Journal of Physics: Conference Series

PAPER • OPEN ACCESS

Experience on HTCondor batch system for HEP and other research fields at KISTI-GSDC

To cite this article: S U Ahn *et al* 2017 *J. Phys.: Conf. Ser.* **898** 082013

View the article online for updates and enhancements.

Related content

- Grids, virtualization, and clouds at Fermilab
S Timm, K Chadwick, G Garzoglio et al.

- Geographically distributed Batch System as a Service: the INDIGO-DataCloud approach exploiting HTCondor
D C Aiftimiei, M Antonacci, S Bagnasco et al.

- Pushing HTCondor and glideinWMS to 200K+ Jobs in a Global Pool for CMS before Run 2
J Balcas, S Belforte, B Bockelman et al.

This content was downloaded from IP address 131.169.5.251 on 03/12/2017 at 21:08

Experience on HTCondor batch system for HEP and other research fields at KISTI-GSDC

S U Ahn, A Jaikar, B Kong, I Yeo, S Bae and J Kim

Global Science experimental Data hub Center, Korea Institute of Science and Technology Information, 245 Daehak-ro Yuseong-gu, 34141 Daejeon, Republic of Korea

E-mail: sahn@kisti.re.kr

Abstract. Global Science experimental Data hub Center (GSDC) at Korea Institute of Science and Technology Information (KISTI) located at Daejeon in South Korea is the unique datacenter in the country which helps with its computing resources fundamental research fields dealing with the large-scale of data. For historical reason, it has run Torque batch system while recently it starts running HTCondor for new systems. Having different kinds of batch systems implies inefficiency in terms of resource management and utilization. We conducted a research on resource management with HTCondor for several user scenarios corresponding to the user environments that currently GSDC supports. A recent research on the resource usage patterns at GSDC is considered in this research to build the possible user scenarios. Checkpointing and Super-Collector model of HTCondor give us more efficient and flexible way to manage resources and Grid Gate provided by HTCondor helps to interface with the Grid environment. In this paper, the overview on the essential features of HTCondor exploited in this work is described and the practical examples for HTCondor cluster configuration in our cases are presented.

1. Introduction

Global Science experimental Data hub Center (GSDC) [1] at Korea Institute of Science and Technology Information (KISTI) [2] is a datacenter which was built by a national funding project to promote fundamental research activities in South Korea. Mainly GSDC provides storage resources to the research groups, who manipulate instruments producing large amounts of data. The volume of data mentioned here is far exceeding the scale of the size which can be dealt within a small lab: from tens of terabytes up to more than tens of petabytes. Currently GSDC supports 6 experiments: ALICE [3], CMS [4], LIGO [5], Belle2 [6], RENO [7] and Genome project. In addition to storage resources, GSDC provides computing power in order to process the large scale of data produced by the experiments. The computing resources of GSDC are managed by the distributed computing resource management system, which is so-called batch system, and the systems used at GSDC were Torque [8] and HTCondor [9]. Which batch system should be used was determined by the dependencies of the software suites that are used or recommended by the research groups. In particular, the Grid middleware suites deployed for the research communities, e.g. EMI [10] for ALICE and Belle2 experiments, or OSG [11] for CMS experiment, was one of the crucial factor for the selection of batch system. In GSDC, Torque batch system was used for ALICE, Belle2 and RENO experiment, and HTCondor batch system was used for CMS, LIGO and Genome project.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

2. Resource Utilization at GSDC

In 2015, we conducted a research to estimate the utilization of batch systems for ALICE, Belle2, CMS and LIGO experiment at GSDC [12]. Both Torque batch systems for ALICE and Belle2 experiment were interfaced with EMI middleware while one HTCondor batch system for CMS was interfaced with the OSG middleware however, the other HTCondor batch system for LIGO were not interfaced with any Grid middleware. The batch systems for CMS and LIGO were used by only local users. The research showed that the estimations on resource utilization of batch systems for ALICE, Belle2, CMS and LIGO were 90%, 65%, 0.7% and 25% respectively as shown in Figure 1. In the research, the resource utilization was defined as the ratio between the entire time of the year and the sum of total wall clock time consumed by jobs. The data used in the measurement was the job accounting data of batch systems for 2015. Consistent trend in utilization for the entire year was shown for ALICE(A) and Belle2(B) due to high job throughput submitted from Grid. On the contrary, the resource utilization of CMS(C) and LIGO(D) was very low and the trend was unpredictable behavior throughout the year due to the chaotic usage performed by local users.

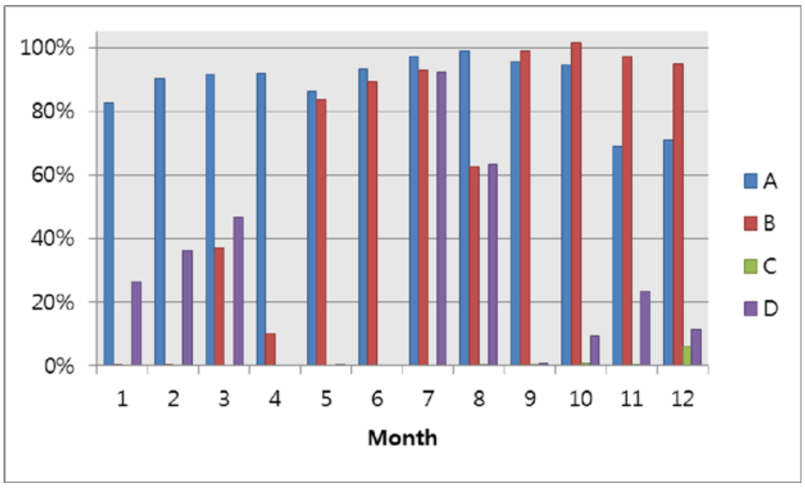


Figure 1. Measurement on resource utilizations for ALICE(A), Belle2(B), CMS(C) and LIGO(D) based on accounting data collected by Torque and HTCondor batch systems in 2015 [12]. The number of job slots, which is equal to the number of logical cores, provided to the experiments are 3,500 for ALICE, 300 for Belle2, 1,000 for CMS and 800 for LIGO.

The dramatic discrepancy between the utilizations can be moderated if the idle resources of one group were allowed to be used by other groups. However, there were two constraints:

- In GSDC, computing resources were dedicated to each experiment according to Memorandum of Understanding (MoU) agreed with the communities. The fulfillment of the MoU, i.e. the physical allocation of resource has been audited regularly. Therefore we provided batch systems separately to each experiment and the fair-sharing was disabled.
- The queues of Torque and HTCondor batch systems cannot be shared since they are different systems. This implies that jobs submitted by ALICE and Belle2 experiment cannot be processed in HTCondor batch systems dedicated to CMS and LIGO experiment.

3. Moving towards HTCondor

The goal of this work was to improve the situation in which the resource utilization was unbalanced between two different batch systems we currently suffer at GSDC. In 2012, we built up a Torque batch system for WLCG Tier-1 center supporting ALICE experiment and we have had an instability issue with Torque since then. About 1,500 job slots were available in Torque batch system and the queue was managed by MAUI scheduler. Because the MAUI scheduler had to be restarted every 3 to 4 days for unknown reasons, a probe that checks the status of the MAUI scheduler was scheduled to run periodically, so that the MAUI service can be restarted when the MAUI service is malfunctioning. The other issue was with the node list of Torque batch system in which it is required batch system to be restarted after the update of the list. These were inefficient for managing a datacenter with large scale of resource pool (about thousands of nodes).

HTCondor was a promising candidate to replace Torque batch system because we already had experience on the administration of HTCondor. However there were two constraints on the replacement of Torque by HTCondor:

- Although we had operation experience of HTCondor batch system, we had few knowledge on resource scheduling for large pool shared among different experiments since we have had the resource dedication policy; we ignored the idle resources of a group.
- Torque batch system was preferred for ALICE and Belle2 experiments to interface with the Grid environment since CREAM-CE, which is a component of EMI middleware functioning as to authenticate and authorize jobs submitted through the Grid before the local batch systems, does not officially support HTCondor batch system. Recently HTCondor-CE for HTCondor batch system to be interfaced with Grid has been getting popular and the ALICE experiment has started to support HTCondor.

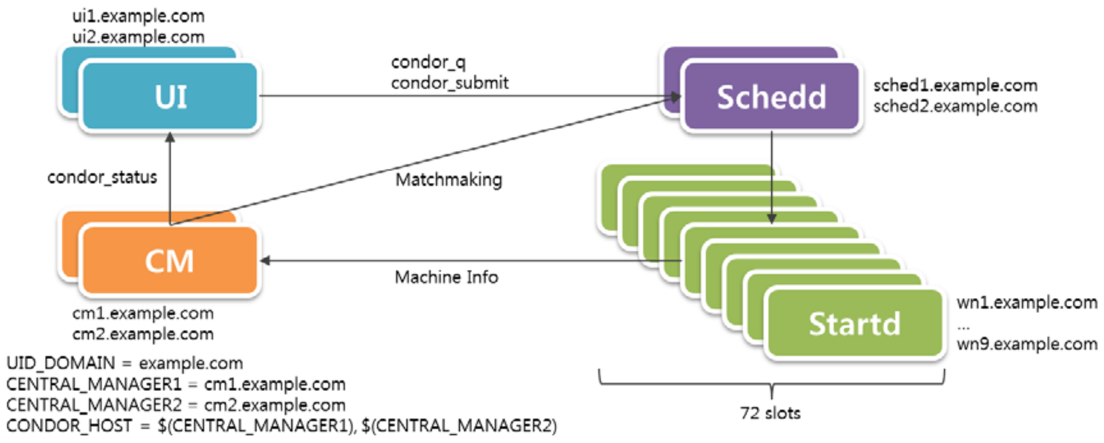
4. Requirements for the New Pool

Our goal was to replace Torque batch system by HTCondor batch system and to share a unified pool with various experiments we support. As described in the previous section, we need to gain further knowledge on HTCondor batch system such as dynamic resource allocation or HTCondor-CE for Grid interface. For this purpose, we built up a test-bed to examine the following requirements:

- Computing resources allocation in the pool had to be managed dynamically. Idle resources belonging to one group should be available to other groups in case of which they demand additional resources exceeding their quota. However, the pledged resources for a group must be guaranteed when the resources occupied by others are claimed by its owner. This is called preemption. To give a better user experience, a delicate and complementary policy on the treatment for the preempted jobs, which are the jobs evicted from where they were running temporarily, has to be considered and implemented.
- Even though computing resources in the new pool were shared for several groups, we liked to separate User Interface (UI) among those groups for better user experience and the convenience of system management. Some cases requiring compilation on real-time before job submission, which may consume resources of the machine, may affect badly other user's experience in case of which UI being shared. Also the exposure of mount points of experiment data and user's scratch to other groups may have potential security vulnerability although the access by others is not permitted.
- Enabling high availability of daemons on submission and central manager nodes was essential to achieve highly available HTCondor pool. In HTCondor batch system, schedd daemon on submission node manages submitted jobs, and collector and negotiator daemons on central manager node collect cluster information in the pool and perform match-making between job and resource specifications, respectively. In principle, one submission node runs one schedd daemon and in general the submission node is used as the UI. As described in the second requirement, we liked to have separate UIs for each group while we have schedd machines in high availability mode. This implied that the UI machines and schedd machines are not identical so that jobs should be submitted from UI machines to schedd machines remotely.

5. A Test-bed Setup for High Availability of Daemons

In order to accommodate the requirements described in the previous section, we have built up a test-bed by deploying HTCondor batch system to a new pool. As shown in Figure 2, the architecture of the test-bed was rather simple however user interface, submission and central manager nodes were deployed on at least two nodes for the configuration of high availability of the daemons. In the pool, 72 job slots was provided by nine execution nodes. In HTCondor



```
UID_DOMAIN = example.com
CENTRAL_MANAGER1 = cm1.example.com
CENTRAL_MANAGER2 = cm2.example.com
CONDOR_HOST = $(CENTRAL_MANAGER1), $(CENTRAL_MANAGER2)
```

Figure 2. A brief structure of the test-bed with hostnames of each machine, basic mechanism among nodes and common configuration shared for HTCondor batch system.

batch system, a basic configuration [13] for high availability of schedd daemon is to make the schedd daemon on one submission node to be active while the other schedd daemons on other nodes to be standby. This is because basically only one schedd daemon in a pool can solely run in the HTCondor batch system. Active-standby mode for schedd daemons was controlled by locking mechanism. A lock file owned by the active daemon was located in a shared file system via network so that the other schedd daemons could access the lock file. In such a way that checking the ownership of the lock file, the other schedd daemons could detect the active daemon's status. Ownership can be taken by one of the other standby daemons if the active daemon is dead or can not temporarily update the file status for unknown reasons. The schedd daemons, whether active or standby, were referred by a single common name, i.e. "had_schedd" in this work. In order for center managers to be high availability mode, all candidate central managers had the same configuration of daemon list and the pool information were shared. In HTCondor batch system, the replication daemon carries out periodic transfer of any file stated in "STATE_FILE" macro in the configuration from the primary central manager to the secondary manager. By stating the location of pool information file in the macro, the pool information can be shared among candidates so that immediate transition of the central manager role can be done when it is needed.

6. Dynamic Resource Allocation with Accounting Group

A negotiator-side resource management in the HTCondor pool [14] was implemented to achieve the dynamic resource allocation, which was one of the requirements described in the previous section. Negotiator daemon of HTCondor supports group quota which can be defined by accounting group parameters. The essential parameters for accounting group are described in Table 1. Firstly, we defined accounting group and set limit (quota) on resources based on the MoU for each experiment. Secondly, we allowed users to exceed their group quota when

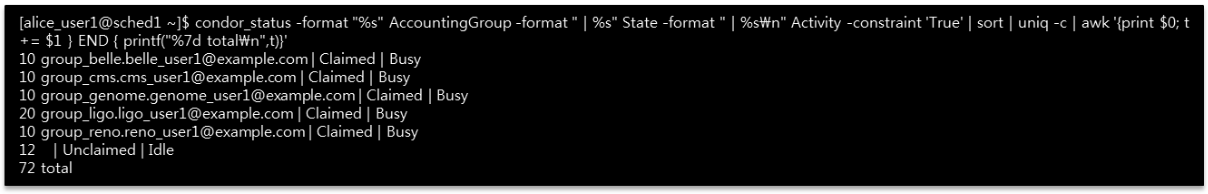
Table 1. A summary of accounting group parameters used in the test-bed.

Parameter	Value
GROUP_NAMES	<*arbitrary name for experiment*>
GROUP_QUOTA_<*arbitrary name for experiment*>	<*number of slots* >
GROUP_ACCEPT_SURPLUS	*TRUE or FALSE*
NEGOTIATOR_CONSIDER_PREEMPTION	*TRUE or FALSE*
PREEMPTION_REQUIREMENTS	<*conditional statement* >

resources belong to other groups are not claimed. Finally, we enabled prompt preemption to guarantee group quota of each experiment when the owner claims the resources if theirs were being used by others. These configurations were placed on negotiator daemon. Practical examples of negotiator-side configuration used in the test-bed are the following:

```
GROUP_NAMES = group_alice, group_cms, group_ligo, group_belle, group_reno,
              group_genome
GROUP_QUOTA_group_alice = 40
GROUP_QUOTA_group_cms = 12
GROUP_QUOTA_group_ligo = 8
GROUP_QUOTA_group_belle = 4
GROUP_QUOTA_group_reno = 4
GROUP_QUOTA_group_genome = 4
GROUP_ACCEPT_SURPLUS = True
NEGOTIATOR_CONSIDER_PREEMPTION = True
PREEMPTION_REQUIREMENTS = (PREEMPTION_REQUIREMENTS) &&
                (((SubmitterGroupResourcesInUse < SubmitterGroupQuota) &&
                (RemoteGroupResourcesInUse > RemoteGroupQuota)) ||
                (SubmitterGroup =?= RemoteGroup))
```

A demonstration of the dynamic resource allocation was performed based on the configuration described in this section. In this demo, we assumed that there is no job processing activity from *group_alice* while all other groups are sharing the resources belong to *group_alice*, i.e. 40 slots. In this situation, a *group_alice* user confirms that only 12 slots are available when he or she checked the status of the pool. This is shown in Figure 3. As any *group_alice* user can claim their quota,

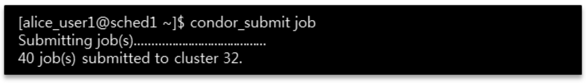


```
[alice_user1@sched1 ~]$ condor_status -format "%s" AccountingGroup -format " | %s" State -format " | %s₩n" Activity -constraint 'True' | sort | uniq -c | awk '{print $0; t
+= $1 } END { printf("%7d total₩n",t)}'
10 group_belle.belle_user1@example.com | Claimed | Busy
10 group_cms.cms_user1@example.com | Claimed | Busy
10 group_genome.genome_user1@example.com | Claimed | Busy
20 group_ligo.ligo_user1@example.com | Claimed | Busy
10 group_reno.reno_user1@example.com | Claimed | Busy
12    | Unclaimed | Idle
72 total
```

Figure 3. Demo: the status of HTCondor pool when a *group_alice* user logged into the submission node.

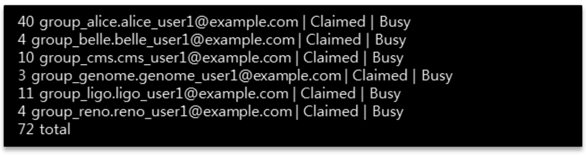
when *group_alice* user submitted 40 jobs, the preemption was placed immediately so that the jobs submitted by *group_alice* user could be successfully allocated to the reserved resources and the jobs owned by other groups were evicted as shown in Figure 4 and 5. However, *group_ligo* still consumed resources above its quota which was taken from unused resources of *group_cms* and the quota for *group_genome*.

In this demonstration, we showed that preemption was performed based on accounting group quota configured on negotiator side however, the behavior of quota was not accurately carried out, which requires further investigation. HTCondor batch system provides *checkpointing* for the evicted jobs which helps the jobs to be gently suspended with saving their progress and to be resume in other places. In order to enable the checkpointing for a job, the job should be re-linked with *condor_compile* and submitted within a specific preset provided by HTCondor batch system, which is called *Standard Universe*. There is a workaround [15] to feature the checkpointing with the *Vanilla Universe*, which is the other preset preferrly used by most of jobs, exploiting BLCR kernel module.



```
[alice_user1@sched1 ~]$ condor_submit job
Submitting job(s)........................................
40 job(s) submitted to cluster 32.
```

Figure 4. Demo: *group_alice* user claims the quota by submitting jobs.



```
40 group_alice.alice_user1@example.com | Claimed | Busy
4 group_belle.belle_user1@example.com | Claimed | Busy
10 group_cms.cms_user1@example.com | Claimed | Busy
3 group_genome.genome_user1@example.com | Claimed | Busy
11 group_ligo.ligo_user1@example.com | Claimed | Busy
4 group_reno.reno_user1@example.com | Claimed | Busy
72 total
```

Figure 5. Demo: status of the pool after *group_alice* user submits jobs.

7. Conclusions

We, GSDC at KISTI, support several experiments as a datacenter and manage the resources with Torque and HTCondor batch systems. Since we have resource dedication policy and the resources are managed by two different batch systems, we have experienced inefficient resource utilization. In order to address this, we need to integrate our batch systems into one and to enable dynamic resource allocation among several experiments. HTCondor batch system was chosen because we have experienced unstable behavior of Torque batch system when it manages large resources and its architecture is not scalable.

In this paper, we described the set-up of a test-bed with HTCondor batch system that accepts dynamic resource allocation policy based on negotiator-side configuration, using parameters related to accounting group. At the same time, we examined the default HTCondor configuration for high availability of the daemons for submission nodes and central manager nodes. We demonstrated the dynamic resource allocation with accounting group quota with the test-bed and we need further investigation on the behavior of quota. In the demonstration, we learned that delicate policies should be established when dealing with preempted (evicted) jobs and their re-allocation. Checkpointing in HTCondor would help the evicted jobs to be resumed in the other places not to waste their CPU clock time already consumed. For the future work, we will enable job to be submitted from separate UIs to remote schedd machines in the test-bed, and develop and expand the test-bed to be deployed in the production level. And also HTCondor-CE will be examined for interfacing the HTCondor batch system with Grid middleware suites.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) through contract N-16-NM-CR01-S01 and the Program of Construction and Operation for Large-scale Science Data Center (K-17-L01-C05-S01).

References

[1] Ahn S U 2017 *Proc. International Symposium on Multiparticle Dynamics (Jeju)*

[2] Korea Institute of Science and Technology Information (KISTI) http://en.kisti.re.kr
[3] The ALICE Collaboration http://alice-collaboration.web.cern.ch
[4] The CMS Collaboration https://cms.cern
[5] The LIGO Scientific Collaboration http://www.ligo.org
[6] The Belle II Collaboration https://www.belle2.org
[7] RENO: An Experiment for Neutrino Oscillation Parameter theta_13 Using Reactor Neutrinos at Yonggwang arXiv:1003.1391 [hep-ex]
[8] Tera-scale Open-source Resource and QUEue manager (TORQUE) http://www.adaptivecomputing.com/products/open-source/torque/
[9] Litzkow M and Linvy M 1990 *Proc. IEEE Workshop on Experimental Distributed Systems (Huntsville)*
[10] EMI - European Middleware Initiative http://wlcg.web.cern.ch/emi-european-middleware-initiative
[11] OSG - Open Science Grid https://www.opensciencegrid.org
[12] Ahn S U and Kim J 2016 *Proc. Int. Conf. on Platform Technology and Service (Jeju)*
[13] HTCondor Version v8.4 Manual http://research.cs.wisc.edu/htcondor/manual/v8.4
[14] Maintaining Accounting Group Quotas With Preemption Policy http://erikerlandson.github.io/blog/2012/06/27/maintaining-accounting-group-quotas-with-preemption-policy/
[15] Checkpointing vanilla jobs with BLCR http://help.uis.cam.ac.uk/supporting-research/research-support/camgrid/camgrid/technical3/blcr