A Feasibility Study on workload integration between HTCondor and Slurm Clusters

R. Du[1,*], *J.* Shi[1,**], *J.* Zou[1,***], *X.* Jiang[1,****], *Z.* Sun[1,†], and *G.*Chen[1,‡]

[1]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, China, 100049

Abstract. There are two production clusters co-existed in the Institute of High Energy Physics (IHEP). One is a High Throughput Computing (HTC) cluster with HTCondor as the workload manager, the other is a High Performance Computing (HPC) cluster with Slurm as the workload manager. The resources of the HTCondor cluster are funded by multiple experiments, and the resource utilization reached more than 90% by adopting a dynamic resource share mechanism. Nevertheless, there is a bottleneck if more resources are requested by multiple experiments at the same moment. On the other hand, parallel jobs running on the Slurm cluster reflect some specific attributes, such as high degree of parallelism, low quantity and long wall time. Such attributes make it easy to generate free resource slots which are suitable for jobs from the HTCondor cluster. As a result, if there is a mechanism to schedule jobs from the HTCondor cluster to the Slurm cluster transparently, it would improve the resource utilization of the Slurm cluster, and reduce job queue time for the HTCondor cluster. In this proceeding, we present three methods to migrate HTCondor jobs to the Slurm cluster, and concluded that HTCondor-C is more preferred. Furthermore, because design philosophy and application scenes are different between HTCondor and Slurm, some issues and possible solutions related with job scheduling are presented.

1 Introduction

There are two local computing clusters in the Institute of High Energy Physics(IHEP), one is a HTCondor cluster, the other is a Slurm cluster. Most jobs running on the HTCondor cluster are single-core jobs, while parallel and multi-core jobs are running on the Slurm cluster.

The resource utilization ratio of the HTCondor cluster has reached more than 90% , which means it has reached the bottleneck of resource provision for now. However, the workload of the Slurm cluster is relatively not heavy, and the resource utilization ration is 50% on average. If HTCondor jobs could be migrated and run on Slurm cluster, users of the HTCondor cluster could have more resources to run their jobs, and resource utilization ratio of the Slurm cluster would be increased at the same time.

*e-mail: duran@ihep.ac.cn
**e-mail: shijy@ihep.ac.cn
***e-mail: zoujh@ihep.ac.cn
****e-mail: jiangxw@ihep.ac.cn
†e-mail: sunzy@ihep.ac.cn
‡e-mail: gang.chen@ihep.ac.cn

To testify that workload integration between HTCondor and Slurm clusters is feasible, section 2 lists and compares related and similar works. Later, three ways to inter-connect the HTCondor cluster and the Slurm cluster are investigated in section 3, while issues and possible solutions related with job scheduling and resource allocation are described in section 4. And section 5, the last part of this proceeding, gives the conclusion of our work.

2 Related works

Similar research about workload migration was done on how to take advantage HPC resources for HEP computing applications. B. O'Brian et al. [1] proposed to run HTC jobs on HPC sites, and listed possible obstacles including job submission, data management and network connections etc. In the following years, many HEP experiments including ATLAS, CMS and ALICE, proposed individual solutions to address these obstacles and to make better use of HPC resources. For example, ATLAS is using ARC CE to bridge grid and HPC cluster, and Event Service is provided to address the issue that jobs may get preempted without any warnings[2–4]. CMS adopts HTCondor and ROCED to run jobs inside of Virtual Machines on HPC clusters[5, 6]. ALICE makes use of ANALISA to simplify access to HPC systems by handling job submission, software management, and local data management[7]. Conventionally, good use cases for HPC cluster are computation heavy applications, that is the reason why most of workload from HEP computing to HPC clusters are simulation jobs. To make I/O intensive jobs from HEP experiments to be run on HPC smoothly, NERSC Cori provides a storage cache layer named Burst Buffer to improve I/O performance[8]. Besides, to make installation of HEP software stack more convenient, a container technology named shifter is developed to provide the same system environment as grid on NERSC Cori[9].

All the mentioned researches aim to solve one or more issues about HTC workload migration to HPC sites. When we try to migrate HTC jobs from the HTCondor cluster to the Slurm cluster at IHEP, similar obstacles such as job submission, system environment are confronted as well. However, because HTCondor and Slurm clusters are located at the same LAN and mount the same shared file systems, network and data are not issues anymore. As a result, we focus on more specific issues including better ways to inter-connect HTCondor and Slurm clusters and other issues related with job scheduling and resource allocation. C. Hollowell et al.[10] tested HTCondor-C to mix HTC and HPC workloads. Comparing with the research C Hollowell et al. did, we tested two more ways, overlap and flocking, to inter-connect HTCondor cluster and Slurm cluster. And HTCondor-C is more preferable for our future work, more details are presented in section 3. Additionally, we also proposed three issues and possible solutions in section 4.

3 Job migration

The first question about workload integration is how to migrate jobs from HTCondor to Slurm. The most instinct way to run HTCondor jobs on the Slurm cluster is by overlapping part of work nodes from the Slurm cluster with the HTCondor cluster. Because there is no official name of this interconnection way, we would call it 'overlap' in this paper. Apart from overlap, HTCondor provides two native methods: flocking and HTCondor-C[11]. In simple words, to reach the aim of job migration from HTCondor and Slurm, flocking treated the HTCondor cluster and the Slurm cluster as two separated pools. And jobs submitted to the HTCondor cluster will be flocked to the Slurm cluster if the jobs are not allowed to run immediately. Comparing with flocking, HTCondor-C is easier to understand. It is designed to run HTCondor jobs on other heterogeneous clusters, for example, a Slurm cluster in our

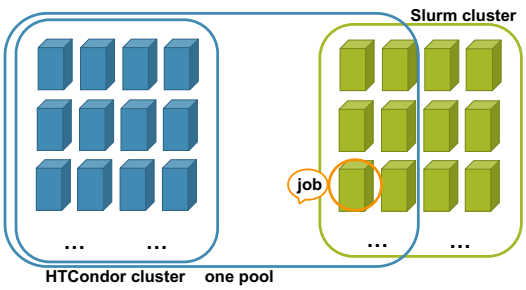

Figure 1: Architecture of overlap.

```
HTCondor cluster : slurm04.ihep.ac.cn
Slurm cluster : slurm03.ihep.ac.cn

# submit new jobs
> condor_submit submit_basic_queue_multi.jdf
Submitting job(s)...
3 job(s) submitted to cluster 46

# check job status
> condor_q 46 -af remotehost
slot1@slurm03.ihep.ac.cn
slot2@slurm03.ihep.ac.cn
slot3@slurm03.ihep.ac.cn
```

Figure 2: Test results of overlap.

Table 1: Key configurations of overlap.

HTCondor cluster, slurm04.ihep.ac.cn : /etc/condor/config.d/conf
DAEMON_LIST = MASTER, SCHEDD, COLLECTOR, NEGOTIATOR, STARTD
CONDOR_HOST = slurm04.ihep.ac.cn
Slurm cluster, slurm03.ihep.ac.cn : /etc/condor/config.d/worker.conf
DAEMON_LIST = MASTER, STARTD
CONDOR_HOST = slurm04.ihep.ac.cn
Slurm cluster, slurm03.ihep.ac.cn : /etc/slurm/slurm.conf
# stop condor_startd when slurm jobs coming
Prolog=/usr/local/slurm/condor_off.prolog
# start condor_startd when slurm jobs finished
Epilog=/usr/local/slurm/condor_on.epilog

circumstances. With HTCondor-C, jobs from HTCondor will be transformed into Slurm jobs and migrated to the job queue of the Slurm cluster. Later, the migrated jobs will get opportunities to run if there are available resources. To testify if these three methods are working, a testbed is set up: a host slurm04 is configured as a HTCondor cluster, and a host slurm03 is configured as a Slurm cluster.

3.1 Overlap

As it is earlier mentioned, HTCondor jobs are run on the Slurm cluster by overlapping part of worker nodes from the Slurm cluster. When it comes to installation and configuration, nodes overlapping is accomplished by running HTCondor daemons on Slurm worker nodes, which means that both Slurm and HTCondor daemons are running on some work nodes, and that is the exact reason why this method is named ‘overlap’.

From the view of HTCondor, those overlapped worker nodes are the same as the original worker nodes of the HTCondor cluster, meanwhile, Slurm does not know that some of its work nodes are added to the HTCondor cluster. Then here comes the question that, what’s going on to happen if both Slurm and HTCondor try to allocate the same resources to jobs? To avoid resource allocation confliction, some codes required to be added to the prolog and epilog scripts of the Slurm cluster. In simple words, the codes in the prolog script were written to stop HTCondor jobs and release resources for the coming Slurm jobs, while the codes in the epilog scripts were provided to start HTCondor jobs when the Slurm jobs are finished. The architecture of overlap is shown in figure 1.

The key configurations of overlap is listed in table 1.

From the test results showed in figure 2, it can be seen that HTCondor jobs could be run in the Slurm cluster with overlap.

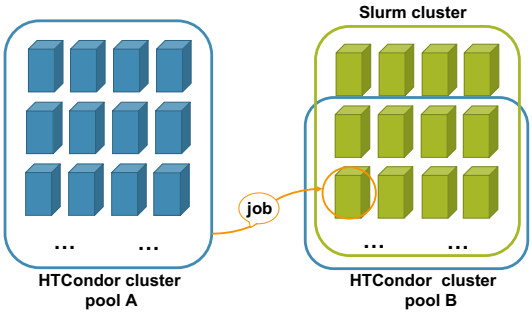

Figure 3: Architecture of flocking.

```
HTCondor Cluster : slurm04.ihep.ac.cn
Slurm Cluster : slurm03.ihep.ac.cn

#  check slots status
> condor_status
Name                     OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime
slot1@slurm04.ihep.ac.cn LINUX      X86_64 Claimed   Busy     0.060 32204  0+00:00:03
slot2@slurm04.ihep.ac.cn LINUX      X86_64 Claimed   Busy     0.000 32204  0+00:00:03
slot3@slurm04.ihep.ac.cn LINUX      X86_64 Claimed   Busy     0.000 32204  0+00:00:03
                       Machines Owner Claimed Unclaimed Matched Preempting  Drain
  X86_64/LINUX                3     0       3         0       0          0      0
         Total                3     0       3         0       0          0      0

# submit a new job which will be flocked to slurm03.ihep.ac.cn
> condor_submit submit_basic.jdf
Submitting job(s).
1 job(s) submitted to cluster 43.

# The new job is running on the Slurm cluster
> condor_q 43 -af remotehost
slot1@slurm03.ihep.ac.cn
```

Figure 4: Test results of flocking.

Table 2: Key configurations of flocking.

HTCondor cluster, slurm04.ihep.ac.cn : /etc/condor/config.d/conf
CONDOR_HOST = slurm04.ihep.ac.cn
FLOCK_TO = slurm03.ihep.ac.cn
FLOCK_COLLECTOR_HOSTS = $(FLOCK_TO)
FLOCK_NEGOTIATOR_HOSTS = $(FLOCK_TO)
Slurm cluster, slurm03.ihep.ac.cn : /etc/slurm/slurm.conf
# stop condor_startd when slurm jobs coming
Prolog=/usr/local/slurm/condor_off.prolog
# start condor_stard when slurm jobs finished
Epilog=/usr/local/slurm/condor_on.epilog
Slurm cluster, slurm03.ihep.ac.cn: /etc/condor/config.d/conf
FLOCK_FROM = slurm04.ihep.ac.cn

3.2 Flocking

According to HTCondor User Manual[11], Flocking is "HTCondor's way of allowing jobs that cannot immediately run within the pool of machines where the job was submitted to instead run on a different HTCondor pool". The difference between overlap and flocking is that flocking treats worker nodes from the Slurm cluster as the second pool, while overlap sees the worker nodes from Slurm cluster as the same pool. Figure 3 shows the architecture of flocking.

Similar with overlap, flocking also has the resource confliction issue to deal with. In order to address the issues, the same prolog and epilog scripts as the ones for overlap were adopted in the Slurm cluster. Table 2 lists the key configurations of flocking.

Flocking was tested, and the results showed that when there are no available resources on the worker nodes of the HTCondor cluster, the jobs were scheduled to the worker nodes of the Slurm cluster which is seen as the second pool of HTCondor. Figure 4 shows the test results.

3.3 HTCondor-C

HTCondor-C is the second native way to connect HTCondor with other batch systems, for example, Slurm in our case. HTCondor-C is responsible to migrate HTCondor jobs to the job queue of Slurm. During this migration, an open source software named blahp is taken advantage to transform an HTCondor job to a Slurm job. The architecture of HTCondor-C is shown in figure 5. It's not necessary to overlap work nodes between the HTCondor cluster

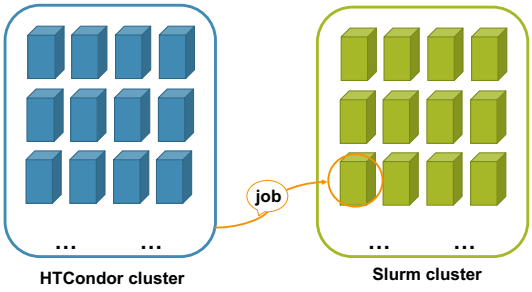

Figure 5: Architecture of HTCondor-C.

```
HTCondor Cluster : slurm04.ihep.ac.cn
# submit an grid universe job
> condor_submit submit_basic_condor_c.jdf
Submitting job(s).
1 job(s) submitted to cluster 41.

# check job status on slurm04.ihep.ac.cn
> condor_q
-- Schedd: slurm04.ihep.ac.cn : <192.168.51.47:6397> @ 07/02/18 18:04:18
OWNER BATCH_NAME      SUBMITTED   DONE  RUN   IDLE  TOTAL JOB_IDS
duran  CMD: sleep.sh   7/2  18:01     _     1     _      1 41.0

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
Slurm Cluster : slurm03.ihep.ac.cn
# check HTCondor job queue
> condor_q
-- Schedd: slurm03.ihep.ac.cn : <192.168.51.46:26921> @ 07/02/18 18:05:09
OWNER BATCH_NAME      SUBMITTED   DONE  RUN   IDLE  TOTAL JOB_IDS
duran  CMD: sleep.sh   7/2  18:01     _     1     _      1 7.0

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended

# check Slurm job queue
> squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                27      comp   bl_a7e50   duran  R       1:23      1 slurm03
```

Figure 6: Test results of HTCondor-C.

Table 3: Key configurations of HTCondor-C

HTCondor cluster, slurm04.ihep.ac.cn : /etc/condor/config.d/conf
CONDOR_HOST = slurm04.ihep.ac.cn
SBIN = /usr/sbin
CONDOR_GAHP = $(SBIN)/condor_c-gahp
HTCondor cluster, slurm04.ihep.ac.cn : /etc/condor/config.d/job_router
JOB_ROUTER_DEFAULTS
[requirements=target.WantJobRouter is True;
MaxIdleJobs = 10;
MaxJobs = 100;]
JOB_ROUTER_ENTRIES
[GridResource = "condor slurm04.ihep.ac.cn slurm04.ihep.ac.cn";
name = "slurm testbed";
set_remote_JobUniverse = 9;
set_remote_GridResource = "batch slurm";
set_remote_Requirements = False;]
JOB_ROUTER_SCHEDD2_NAME = slurm03.ihep.ac.cn
JOB_ROUTER_SCHEDD2_POOL = slurm03.ihep.ac.cn
DAEMON_LIST = $(DAEMON_LIST) JOB_ROUTER
Slurm cluster, slurm03.ihep.ac.cn: /etc/condor/config.d/conf
SEC_DEFAULT_NEGOTIATION = OPTIONAL
SEC_DEFAULT_AUTHENTICATION_METHODS = CLAIMTOBE

and the Slurm cluster. As a result, HTCondor-C has a clearer architecture compared with overlap and flocking.

To enable HTCondor-C, an HTCondor daemon named job_router is started. Settings and configurations of HTCondor-C are shown in table 3.

After HTCondor jobs were migrated to the job queue of the Slurm cluster, HTCondor could get status of the migrated jobs, at the same time, these migrated jobs were treated as Slurm jobs and managed according to Slurm scheduling algorithms. Test results of HTCondor-C is shown in figure 6.

3.4 Comparison of overlap, flocking and HTCondor-C

HTCondor jobs could be migrated to and running in Slurm cluster with all the three methods. However, there are preference considering configuration, management and scheduling polices. Table 4 shows the comparison results of overlap, flocking and HTCondor-C.

Table 4: Comparison of overlap, flocking and HTCondor-C

Metrics	**Overlap**	**Flocking**	**HTCondor-C**
Configuration	Scale up with worker nodes number	Scale up with pools number.	Scale up with cluster number
Customized job scheduling	Almost none	Almose none	Rich : Job Router, Slurm spank plugins, blahpd

It can be seen that though overlap and flocking are easier to configure, there are almost no mechanisms to customize scheduling algorithms which is quite important for the next development step. It would be better if customized scheduling algorithms could take effect, so that only qualified jobs could be migrated and run in the Slurm cluster. Besides, if circumstances changed, it would be more flexible to adjust with customized scheduling algorithms. For example, when there are more resources from the Slurm cluster provided for the migrated jobs, we could bring heavier workload to the Slurm cluster by adopting a greedy scheduling algorithm.

On the contrary, HTCondor-C provides job-router [11] to set some parameters related with job scheduling. Besides, a customized Slurm SPANK plugin [12] for job launching could be implemented. SPANK provides a generic interface to dynamically modify the job launch code in Slurm. Developers only need to compile SPANK plugins against Slurm‘s spank.h header file[13] and configure plugstack.conf file, and these plugins will be loaded at runtime during the next job launch. As a result, HTCondor-C is more preferred based on the requirements.

4 Job scheduling issues and possible solutions

The next step after job migration is job scheduling and job running. In this step, three issues are confronted, which are large job quantity, resource sharing and system environment. The following subsections describe the issues and possible solutions.

4.1 The issue of large job quantity

HTCondor is designed to schedule High Throughput Computing jobs, which means that HTCondor is good at dealing with large quantity of single-core jobs. On the contrary, Slurm is suitable to schedule multi-core jobs with high degree of parallelism while not good at handling large quantity of jobs[14].

To solve this issue, with HTCondor-C adopted, some limitations could be set in the job_router daemon of HTCondor, such as the maximum number of jobs allowed to migrate to the Slurm cluster. On the other side, a Slurm SPANK plugin could be developed to choose proper jobs from HTCondor to launch. With this SPANK plugin loaded, during the job selection procedure, if the workload of the Slurm cluster is heavy, the number of jobs to be launched could be limited. As a result, lighter scheduling burden is added to Slurm.

4.2 The issue of resource sharing

Computing resources of the HTCondor cluster and the Slurm cluster are funded by multiple experiments, and experiments supported by HTCondor and Slurm clusters are different. When some qualified HTCondor jobs are migrated to the Slurm cluster, a question is arisen that which resources should be allocated to the jobs submitted from different experiments.

To answer this question, a possible solution is to provide an extra Information System. The Information System is aimed to provide resource sharing information among multiple experiments. With this information, Slurm is able to allocate resources to the migrated jobs after share permission is checked.

4.3 The issue of system environment

As mentioned in the section 4.2, experiments supported by HTCondor and Slurm clusters are different. As a result, the system environment requirements to run jobs are different. If HTCondor jobs are migrated to Slurm cluster and ready to run, different system environment could be an issue. A possible solution to this issue is to adopt the container technology to provide lightweight virtual system environment. To make it working, both Docker and Singularity were investigated, and Singularity is more preferred for the moment because it's safer and supported by Slurm[15][16].

5 Conclusion

There are two aims of integrating workload between HTCondor and Slurm clusters. One is to improve resource utilization of the Slurm cluster, the other is to reduce job queued time of the HTCondor cluster. To testify that the integration is feasible, three ways were investigated to bridge HTCondor and slurm clusters. And HTCondor-C is more preferred because of rich customized scheduling strategies including HTCondor job-router configuration and Slurm SPANK plugins. To schedule and run HTC jobs on Slurm Cluster, three issues were proposed about large quantity jobs, resource sharing and system environment, and possible solutions are in progress. For the next step, singularity will be tested and developed more deeply to solve the system environment issue, later a Slurm SPANK plugin will be implemented for the large job quantity issue, and the resource sharing Information System is coming at last.

6 Acknowledgements

We wish to thank all the colleagues who contributed to this proceeding. The work presented is supported by the National Key Research and Development Program of China "Development of High Performance Application Software System for Scientific Discovery in High Energy Physics" (No. 2017YFB0203203), the National Natural Science Foundation of China "The Two-staged Job Scheduling Algorithms and Resource Management Research about Workload Integration between HTC Cluster and HPC cluster" (No. 11805225), and the National Natural Science Foundation of China "Container Virtualization Applied to High Energy Physics Computing" (No. 11775250).

References

[1] B. O'Brian, R. Walker, A. Washbrook, *Leveraging HPC resources for High Energy Physics*, in *Proceedings of CHEP2013* (2013), p. 032104

[2] D. Cameron, A. Filipcic, W. Guan, V. Tsulaia, R. Walker, T. Wenaus, *Exploiting opportunistic resources for ATLAS with ARC CE and the Event Service*, in *Proceedings of CHEP2017* (2017), p. 052010

[3] A. Filipcic, *Integraton of the Chinese HPC Grid in ATLAS Distributed Computing*, in *Proceedings of CHEP2017* (2017), p. 082008

[4] A. Filipcic, S. Haug, M. Hostettler, R. Walker, M. Weber, *ATLAS computing on CSCS HPC*, in *Proceedings of CHEP2015* (2015), p. 092011

[5] D. Hufnagel, *CMS use of allocation based HPC resources*, in *Proceedings of CHEP2017* (2017), p. 092050

[6] G. Erli et al., *On-demand provisioning of HEP compute resources on cloud sites and shared HPC centers*, in *Proceedings of CHEP2017* (2017), p. 052021

[7] M. Fasel, *Using NERSC High-Performance Computing (HPC) systems for high-energy nuclear physics applications with ALICE*, in *Proceedings of ACAT2016* (2016), p. 012031

[8] W. Bhimji et al., *Extreme I/O on HPC for HEP using the Burst Buffer at NERSC*, in *Proceedings of CHEP2017* (2017), p. 082015

[9] L. Gerhardt, W. Bhimji, S. Canon, M. Fasel, D. Jacobsen, M. Mustafa, J. Porter, V. Tsulaia, *Shifter: Containers for HPC*, in *Proceedings of CHEP2017* (2017), p. 082021

[10] C. Hollowell, J. Barnett, C. Caramarcu, W. Streckerkellogg, A. Wong, A. Zaytsev, *Mixing HTC and HPC Workloads with HTCondor and Slurm*, in *Proceedings of CHEP2017* (2017), p. 082014

[11] *Htcondor user manual*, http://research.cs.wisc.edu/htcondor/manual/v8.6/index.html, online, accessed 18-Nov-2018

[12] *Slurm spank plugins*, https://slurm.schedmd.com/download.html, online, accessed 16-Feb-2019

[13] *Slurm spank header file*, https://github.com/SchedMD/slurm/blob/master/slurm/spank.h, online, accessed 16-Feb-2019

[14] *Slurm high throughput computing*, https://slurm.schedmd.com/high_throughput.html, online, accessed 16-Feb-2019

[15] *Docker security document*, https://docs.docker.com/engine/security/security/, online, accessed 16-Feb-2019

[16] *Slurm container document*, https://slurm.schedmd.com/containers.html, online, accessed 16-Feb-2019