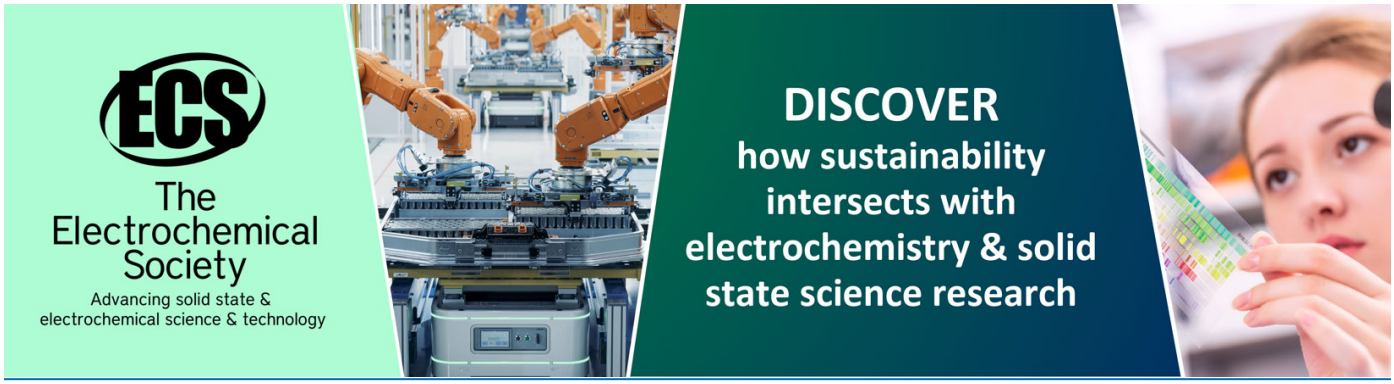
Pyglidein – A Simple HTCondor Glidein Service



IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 092018 doi:10.1088/1742-6596/898/9/092018

Pyglidein –A Simple HTCondor Glidein Service

D Schultz[1], B Riedel[2], G Merino[1]

[1] Wisconsin IceCube Particle Astrophysics Center, University of Wisconsin-Madison, 222 W Washington Ave, Suite 500, Madison, WI 53703, USA
[2] Computation Institute, Searle Chemistry Laboratory, The University of Chicago, 5735 South Ellis Avenue, Chicago, IL 60637

E-mail: david.schultz@icecube.wisc.edu

Abstract. A major challenge for data processing and analysis at the IceCube Neutrino Observatory presents itself in connecting a large set of individual clusters together to form a computing grid. Most of these clusters do not provide a "standard" grid interface. Using a local account on each submit machine, HTCondor glideins can be submitted to virtually any type of scheduler. The glideins then connect back to a main HTCondor pool, where jobs can run normally with no special syntax. To respond to dynamic load, a simple server advertises the number of idle jobs in the queue and the resources they request. The submit script can query this server to optimize glideins to what is needed, or not submit if there is no demand. Configuring HTCondor dynamic slots in the glideins allows us to efficiently handle varying memory requirements as well as whole-node jobs. One step of the IceCube simulation chain, photon propagation in the ice, heavily relies on GPUs for faster execution. Therefore, one important requirement for any workload management system in IceCube is to handle GPU resources properly. Within the pyglidein system, we have successfully configured HTCondor glideins to use any GPU allocated to it, with jobs using the standard HTCondor GPU syntax to request and use a GPU. This mechanism allows us to seamlessly integrate our local GPU cluster with remote non-Grid GPU clusters, including specially allocated resources at XSEDE supercomputers.

1. Introduction

The IceCube detector [1] is located at the geographic South Pole and was completed at the end of 2010. It consists of 5160 optical sensors buried between 1450 and 2450 meters below the surface of the South Pole ice sheet and is designed to detect interactions of neutrinos of astrophysical origin. The IceCube Collaboration's worldwide computing grid consists of many independent clusters that must be assembled into a useful whole. A global shared pool allows for a single submission point. Not only is this a great benefit to users, it also simplifies administration by making it easier to track jobs and monitor usage. IceCube simulations are heavily reliant on GPU computing for photon propagation in ice [2], so the shared pool helps maintain a balanced ratio of GPUs to CPUs. Clusters can be all CPU—so cannot run the full simulation locally—but still contribute to the shared pool.

1.1. Historical use and tests

The IceProd [3] software framework was developed by IceCube in 2006 to manage distributed simulation workloads. It has been used in production for 11 years to handle the data processing and simulation needs for the IceCube collaboration. IceProd instances were run directly on each cluster's submit machine, submitting to the local cluster. Only IceProd could see the shared pool, making it

impossible for non-production users to use the resources. It also required IceProd to know how to submit to all cluster types, resulting in extraneous code not related to the main dataset management task. Historically, IceProd has been difficult to install at new sites and was a large burden for collaborators. We started efforts to simplify remote cluster submission over the last few years, exploring several technologies and solutions.

GlideinWMS [4] is used by the Open Science Grid (OSG) [5] to connect university computing resources together into one large HTCondor [6] pool. The CMS experiment has used this for years to aggregate US resources, and has lately expanded this model to its global pool. Pilot jobs start the `startd` HTCondor daemon, remotely connecting to the pool central manager and acting like any other node in the pool. This remote starting of an HTCondor `startd` daemon is referred to as a glidein. We connected several of our larger sites together using GlideinWMS, with help from OSG. This is currently how IceCube sites running a grid submission compute element (CE) are connected to the global pool. Unfortunately, many IceCube sites do not run a CE and cannot be accessed so easily.

The ATLAS experiment has a similar pilot based workload management system, the AutoPyFactory (APF) [7]. With the addition of an HTCondor-G plugin [8], it can also start pilot glideins. It was tested with several European sites that use CREAM CEs [9]. For IceCube's use, it was not investigated further because of the limitation of only submitting to CEs, which we already use GlideinWMS for.

2. Pyglidein design

Two main design elements of pyglidein are the submission to any cluster independent of local batch system, and using a polling job submission model rather than the push model used by CEs. The submission independent of local batch system is achieved through a plugin model already tested in IceProd. To poll jobs from the central queue, a client script runs on the local submit machine, queries the central queue for jobs and their requirements, and submits matching glideins to the local queue. Polling helps to get around any incoming firewalls and works seamlessly with network address translation (NAT).

Another main goal of the pyglidein design is to keep it as simple as possible. Part of the reason not to modify GlideinWMS was its complexity. For this reason, most of pyglidein was written in Python, with only minimal shell scripts when needed. Additionally, some features that other glidein systems support will not appear here. There are no plans to support other experiments.

2.1. Server

The server is designed to be a lightweight and efficient way to communicate idle job requirements to the clients and gather status information on system performance. A Python script runs as a daemon on a machine with an HTCondor `schedd` daemon. The HTCondor queue status is queried in 5 minute intervals to reduce load on the scheduler. Idle jobs are recorded and grouped according to resource usage.

The server script uses JSON-RPC [10] through a HTTP server for communication with the outside world. Besides sending a list of idle jobs, the server also receives heart beat and status information from the clients. The HTTP server also has a status page, displaying the current idle jobs and the status of connected clients.

2.2. Client

The client script is where most of the logic in selecting and submitting glideins occurs. The first step is to contact the server for the idle job list, as mentioned in Section 2.1. This list is then filtered according to the site's available resources. For instance, a site may limit memory to under 3 GB. The client will consequently ignore any job that requests more than that. This also plays a large role in GPU matching, since most clusters do not have GPUs. Conversely, some clusters only have (or the allocation might only allow for) GPUs, so only GPU jobs are accepted. The job list is then sorted by resource preference. By default, GPU and high memory jobs are considered first. This is a configuration option though.

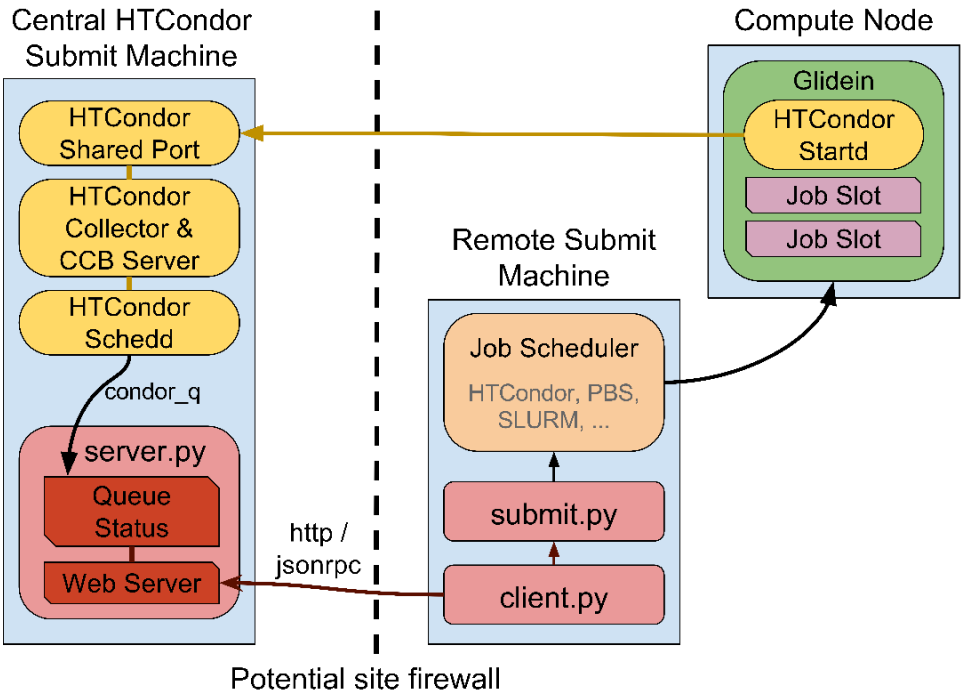


Figure 1 The architectural layout of the pyglidein system. Components of HTCondor relevant to glideins are included.

At this point, the client calls the submit method of the configured batch system class, detailed in Section 2.2.1. It is given the resource requirements, as well as the number of jobs to submit with those resources. After submission, monitoring information is sent back to the server to detail the state of the client and what it submitted. If configured, a cleanup command will be called in case a cluster does not remove temporary working directories after job completion.

2.2.1. Cluster submission. Batch system translation is built-in to pyglidein. The client configuration file specifies which batch system is in use. This maps to a class in a python file, which contains native commands for that batch system. Using inheritance avoids code duplication for common functions and similarities between batch systems. Currently supported are several variants of PBS, SLURM, Univa Grid Engine (UGE), LSF, and HTCondor. Each follows a similar pattern. First, it generates a submit file matching the resource requirements. Then it creates a shell script which, when run, will clear the environment (except a list of configured variables) and call the glidein start script. The environment is cleared to prevent interference between local environment variables and the glidein. This allows, for instance, our glidein to run inside other glideins from systems mentioned in Section 1.1. The client configuration also allows for custom directives to be added to the submit file. This helps with cluster-specific configuration, like specifying a user account or project.

2.3. Glidein

The glidein itself runs on the compute node of a cluster and starts an HTCondor `startd` daemon. It communicates with a centralized HTCondor collector via a shared port and CCB server [11], which allows it to run behind most firewalls. This strategy is built-in to HTCondor and only requires some configuration to set up.

Part of our customization of the glidein was to enable HTCondor partitionable slots [12]. Instead of the glidein only starting a single slot that matched the initial request when it was scheduled, it can dynamically size itself within those initial bounds. For example, an initial request for 2 CPUs and 10GB of memory can be split into two slots with 1 CPU each. This extends the usefulness of each glidein

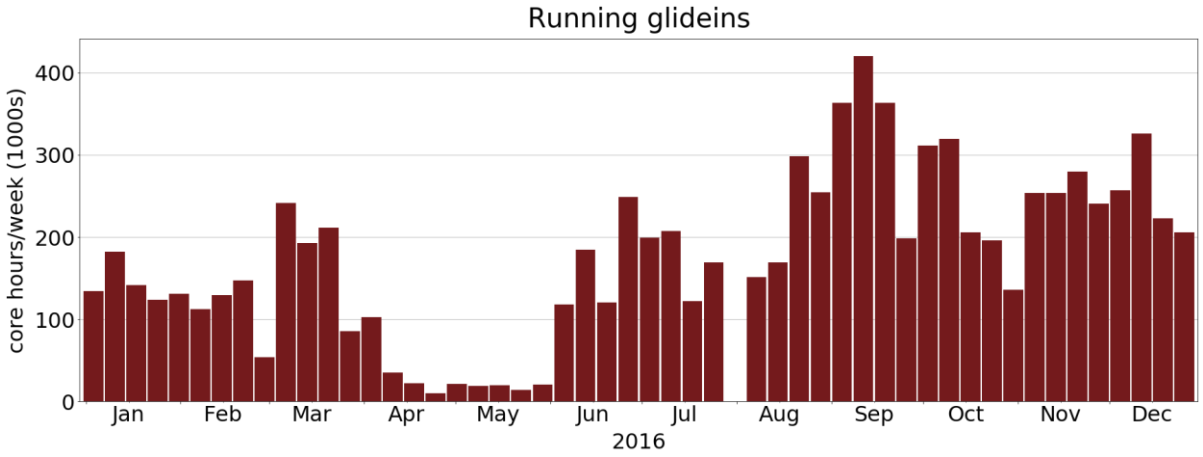


Figure 2 Accumulated time of running glideins during 2016, binned per week. The drop in April / May was a production issue, while the missing week at the end of July was a central storage failure.

beyond a single job. It also greatly eases the configuration of large or whole machine glideins, as they are configured the same as all other glideins.

The other customization is for GPUs. We must pass the correct GPU configuration from the cluster scheduler to HTCondor and to the executed job. Part of the submission environment shell script in Section 2.2.1 standardizes the incoming GPU information before passing it to the glidein start script. This script then configures HTCondor to use a custom GPU resource similar to how CPUs and memory are used [13]. It also allows for multiple GPUs to be configured in the same glidein. When a job slot is created with a GPU, the appropriate environment variables are automatically set before the actual job starts.

IceCube relies on CernVM-FS [14] for distributed access of software and small data files. For sites that have CernVM-FS installed, it is used natively. Otherwise, an I/O interposition agent called Parrot [15] 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. A wrapper script before each job starts up checks for the existence of CernVM-FS and, if necessary, starts Parrot.

3. Practical usage experiences

We have now run the pyglidein system in production for one year. Installing at each cluster is significantly easier than the IceProd system, such that it rapidly took the lead as the main simulation computing platform. Our global pool is now larger than ever before. It has also allowed us to easily share the distributed pool between official production and user analysis workloads. Users now have access to a larger pool with no change to their workflow compared to using OSG GlideinWMS.

As seen in Figure 2, we have now run several million hours of work on pyglidein without any serious issues. One of the most prominent issues has been with Parrot and OpenCL. OpenCL interacts at a fairly low level with the hardware, which has been troublesome with the way Parrot works. Through interaction with the Parrot team these situations have been resolved.

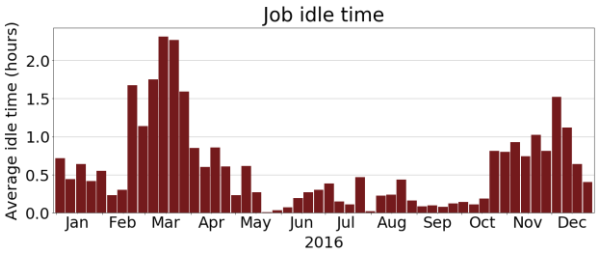


Figure 3 Average idle time before a job starts running, binned per week.

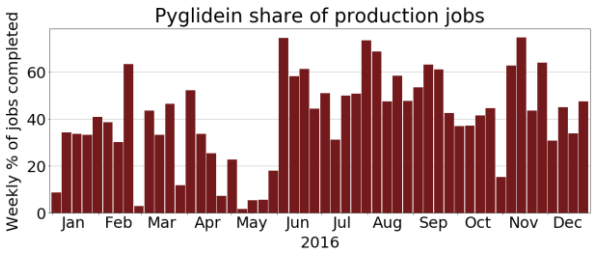


Figure 4 Percent of simulation production jobs run on pyglidein, binned per week.

Almost all activity has occurred on platforms running variants of RHEL 6, with a single glidein tarball and a homogenous pool. We now have access to two RHEL 7 clusters and are actively working on integrating them into the pool. Besides a new glidein tarball, job matching must also be considered. Several future clusters will also have Ubuntu-based machines, so our approach should be flexible regarding platforms.

4. Conclusions

Pyglidein is considered a success for IceCube, allowing easier and greater access to our global computing resources. Using HTCondor flocking, we have connected this to our existing GlideinWMS pool, presenting a single global pool to users. Pyglidein is running well at the current level, but we'd like to scale up an order of magnitude in the next few years, while keeping the effort required to operate it under control. Future work will also focus on multi-platform support, Linux containers, and limiting sites to different groups of users.

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, the Center for High Throughput Computing (CHTC) at the University of Wisconsin–Madison, the Open Science Grid (OSG) grid infrastructure; U.S. Department of Energy, and National Energy Research Scientific Computing Center; Natural Sciences and Engineering Research Council of Canada, WestGrid and Compute/Calcul Canada; Swedish Research Council, Swedish Polar Research Secretariat, Swedish National Infrastructure for Computing(SNIC), and Knut and Alice Wallenberg Foundation, Sweden; German Ministry for Education and Research (BMBF), Deutsche Forschungsgemeinschaft (DFG), Helmholtz Alliance for Astroparticle Physics (HAP), Research Department of Plasmas with Complex Interactions (Bochum), Germany; Fund for Scientific Research (FNRS-FWO), FWO Odysseus programme, Flanders Institute to encourage scientific and technological research in industry (IWT), Belgian Federal Science Policy Office (Belspo); University of Oxford, United Kingdom; Marsden Fund, New Zealand; Australian Research Council; Japan Society for Promotion of Science (JSPS); the Swiss National Science Foundation (SNSF), Switzerland; National Research Foundation of Korea (NRF); Danish National Research Foundation, Denmark (DNRF).

References

[1] Halzen F 2003 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** volume 13, pages 13-16, July 2003

[2] Chirkin D 2013 Photon tracking with GPUs in IceCube *Nucl. Instrum. and Methods Phys. Res. Section A: Accelerators, Spectrometers, Detectors Associated Equipment* **725** (0)141–143 VLVvT 11, Erlangen, Germany, 12 – 14 October, 2011. 5th International Workshop on Very Large Volume Neutrino Telescopes, The future of high-energy neutrino astronomy

[3] Aartsen M G *et al* 2015 The IceProd framework: Distributed data processing for the IceCube neutrino observatory *J. Parallel Distrib. Comput.* **75** 198–211

[4] Sfiligoi I 2008 *J. Phys.: Conf. Ser.* **119** 062044

[5] Pordes R *et al* 2007 *J. Phys.: Conf. Ser.* **78** 012057

[6] Thain D, Tannenbaum T and Livny M 2005 Distributed computing in practice: the Condor experience *Concurrency and Computation: Practice and Experience* **17** 323–356

[7] Caballero J on behalf of the ATLAS Collaboration 2012 AutoPyFactory: a scalable flexible pilot factory implementation *J. Phys.: Conf. Ser* **396** 032016

[8] Taylor R P *et al* 2015 The evolution of cloud computing in ATLAS *J. Phys.: Conf. Ser* **664** 022038

[9] Aiftimiei C *et al* 2008 Job submission and management through web services: the experience with the CREAM service *J. Phys.: Conf. Ser* **119** 062004

[10] JSON-RPC 2.0 specification, http://www.jsonrpc.org/specification
[11] Tannenbaum T 2010 What's new in condor? what's coming up? *Condor Week 2010* (Madison, WI: University of Wisconsin) http://research.cs.wisc.edu/htcondor/CondorWeek2010/condor-presentations/tannenba_roadmap_2010.pdf
[12] Thain G 2012 Dynamic Slot Tutorial *Condor Week 2012* (Madison, WI: University of Wisconsin) http://research.cs.wisc.edu/htcondor/CondorWeek2012/presentations/thain-dynamic-slots.pdf
[13] Knoeller J 2014 Managing GPUs in HTCondor 8.1/8.2 *HTCondor Week 2014* (Madison, WI: University of Wisconsin) http://research.cs.wisc.edu/htcondor/HTCondorWeek2014/presentations/KnoellerJ-GPU.pdf
[14] Blomer J, Buncic P and Fuhrmann T 2011 *Proc. of the 1st int. workshop on Network-aware data management (NDM'11)* 49–56
[15] Thain D and Livny M 2005 *Scalable Computing: Practice Experience* **6** 9