EPJ Web of Conferences **116**, 07003 (2016)
DOI: 10.1051/epjconf/201611607003

IceCube simulation production and the transition to IceProd2

David Schultz[a]

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

Abstract. IceCube's simulation production relies largely on dynamic, heterogeneous resources spread around the world. Datasets consist of many thousands of job workflow subsets running in parallel as directed acyclic graphs (DAGs) and using varying resources. IceProd is a set of Python daemons which process job workflow and maintain configuration and status information on jobs before, during, and after processing. IceProd manages a complex workflow of DAGs to distribute jobs across all computing grids and optimize resource usage. IceProd2 is a new version of IceProd with substantial increases in security, reliability, scalability, and ease of use. It is undergoing testing and will be deployed this fall.

1. Introduction

IceProd has served the IceCube collaboration well over the last 8 years, processing Monte Carlo simulations, detector data, and analysis levels [1]. It has run thousands of CPU-core years and stored over 2PB of data. But it has experienced its share of issues and the initial software is running into fundamental design limitations. This article describes that history and the development of a second version of IceProd. The new version is a complete rewrite of the code base, fixing long-standing issues and preparing for the increased load of the next few years.

2. IceCube and computing

The IceCube collaboration uses two main data centers, one in North America (University of Wisconsin-Madison) and one in Europe (DESY Zeuthen). Other institutions generally only have limited access to shared campus resources or smaller compute clusters. Additionally, there is significant availability of opportunistic resources via providers like the Open Science Grid [2]. Currently, over 50% of all CPU resources are opportunistic. One of the highlights of the IceCube simulation chain is the direct propagation of photons using GPU accelerators [3]. Over the last 3 years IceCube has deployed more than 500 GPUs to help run this part of the simulation chain.

The IceCube simulation and data processing jobs have a large variability in resource requirements. Although most simulation jobs need between 2–4 GB of memory, several types of jobs require 10+ GB of memory. Certain simulation generation jobs require significant scratch disk, on the order of

[a] e-mail: david.schultz@icecube.wisc.edu

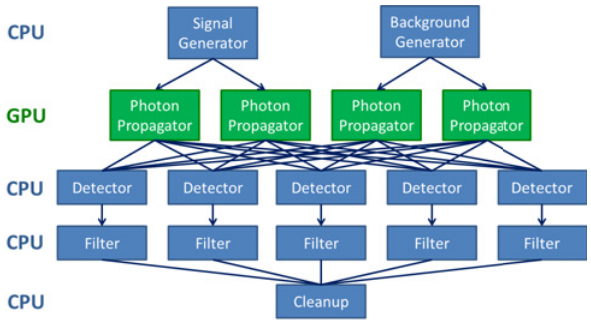

Figure 1. Simulation job streams contain CPU and GPU jobs running at different sites. This DAG has many interdependent jobs making up one "stream."

100 GB, to manage temporary files. Finally, one of the problems with Monte Carlo simulations is that rare events, such as extremely high energy events, can take an order of magnitude more resources, especially walltime. Handling these randomly occurring events is an ongoing issue.

Based on this information, requirements have become apparent [4]. Because the resources aren't owned, anything that requires root permission is unlikely to work. The mix in resources leads to multiple OS types and schedulers that need to be supported. Many of the smaller sites cannot handle glideinWMS, an easy way to join multiple sites into a global pool [5]. This points to a light-weight and flexible approach, with needs not met by other workflow systems.

2.1 IceProd job workflow

The core software framework of IceCube, called IceTray, is stream-based with modules that act on events in the stream [6]. Simulation jobs run similarly, with a series of modular jobs forming a stream (or multiple streams) of data that is incrementally processed to further levels. A series of these streams can run in parallel with different random seeds, and forms a dataset. It is typical to have up to a few hundred datasets running concurrently.

The technical term for the individual job stream is a directed acyclic graph (DAG). In Fig. 1 a more complex DAG can be seen, where two simulation generators run on CPUs, feeding four GPU jobs, which in turn feed their output to five parallel CPU jobs. These complex DAGs are used in real production, but they currently create significant scheduling overhead with extra database lookups to check dependency status; thus, they are only used when really needed. Simpler DAGs are preferred.

3. IceProd v1

The initial version of IceProd is a set of user-space site-local Python daemons backed by a central database [7]. All knowledge of state is stored in the database, with the Python daemons interacting with the queueing system or the jobs directly. One daemon submits new jobs and keeps track of the queue. Another daemon listens for XML-RPC from running jobs so their status can be updated as the job runs through its lifecycle. A third daemon is responsible for any cleanup tasks that are required, and a final daemon responds to user commands such as dataset submission, suspension, or deletion. A web interface was developed in PHP that interfaced with the database and displayed its contents in an easy to read format, along with forwarding user commands to the IceProd daemon to control datasets. Plugins have been designed for each batch system or other job submission interface. This turns a multitude of

heterogeneous systems into a homogeneous API that IceProd can interact with. It also allows for new systems to be incorporated over time with minimal effort.

One of the major additions to IceProd came with the use of GPUs, making it necessary to break up jobs into CPU and GPU parts. Initially, with GPUs in a testing phase, this was accomplished locally at a few sites with DAGMan for HTCondor [8]. This worked fairly reliably, but did not extend to other job submission systems and did not allow jobs from one site to utilize GPUs at another site. The DAG feature was integrated directly into IceProd two years ago and solved cross-site linking.

When IceProd was first designed 8 years ago computing clusters were generally small, only a few hundred nodes at most, and with jobs taking hours to complete the queue rate was fairly slow. Today shorter jobs and clusters of thousands of nodes are showing bottlenecks in multiple places of IceProd v1. With jobs being individually queued and typically taking a second or more to queue, we are reaching the situation where constantly queueing can't saturate a cluster. The move to DAGs of jobs has led to a substantial rise in the number of queries on the database. At times the sheer number of queries per second is enough to slow things down, and we have to be very careful of the new queries we add. We are also starting to hit connection limits because of simultaneous updates of job statuses.

Several design decisions are hampering new ideas and current activity. There are better ways to do some things, but they are almost impossible to integrate into the current framework. On top of this is the code cruft accumulated over 8 years of patches without much cleanup.

4. IceProd v2

The second version of IceProd is a complete rewrite of the codebase, focusing on current and future usage and eliminating old usage patterns. Instead of being add-ons, features like global DAG are now standard, with a more efficient implementation that was not possible before. There is also a strong focus on unit tests, code coverage and documentation, which were lacking in IceProd v1. Python 3 and other future-looking software is fully supported.

There are several major improvements, but one of the overarching ones is to be able to run on stock Python with no extra packages installed. While performance is degraded, it is sufficient for individual users or testing purposes. It also allows a single-site IceProd instance to be downloaded, set up, and started very quickly. This would allow, for instance, individual analyzers to set up their own grid infrastructure [4].

4.1 Database

The database is the single largest change to IceProd v2. It has moved from one central database to a local database at each site. This allows standalone single-site instances as well as continued running when the master/global queue is down. It also has a side effect of spreading out the load, since most requests from jobs or users only need to talk to the local database instead of all hitting the master.

The primary database type is SQLite, which comes integrated with Python. Support is also available for MySQL for higher performance; MongoDB support is planned to handle the large amount of statistics accumulated at the master node.

4.2 Scalability

All external communications with IceProd v2 flow through the web server, so it must be fast and scalable. Not only does it handle user-facing websites, but also job-to-site and site-to-site communications. It is built on non-blocking principles, offloading longer tasks to worker threads or other processes. It can easily handle 10 thousand connections per minute, even with over a thousand sustained connections.

4.3 Job optimization

One long-desired feature for IceProd has been a pilot infrastructure. Especially in the last few years, shorter jobs under an hour have become normal. Clusters and queueing systems generally don't like short jobs, so the obvious solution is to run multiple jobs one after the other inside the "single" job the queueing system is running. The pilot infrastructure takes this a step further by only asking for jobs that match the node's resources where the pilot is started. This allows jobs to run where they are best matched and reduces failures.

4.4 Software distribution

While not technically part of IceProd, software distribution is strongly related to it. Jobs need software to run, and for IceProd v1 this meant tarballs shipped with each job and extracted on the worker node. While this will still be supported for legacy operations in IceProd v2, the primary mode of software distribution is moving to CernVM-FS [9]. For sites that have CernVM-FS installed, it is used natively. Otherwise, an I/O interposition agent called Parrot is used to make the filesystem appear to exist, dynamically fetching necessary files from the network and presenting them to the application in the correct location [10].

4.5 JSON API

IceProd v2 will feature a full JSON API as the primary interface to controlling IceProd. This allows great flexibility because of JSON's ease of use and ubiquity. This gives a single, shared interface that both the website and command line clients can use as a backend. It also allows easy creation of new frontends in the future.

5. Conclusions

IceCube simulation production has heterogeneous, significantly opportunistic resources and diverse job requirements. IceProd is an easy to install user-level middleware designed to easily manage and monitor large sets of distributed jobs. With IceProd v1 straining under current usage, IceProd v2 is a complete rewrite designed to easily handle current production and more easily cope with future needs. It can scale from a single user at one site up to sites around the world and many millions of jobs. IceProd v2 is currently in beta testing for production deployment, which is scheduled for late 2015. Though it was originally developed for managing IceCube simulation production, IceProd is general enough for many types of grid applications and is generally available to the scientific community.

The following entities supported this work: U.S. National Science Foundation; Center for High Throughput Computing at the University of Wisconsin–Madison; Open Science Grid and the U.S. Department of Energy; National Energy Research Scientific Computing Center; WestGrid and Compute/Calcul Canada; Deutsches Elektronen-Synchrotron Grid Centre.

References

[1] M.G. Aartsen et al., J. Parallel Distrib. Comput. *The IceProd framework: Distributed data processing for the IceCube neutrino observatory* **75**, 198–211 (2015)
[2] R. Pordes et al., J. Phys. Conf. Ser. **78**, 012057 (2007)
[3] D. Chirkin, Nucl. Instrum. And Methods Phys. Res. Section A: Accelerators, Spectrometers, Detectors Associated Equipment *Photon tracking with GPUs in IceCube* **725**, 141–143 (2013)

[4] D. Schultz, J. Phys. Conf. Ser. **664**, 062056 (2015)
[5] I. Sfiligoi, J. Phys. Conf. Ser. **119**, 062044 (2008)
[6] T. DeYoung, Int. Conf. on Comp. in High-Energy Phys. And Nucl. Phys. (CHEP 2004) *IceTray: a software framework for IceCube* 463–466 (2005)
[7] J. C. Díaz-Vélez, J. Phys. Conf. Ser. **513**, 032026 (2014)
[8] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflows for e-Science *Workflow in Condor* (2007)
[9] J. Blomer, P. Buncic, T. Fuhrmann, Proc of the 1st int. workshop on Network-aware data management (NDM'11) 49–56 (2011)
[10] D. Thain, M. Livny, Scalable Computing: Practice Experience **6**, 9 (2005)