 200 prevented us from going higher.

HTCondor daemons running on worker nodes were configured using environmental variables set by the orchestration scripts and a global pool-specific configuration file created by the orchestration scripts in the pool directory. The configuration file contained certain optimizations, such as an infinite `CLAIM_WORKLIFE`.

Typically, the PBS job would complete when the orchestrator scripts self-shutdown either in anticipation of hitting the walltime limit, or a predefined amount of time after the pool ran out of idle jobs. The remaining simulations would be completed by submitting a smaller PBS job, this time without a submit file. The intention was to minimize time "wasted" by idle workers (see Section 5 for reasoning).

The orchestration scripts used files in the pool directory to communicate the state of the pool and perform coordinated emergency shutdown that was cleaner than just deleting the PBS job. For example: if the pool's queue ran out of idle jobs, a file called `pool_no_idle_jobs` would be created; if a file called `pool_kill` was detected, all worker nodes would independently shut themselves down. Similar functionality could be implemented by other means, but, given Titan's idiosyncrasies, this approach was chosen for its simplicity.

A fail-safe monitoring script was implemented to trigger pool shutdown if HTCondor on the central manager malfunctioned. The script ran in the central manager container (a better design is discussed in Section 5).

To minimize the likelihood of the central manager malfunctioning, it was started by a dedicated aprun process on an execute node that did not run any simulations. The central manager could not run on the PBS service node due to severe `ulimit` constraints on the number of processes and open files.

3.3 Operations and Monitoring

Real-time resource utilization on each node was monitored and logged to the Lustre shared file system. These files could be watched from any of Titan's interactive nodes.

When necessary, interactive monitoring and operation of the central manager and worker nodes was done by logging in to those containers using ssh from a service node.

Since HTCondor state was stored in Lustre, containerized utilities such as `condor_q` could be used to examine the state of the pool from service nodes (it could not be done from login nodes since Singularity was not available there).

3.4 Data Transfer

Data transfer was accomplished out-of-band, manually, using `globus-url-copy`. Since the files on Titan's Lustre file system are periodically purged, output data was uploaded to IceCube storage periodically.

4 Analysis of Results

At the time of writing, 83,988 simulations were executed, producing 2.8TB of output data, and expending 494,889 Titan-hours, which is equal to 16,496 GPU-hours.

About 0.4% of output files were obviously invalid because they could not be parsed, or were zero-size. Log analysis showed that most were the result of simulations being killed by HTCondor because they ran too long or used too much memory. Detection of invalid files was performed outside of Titan using a simple script. Since processing time and memory requirements of individual simulations are deterministic, we did not attempt to re-process the invalid files on Titan.

According to Titan's accounting, overall, GPUs were in use about 90% of the time. The following sources of 10% inefficiency can be identified: (1) The time it takes the orchestrator script to bring up the worker nodes. For some jobs, the time it takes to run `condor_submit` (a possibly non-trivial overhead if a lot of jobs are involved, not helped by the fact that `condor_schedd` log was stored on Lustre). (2) Time it takes HTCondor to dispatch first jobs to all worker nodes. (3) Job scheduling overhead. (4) Simulation scripts' start-up overhead. (5) Output compression and write-out. (6) Some worker nodes sitting idle if the pool ran out of jobs. (7) Worker node failures.

According to HTCondor accounting, 5% of time was spent re-running interrupted jobs.

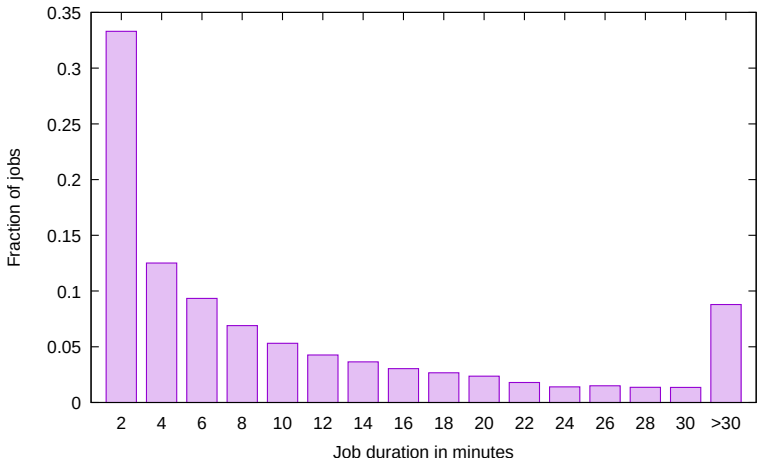

Figure 1. Distribution of job durations

HTCondor's `CommittedTime` distribution, based on a normalized sub-sample of jobs that completed successfully, is presented in Figure 1. Note that the rightmost bucket is much larger than the rest. The average job duration of successful simulations was about 10 minutes, and the median was a little under 5 minutes. About 10% of jobs were longer than 30 minutes, and about 1% of jobs were longer than 1 hour.

5 Issues Encountered

Issues with instability were encountered, some of which remain unsolved and/or poorly understood. All of these involved difficult-to-reproduce malfunctions of containerized processes. The worker initialization script would fail with "File not found" error on an existing

file. The HTCondor worker or central manager setup scripts would fail silently. Some initialization scripts would fail with a segmentation fault in unexpected places. Simulation scripts, after a period of normal operation, would unexpectedly and repeatedly begin to segfault for no apparent reason, turning the worker node into a job black hole.

It is likely that many of these have been caused by Titan's Lustre file system being unable to handle a large number of worker nodes concurrently issuing many small I/O operations. This may partially explain the higher likelihood of errors during initialization of larger pools (800+ nodes). To mitigate this, worker node initialization has been splayed over time, frequency of actions involving small I/O operations by the orchestration scripts has been reduced, and non-essential HTCondor logs have been moved off Lustre.

Other errors were probably a result of inconsistent container state caused by worker nodes running out of memory, but not killing the entire container. This was mitigated by lowering the threshold of memory usage at which a job is killed cleanly by HTCondor.

6 Lessons Learned

Perhaps the most important and potentially impactful "lesson" is the confirmation that it could be highly advantageous to be able to operate an HTCondor central manager that is separate from the PBS jobs that start-up HTCondor worker nodes. This would greatly simplify orchestration scripts and HTCondor job management, improve reliability, and reduce wait times for Titan resources. For example, the central manager could be a longer-running container, possibly with access to non-Lustre storage, and not subjected to the severe `ulimit` restrictions imposed on the processes running on service nodes.

6.1 Titan Environment

In limited testing using simulations of our workload, GeForce GTX 1080 GPUs were found to be about 5 times faster than Titan's Kepler K20x GPUs.

TCP/IP communication between execute nodes and service nodes over Titan's Infiniband network is possible. This is not documented in Titan's User Guide and initially there were doubts if it would work.

Job wait times on Titan were highly unpredictable. At times there would be thousands of unused nodes available for many hours, at other times a job requesting about 1000 nodes for a few hours would spend almost a day waiting in the queue.

PBS job history that is accessible via command line tools is purged every 3 days. However, older historical data, including per-job GPU utilization, can be requested by contacting support.

6.2 Workload Characteristics

A number of issues have been caused by unpredictable, highly variable simulation run times, compounded by their unwieldy, long-tailed distribution. This interacted poorly with Titan's inflexible PBS jobs and billing model. At a certain point, an HTCondor pool would run out of idle jobs to schedule. Some worker nodes would begin to run out of tasks and sit idle as a result, while others worked on their simulations, resulting in wasted allocation time, since each reserved worker, idle or busy, has the same cost. Thus, there is a trade-off to be made: shutdown the pool early, and avoid paying for idle workers, but lose progress made on still running simulation (and re-run them later), or shutdown late, avoiding redoing too much of the running simulations later, but pay for potentially large number of idle workers. The unfavorable nature of the runtime distribution made the optimal decision non-obvious.

6.3 Application Design

Since container development iteration speed was slow due to its size, during initial development and testing, when changes to the container were made often, the container was stored unpacked as a directory on Titan. This saved a lot of time.

It is important to have fail-safe mechanisms, and to design orchestration scripts to terminate the PBS job if the pool encounters serious problems in order to minimize the amount of allocation wasted. Unfortunately, stability issues have only been partially solved at this point, and there remains a non-trivial probability of a pool failing to start properly, or various sorts of worker node failures. A possible improvement to the current situation would be to run more sophisticated independent fail-safe scripts on both the central manager and the service node, since they each have different failure modes.

7 Conclusion

We conclude that a specialized HPC system like Titan can be effectively used for HPC particle physics simulation workloads such as IceCube's. Support for Singularity containers greatly simplified adopting our application to the new environment. Significant resource utilization gains would be possible if we had the capability to run HTCondor central manager as a long-lived service accessible from Titan's internal network.

8 Acknowledgements

We acknowledge the support from the following agencies: U.S. National Science Foundation-Office of Polar Programs, U.S. National Science Foundation-Physics Division, University of Wisconsin Alumni Research Foundation.

References

[1] Halzen, "IceCube A Kilometer-Scale Neutrino Observatory at the South Pole", IAU XXV General Assembley, Sydney, Australia, 13-26 July 2003, ASP Conf. Series vol 13, p 13-16

[2] Titan supercomputer at Oak Ridge Leadership Computing Facility, https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan

[3] TOP500 Supercomputer Sites, https://www.top500.org

[4] TORQUE Resource Manager, http://www.adaptivecomputing.com/products/torque/

[5] Moab HPC Suite, http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-suite-enterprise-edition

[6] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing, "The application level placement scheduler", Cray User Group, 2008.

[7] HTCondor project, "HTCondor" [software], version 8.7.6, 2018. Available from https://research.cs.wisc.edu/htcondor/ [accessed 2018-10-03]

[8] Singularity containers project, "Singularity" [software], version 2.4.1 2018, Available from https://github.com/sylabs/singularity [accessed 2018-10-03]

[9] R. Pordes et al, "Open Science Grid", J. Phys.: Conf. Ser. 78 012057 (2007)

[10] J. Blomer, P. Buncic, T. Fuhrmann, "CernVM File System project", Proc. of the 1st int. workshop on Network-aware data management (NDM'11) 49–56 (2011)

[11] Schwan, Philip, "Lustre: Building a file system for 1000-node clusters", Proceedings of the 2003 Linux Symposium, vol. 2003.

[12] Singularity Tools for Titan, [software], https://github.com/olcf/SingularityTools/ [accessed 2018-10-03]