Mixing HTC and HPC Workloads with HTCondor and Slurm

Mixing HTC and HPC Workloads with HTCondor and Slurm

C Hollowell, J Barnett, C Caramarcu, W Strecker-Kellogg, A Wong and A Zaytsev

Brookhaven National Laboratory, Upton, NY 11973, USA

E-mail: hollowec@bnl.gov, jbarnett@rcf.rhic.bnl.gov, caramarc@bnl.gov, willsk@bnl.gov, tony@bnl.gov, alezayt@bnl.gov

Abstract. Traditionally, the RHIC/ATLAS Computing Facility (RACF) at Brookhaven National Laboratory (BNL) has only maintained High Throughput Computing (HTC) resources for our HEP/NP user community. We've been using HTCondor as our batch system for many years, as this software is particularly well suited for managing HTC processor farm resources. Recently, the RACF has also begun to design/administrate some High Performance Computing (HPC) systems for a multidisciplinary user community at BNL. In this paper, we'll discuss our experiences using HTCondor and Slurm in an HPC context, and our facility's attempts to allow our HTC and HPC processing farms/clusters to make opportunistic use of each other's computing resources.

1. Introduction

The RHIC/ATLAS Computing Facility (RACF) at Brookhaven National Laboratory (BNL) is the primary computing center for the experiments at RHIC, and the US Tier1 computing facility for the ATLAS experiment at LHC. As of Fall 2016, the RACF has served the High Energy and Nuclear Physics (HEP/NP) community for over 20 years. We currently provide our users with a processor farm consisting of over 50,000 logical CPU cores, and host more than 70 PB of data on tape (HPSS), and 45 PB on disk (GPFS, NFS, dCache and XROOTD).

Experimental HEP/NP processing workloads have traditionally fit a High Throughput Computing (HTC) model: they are generally embarrassingly parallel, with individual jobs fitting on a single batch processing host, and in a number of cases using a single CPU core [1], [2]. As such, experimental HEP/NP jobs usually do not make use of multi-node parallelization libraries such as MPI or PVM. Therefore, they typically do not require specialized low-latency node communication interconnects such as Infiniband, or batch systems supporting complex multi-node job scheduling policies.

HEP/NP accelerator offline data processing runs efficiently and economically on commodity hardware. As a result, for many years the majority of the processing resources for the RHIC and LHC experiments have consisted of commodity x86/x86_64 servers, with only limited use of specialized processing hardware such as GPUs and MIC processors [3]. Besides simulation, HEP/NP processing workloads tend to be data intensive, however. This makes offloading them (other than simulation) to commercial cloud providers expensive, due to imposed data storage and transfer charges [4]. Such migration is even more challenging at sites with limited wide area network connectivity to cloud providers. Therefore, dedicated/local datacenters are still necessary, and are currently used to support the majority of data processing for RHIC and LHC [5].

Recently, scientific computing at BNL has undergone some consolidation as a result of the laboratory's Computational Science Initiative (CSI). As a result, RACF is now a part of BNL's Scientific Data & Computing Center (SDCC), and our administrative purview has been extended to maintaining BNL's High Performance Computing (HPC) systems, as well as our traditional HTC processing installations. As part of the consolidation process, disparate older HPC clusters maintained by different organizations (Biology, Center for Functional Nanomaterials, etc.) have been retired, and replaced with two new centralized systems: the Institutional Cluster (IC), and the Knights Landing (KNL) Cluster. Specific details on the hardware for these systems is available in Table 1.

Table 1. SDCC HPC cluster hardware.

System	**Specifications**
Institutional Cluster (IC)	108 HP Proliant XL190r nodes (adding another 92 soon)
	54 Apollo r2600 chassis
	Per node configuration:
	2 Intel Xeon E5-2695v4 CPUs @ 2.1 GHz
	256 GB DDR4 2400 MHz RAM
	2 Nvidia K80 GPUs
	EDR Infiniband interconnect
KNL Cluster	144 Intel S7200AP-based nodes
	36 Intel H2312XXL2 chassis
	Per node configuration:
	1 Knights Landing Xeon Phi 7230 CPU @ 1.3 GHz
	192 GB DDR4 1200 MHz RAM
	Dual-rail Omni-Path interconnect

2. Adaptation to Support HPC Infrastructure

While there is a significantly overlapping skill set required for the administration and design of HTC and HPC farm/cluster systems, there were a number of technologies our staff had to become familiar with to support our new HPC infrastructure. This included hardware such as GPUs, MIC (Xeon Phi) processors, and low-latency interconnects, including Infiniband and Omni-Path. We also had to adapt to the installation and support of the multiple MPI implementations required by our users, including OpenMPI, MPICH, and MVAPICH.

The batch system that has been in use at our facility for a number of years, HTCondor [6], is primarily designed for HTC use. While it does support multi-node job scheduling, this functionality is somewhat limited. As a result, we adopted a different batch system to manage our HPC processing resources: Slurm [7].

3. Batch Systems

3.1. HTCondor

HTCondor (originally known simply as "Condor") is a batch system which has been developed and maintained by the University of Wisconsin at Madison since the early 1980s [8]. For quite some time, the focus of HTCondor has been on supporting HTC [9]. This became even more evident when the project incorporated HT, for "High Throughput", into its name in 2012 [10]. The primary goal of HTCondor is to fill all available batch slots in a pool as quickly as possible. Fairness of resource assignment (i.e. user priorities) is a secondary consideration, which is balanced over a large integral of time.

We initially introduced HTCondor into our facility as an LSF replacement over 10 years ago (2004). At the time, RACF was a fairly early adopter of HTCondor in the experimental HEP/NP computing community. We quickly found that the system was highly scalable in handling large numbers of disparate jobs. Currently, a single HTCondor *condor_collector/condor_negotiator* pool in our environment easily manages more than 25,000 batch slots and running jobs, with more than 200,000 idle jobs in the queue. This is a critical feature of an HTC job scheduler, as these systems are designed to run large numbers of simultaneous, loosely coupled jobs. As HTCondor is freely available open source software, its use has also saved us considerable LSF licensing costs.

While HTCondor is an excellent batch system for HTC use, its HPC support is fairly limited. HTCondor jobs specify an execution universe in their job description files (JDFs), which define the environment required for execution. For HTC jobs, this is typically the "vanilla universe", or the "standard universe". HTCondor supports the execution of multi-node MPI jobs via its "parallel universe". However, when one uses this universe, all jobs must be submitted from a single submit host running *condor_schedd*, referred to as the "dedicated scheduler" [11]. Our HTC systems, on the other hand, utilize many HTCondor submit hosts to improve scaling, and service reliability.

When using a dedicated scheduler for the "parallel universe", only best fit or FIFO scheduling is supported. As such, the "parallel universe" can't easily be used in an environment where different users/groups pay for shares of a cluster, and opportunistic usage is permitted. Finally, using MPI with the "parallel universe" requires users to utilize custom "mpiscripts" that call *condor_ssh* to start-up processes on different nodes. This adds an extra step for users, which can be somewhat confusing, and isn't required with more HPC-centric batch systems. Using SSH for job spawning also increases job start-up times, particularly for large parallel jobs. For these reasons, HTCondor's "parallel universe" is effectively restricted to use on small dedicated HPC clusters.

3.2. Slurm

Slurm is currently developed and maintained by SchedMD, with a number of outside contributors. It's a popular batch system for HPC use, and is presently utilized on approximately 60% of the supercomputers in the TOP500 list [12]. Like HTCondor, Slurm is also freely available open source software. The majority of the HPC clusters at BNL were running Moab/TORQUE. In order to eliminate the associated licensing costs with this software, we moved to the use of Slurm on our IC and KNL clusters.

Slurm supports a number of useful and required features for our HPC systems. Most importantly, it provides highly available advanced fairshare scheduling for parallel multi-node jobs. This includes support for the scheduling of GPU resources, resource partitioning, whole node scheduling, and advanced reservation. Slurm is also network-topology aware, and has the ability to make scheduling decisions based on topology.

The Knights Landing Xeon Phi processors in our KNL cluster support multiple NUMA configurations for their on-chip MCDRAM memory. Unfortunately, reboots are necessary to switch the CPUs between the various configurations (independent, cached, and hybrid). Slurm allows users to request a particular configuration for their jobs on these systems, and will schedule reboots accordingly [13].

Slurm also supports PMI2/x process spawning. Through PMI2/x, MPI implementations can interact directly with the batch system when starting processes on multiple nodes, rather than having to use an external mechanism such as SSH. This simplifies the execution of MPI jobs for users, and decreases multi-node job start-up time.

However, there are some impediments to using Slurm in an HTC environment. We'd encountered some anecdotal reports from other HEP/NP sites that Slurm is not capable of handling the large numbers (many thousands) of running/queued jobs required for HTC use. However, Slurm recently published an HTC optimization document, so this may no longer be the case [14]. On the other hand, Slurm includes a number of features not required for HTC use, and the existence of these features can somewhat complicate administration. It also requires that users be added to Slurm's accounting DB before using the batch system (when accounting is enabled): this is an extra administrative step not necessary with HTCondor. Finally, as HTCondor is reliably/efficiently scheduling resources on our HTC systems, and because our users have extensive experience with this batch system, it isn't desirable to change.

4. Mixing HTC and HPC Batch Workloads

As discussed, due to disparate requirements and capabilities, it was necessary to use different batch systems for our HTC and HPC installations (HTCondor, and Slurm, respectively). While no official resource sharing relationship has been defined between our HTC and HPC systems, from a technical level, we were interested in determining if it would be possible to enable opportunistic usage between them.

4.1. Using HTCondor's Job Router with Slurm

Due to the inherent packing inefficiencies of multi-node jobs, there are often unused processing resources available in HPC clusters. Figure 1 provides an example of this situation, which arises due to the fact that multi-node HPC jobs usually require a specific number of nodes/CPUs to be allocated to them. The unused nodes/CPUs in an HPC cluster can be filled with single-node or single-core HTC jobs, making opportunistic usage by HTC users a desirable potential option.

Since version 7.1.0, HTCondor has included the *condor_job_router* daemon. This daemon provides the ability to automatically transform idle "vanilla universe" jobs to the "grid universe". In the March 2016 8.5.3 development release, HTCondor added support for a new "slurm" batch type to the "grid universe". This support allows HTCondor to interface with Slurm, and we investigated this functionality for HTC $\rightarrow$ HPC opportunistic use.

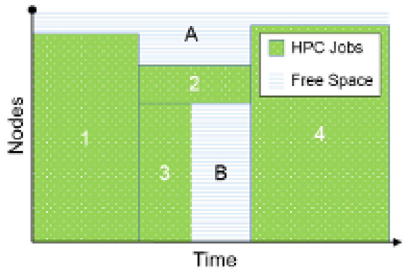

Figure 1. Packing inefficiencies in HPC clusters.

We setup HTCondor and Slurm batch pools in a test environment, and enabled the *condor_job_router* on the HTCondor submit host. A separate HTCondor pool, consisting of the standard HTCondor central manager daemons (*condor_negotiator*, and *condor_collector*), as well as *condor_schedd*, was started on the Slurm submit system, and HTCondor-C (standard HTCondor network protocol) communication was enabled between the pools. A low priority, preemptible, "condor" partition was added to the Slurm configuration on the test HPC cluster. The relevant portions of the configuration for the batch systems are presented in detail in Figure 2, and an overall view of the architecture is shown in Figure 3.

An alternative to setting up a separate HTCondor pool on the Slurm submission system would have been to use BoSCO (BLAHP over SSH Condor Overlay) [15] for the routed "grid universe" jobs. However, this would have required enabling user SSH access between the HTCondor and Slurm submit hosts. While we were interested in exploring opportunistic batch use between our

HPC and HTC resources, we wanted to avoid giving the users of these systems interactive/SSH access to each other's hosts.

HTCondor job router daemon configuration (/etc/condor/config.d/20_router):

```
# These settings become the default settings for all routes
JOB_ROUTER_DEFAULTS = \
[ \
    requirements=target.WantJobRouter is True; \
    MaxIdleJobs = 20; \
    MaxJobs = 100; \
    /* now modify routed job attributes */ \
    /* remove routed job if it goes on hold or stays idle for over 6 hours */ \
    set_PeriodicRemove = JobStatus == 5 || \
        (JobStatus == 1 && (CurrentTime  QDate) > 3600*6); \
    set_requirements = true; \
]
JOB_ROUTER_ENTRIES = \
[ GridResource = "condor slurmsub01.sdcc.bnl.gov slurmsub01.sdcc.bnl.gov"; \
    name = "Slurm_Pool"; \
    set_remote_JobUniverse = 9; \
    set_remote_GridResource = "batch slurm"; \
    set_remote_Requirements = False; \
]
JOB_ROUTER_SCHEDD2_NAME = condsub01.rcf.bnl.gov
JOB_ROUTER_SCHEDD2_POOL = condmaster01.rcf.bnl.gov
```

Slurm "condor" opportunistic use partition configuration:

```
PartitionName=condor Shared=FORCE OverSubscribe=YES State=UP Nodes=hpcnode00[1-6]
MaxTime=100:00:00 DefaultTime=05:00 qos=condor Priority=1 PreemptMode=cancel
Default=no
```

Figure 2. HTCondor job router and Slurm configurations.

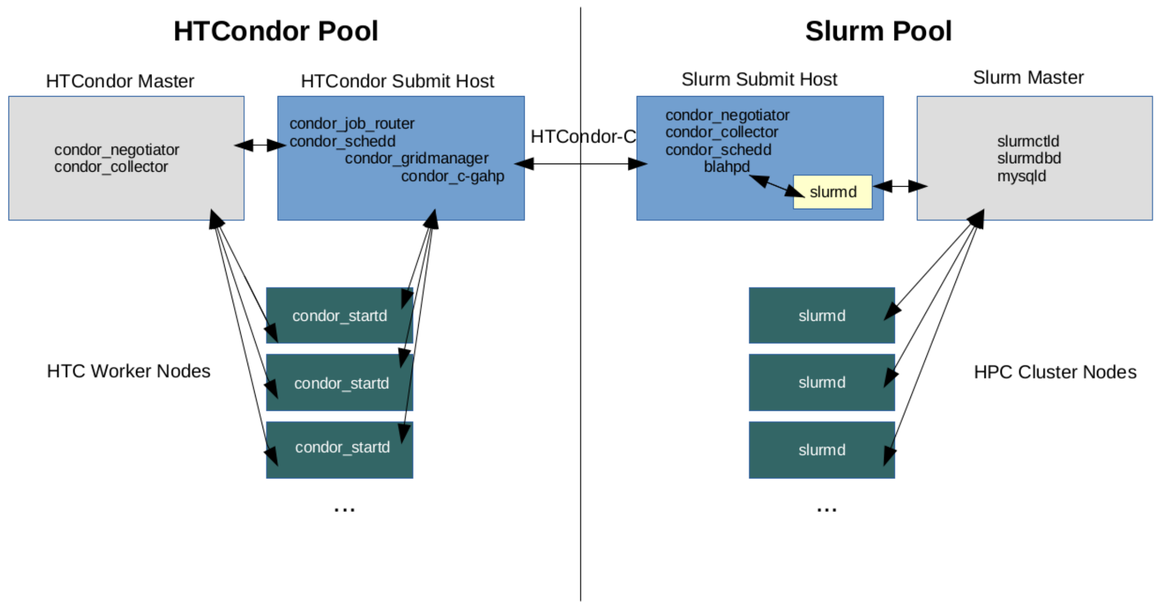

Figure 3. Architecture: HTCondor job router with Slurm.

In order to opt-in to use of the Slurm-managed resources, test users added a *WantJobRouter* jobad line to their JDFs, advertising their request to be considered for routing, and submitted their "vanilla universe" jobs to the local *condor_schedd* as usual. If no job slots were available on the local HTCondor-managed resources, the *condor_job_router* daemon would submit a new copy of the job to the *condor_schedd* with the universe transformed to "grid". The *condor_schedd* then spawned a *condor-c_gahp* server to submit the job to the remote HTCondor pool running on the Slurm submit host. Finally, the *condor_schedd* running on that system would start a *blahpd* (Batch Local ASCII Helper Protocol Daemon) process to submit the job to Slurm, via *slurm_submit.sh* (which calls *sbatch*), and manage its life cycle. In our configuration, jobs remaining idle in the Slurm queue for over 6 hours were removed, and reconsidered for local execution. An example of the job routing process in action is presented in detail in Figure 4.

Using our test configuration, we were able to successfully demonstrate opportunistic usage of Slurm-managed HPC resources by HTCondor. However, we did run into a few issues. First, it would take up to five minutes for the local copy of the job in the HTCondor queue to reflect the remote/routed job state. Fortunately, the local accounting statistics were updated to accurate final values upon job completion. Also, the HPC HTCondor pool spool directory had to be located in a shared filesystem. This was because HTCondor copied the users' executables to this directory, and it needed to be available on the Slurm compute hosts for execution. Finally, in order to allow our HTC users to run in Slurm, we had to setup a cron script to periodically add users (via *sacctmgr*) with opportunistic jobs in the HTCondor queue to Slurm's accounting DB, if they weren't already present. The implementation of this script made use of a cached list of previously seen users to avoid unnecessary calls to *sacctmgr*.

4.2. HPC Opportunistic Usage of HTC Resources

Unfortunately, for a number of reasons, it isn't currently possible to allow our HPC users to opportunistically run HPC jobs on our HTC systems. This is primarily due the fact that we don't want to run a single HTCondor "dedicated scheduler" to support the "parallel universe" on these systems. The lack of GPUs, Xeon Phi processors, and low latency interconnects on our HTC hosts also limits the ability to run HPC workloads on our HTC farms, as most HPC applications executed here require one or more of these technologies to run efficiently.

However, some of our HPC users have been able to reduce their jobs to execution on a single CPU core or individual host, effectively making them HTC jobs. For example, this is the case with some of the VASP [16] based materials modeling jobs run on our systems. For these jobs, there is a possibility of using HTCondor flocking to enable opportunistic usage of HTC resources. Unfortunately, there is no way to transparently route jobs in the Slurm queue to HTCondor; users would need to submit such jobs directly to the HTCondor pool on the Slurm submit system.

5. Conclusions

As a result of BNL's CSI consolidation, after many years of maintaining HTC processing installations for HEP/NP, the RACF has now also been tasked with administrating centralized HPC systems for a multidisciplinary user community at the laboratory. Because of the differing requirements from our HTC and HPC users, we're administrating separate batch systems, HTCondor and Slurm, respectively, to manage their resources.

Using a test configuration we've developed which utilizes the *condor_job_router* daemon, it's technically possible to allow our HTC/HTCondor users to opportunistically utilize our HPC/Slurm resources, with minimal changes to their job submission procedures. Due to intrinsic HPC cluster packing inefficiencies, and the "commodity-centric" nature of HTC jobs, opportunistic usage of unused HPC cluster resources is an appealing capability.

If HPC users are able to reduce their jobs to more HTC-like workloads, it's possible to enable opportunistic usage of our HTC installations via HTCondor flocking. However, this requires HPC users to eliminate their requirements for specialized processing hardware/interconnects, and fit their jobs on individual batch hosts. As there are no built-in cross-batch system job routing capabilities in Slurm, users would have to manually submit such jobs to HTCondor.

One-line HTCondor JDF addition to opt-in:

```
+WantJobRouter = LastRejMatchTime =!= UNDEFINED
```

This ensures jobs are only considered for routing to Slurm if there are no local HTCondor slots available.

Submit the job to HTCondor:

```
condsub01% condor_submit myjob.jdf
```

Query the local HTCondor queue; two copies of the job are present, indicating there were no local HTCondor resources available to run the job, and it has been routed to Slurm:

```
condsub01% condor_q
 Schedd: condsub01.rcf.bnl.gov :
ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
5216.0   testuser        9/21 16:07   0+00:00:00 I  0   122. testloop.sh
5217.0   testuser        9/21 16:07   0+00:00:00 I  0   122. testloop.sh
```

Query the HTCondor universe of the jobs in the queue. The submitted job is a "vanilla universe" job (Universe = 5), and the routed job (5217) has had its universe changed to "grid" (Universe = 9):

```
condsub01% condor_q -format '%s:' ClusterID -format 'Universe = %s\n' JobUniverse
5216:Universe = 5
5217:Universe = 9
```

Obtain some additional information about the routed job, including the ID of the job it was routed from, and the grid interfaces used to remotely submit the job to Slurm:

```
condsub01% condor_q 5217 -autoformat:n RoutedFromJobID GridResource remote_GridResource
5216.0
condor slurmsub01.sdcc.bnl.gov slurmsub01.sdcc.bnl.gov
batch slurm
```

On the Slurm submit host, query the local HPC HTCondor scheduler, and examine some additional information about the job routed there:

```
slurmsub01% condor_q -nobatch
 Schedd: slurmsub01.sdcc.bnl.gov :
ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
3793.0  testuser         9/21 16:09   0+00:00:00 I  0   122. testloop.sh
slurmsub01% condor_q 3793 autoformat:n JobUniverse GridResource remote_GridResource
9
batch slurm
undefined
```

Examine the Slurm job queue, and note that the routed job has started to run:

```
slurmsub01% squeue
    JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    5483     condor bl_76b98 testuser R       00:30      1 hpcnode005
```

Figure 4. Job routing example.

References

[1] High throughput computing: https://en.wikipedia.org/wiki/High-throughput_computing
[2] Blomer J, et al. 2011 Decentralized Data Storage and Processing in the Context of the LHC Experiments at CERN *Munich, Tech. U. CERN-THESIS-2011-251*
[3] WLCG and its Technical Architecture: https://www.gc3.uzh.ch/edu/lsci2012/lecture11.pdf
[4] Bloom K and Gerber R 2013 Computing Frontier: Distributed Computing and Facility Infrastructures *Proceedings, 2013 Community Summer Study on the Future of U.S. Particle Physics: Snowmass on the Mississippi (CCS2013)*
[5] Wong A, et al. 2017 The role of dedicated computing centers in the age of cloud computing *To be published in the CHEP 2016 proceedings, J. Phys.: Conf. Ser.*
[6] HTCondor: https://research.cs.wisc.edu/htcondor
[7] Slurm Workload Manager: https://slurm.schedmd.com/
[8] Thain D, et al. 2005 Distributed computing in practice: The Condor experience *Concurrency and Computation: Practice & Experience - Grid Performance Volume 17 Issue 2-4*
[9] HTCondor - High Throughput Computing: https://research.cs.wisc.edu/htcondor/htc.html
[10] "Condor" name changing to "HTCondor": https://www-auth.cs.wisc.edu/lists/htcondor-world/2012/msg00008.shtml
[11] HTCondor 8.4 Manual - Parallel Applications (Including MPI Applications): http://research.cs.wisc.edu/htcondor/manual/v8.4/2_9Parallel_Applications.html
[12] Slurm Workload Manager: https://en.wikipedia.org/wiki/Slurm_Workload_Manager
[13] Intel Knights Landing (KNL) User and Administrator Guide: https://slurm.schedmd.com/intel_knl.html
[14] High Throughput Computing Administration Guide: https://slurm.schedmd.com/high_throughput.html
[15] Bosco - https://bosco.opensciencegrid.org/about/
[16] About VASP - https://www.vasp.at/index.php/about-vasp/59-about-vasp