A new Self-Adaptive Dispatching System for local clusters

Bowen Kan¹, Jingyan Shi¹, Xiaofeng Lei¹²

¹Institute of High Energy Physics, Beijing, China
²Graduate University of Chinese Academy of Sciences, Beijing, China

E-mail: kanbw@ihep.ac.cn

Abstract. The scheduler is one of the most important components of a high performance cluster. This paper introduces a self-adaptive dispatching system (SAPS) based on Torque¹¹ and Maui²². It promotes cluster resource utilization and improves the overall speed of tasks. It provides some extra functions for administrators and users. First of all, in order to allow the scheduling of GPUs, a GPU scheduling module based on Torque and Maui has been developed. Second, SAPS analyses the relationship between the number of queueing jobs and the idle job slots, and then tunes the priority of users’ jobs dynamically. This means more jobs run and fewer job slots are idle. Third, integrating with the monitoring function, SAPS excludes nodes in error states as detected by the monitor, and returns them to the cluster after the nodes have recovered. In addition, SAPS provides a series of function modules including a batch monitoring management module, a comprehensive scheduling accounting module and a real-time alarm module. The aim of SAPS is to enhance the reliability and stability of Torque and Maui. Currently, SAPS has been running stably on a local cluster at IHEP (Institute of High Energy Physics, Chinese Academy of Sciences), with more than 12,000 cpu cores and 50,000 jobs running each day. Monitoring has shown that resource utilization has been improved by more than 26%, and the management work for both administrator and users has been reduced greatly.

1. Introduction

Currently there are more than 12,000 cpu cores and 256 GPU cards running on the local cluster at IHEP (Institute of High Energy Physics, Chinese Academy of Sciences). The cluster provides computing services for several HEP experiments including BESIII³³, Yangbajing International Cosmic Ray Observatory in Tibet⁴⁴, the Daya Bay Reactor Neutrino Experiment⁵⁵, CMS⁶⁶ and ATLAS⁷⁷. Other experiments currently being planned, such as the Jiangmen Underground Neutrino Observatory⁸⁸ and the Large High Altitude Air Shower Observatory⁹⁹ will be supported in the near future. A local high performance cluster is a good choice for data analysis, and an efficient scheduler is one of the most important components for its operation.

There are many kinds of schedulers running at different clusters. HTCondor¹⁰¹⁰ is free, open-source and developed at the University of Wisconsin-Madison. It obeys the license of Apache 2.0¹¹¹¹. HTCondor is good at scheduling intensive serial tasks with very nice efficiency. PBS Professional¹²¹² has been used by thousands of cluster worldwide. This is the commercial software developed by Altair for high-performance computing environments. It provides clusters with good resource utilization and efficiency. However, its high commercial cost prevents a lot of clusters from using it. SLURM¹³¹³, acronym for Simple Linux Utility for Resource Management, is an open source software. It runs on
Linux cluster and is suitable for all sizes of the cluster. The resource management of the world fastest supercomputer Tianhe-2, according to Top 500 list[14], is based on SLURM. SLURM is a fast and reliable batch system solution as proven by tests carried out by INFN[15]. However, SLURM is famous for its complicated management.

Torque and Maui are derived from open source PBS[16] which had been widely used ten years before. Torque is a resource manager developed by NASA[17] Ames Research Center. It is easy to be configured and is capable of supporting a large scale cluster. Torque has the share capabilities. It has a very simple scheduler pbs_sched, which only has FCFS (First Come First Serve) scheduling policy. Obviously this cannot satisfy most of clusters requirements. Maui is the predecessor of the commercial scheduler Moab. It is an open source scheduler with rich scheduling policies. As it can be used as an external scheduler, the most common use of the cluster is to run Torque with Maui for the management of cluster resources and job scheduling.

2. ARCHITECTURE

Torque and Maui are used at the IHEP local cluster. Since Maui does not support GPU (graphics processing unit) scheduling, we needed to find a way to schedule jobs to GPU cards. Inefficient resource utilization was an additional problem. There are more than 60 queues configured at the IHEP local cluster, and each queue has the “max_user_run” attribute to avoid too many resources being occupied by one user. This caused some job slots to be idle even when there were jobs in the queue, since the number of running jobs owned by a given user had reached the “max_user_run” limit. Another problem was system stability. The cluster has been expanded step by step, so both new and old work nodes are running together. The hardware error rate of the old nodes is much higher than that of the new ones. An unexpected hardware error on an old work node causes the work node to fail to respond to the periodic enquiry from the Torque server. The Torque server has to wait for a rather long time to make sure the enquired node is down. The waiting period occupies the tcp port and this blocks communication between the Torque server and other nodes. Another problem was that some unhealthy nodes could not run jobs, but the scheduler was not aware of it. For example, if no data disk was mounted at a work node, all jobs scheduled to that work node would fail at once. As the scheduler was not aware of the problem, it would keep scheduling jobs to the problematic node.

In order to solve these problems, we designed and implemented a new SAPS system, acronym for Self-Adaptive DisPatching System, based on Torque and Maui. Besides all the functions of Torque and Maui, SAPS provides a GPU card scheduling function module for GPU jobs, a new scheduling algorithm to optimize resource utilization, and a monitoring module to exclude work nodes in error states.

Figure 1 shows the architecture of SAPS. In the SAPS system, the users submit jobs to the user server, which is in fact a proxy passing the job to the resource scheduler. After receiving the jobs from the user server, the resource scheduler distinguishes them as GPU jobs or CPU jobs based on the job requirements. Different jobs are scheduled through different job management interfaces to a suitable work node. The resource management module collects the work node status from the cluster and stores them to the information storage. The work nodes status from the resource management module are provided to the reporting module, alarm module and monitoring module, and some web pages are provided by those modules.
3. Implementation

3.1. GPU scheduling module

Commodity GPUs have evolved rapidly and have been recognized as high performance accelerators for data-parallel computing at IHEP local cluster. Each GPU card can only be occupied by one job at a time. That is quite different from CPUs. A dedicated GPU scheduling policy is therefore necessary. Figure 2 shows the data flow diagram in GPU scheduling.

Users submit jobs to the corresponding queues. If the job is GPU card dependent (GPU Jobs), it is considered a GPU job. The program will keep checking until idle resources are available for the waiting jobs. Once the required resources are assigned, the job is marked in the resource management
and the stored information is updated. Finally, the jobs and resource reservation information are sent to Torque and Maui.

If the job does not require a GPU card, it is considered a regular job (CPU Jobs). In this case, the module simply keeps checking the availability of CPU resources and finally hands the job over to Torque and Maui directly. Finally, the storage is updated with the number of CPUs occupied from Torque and Maui.

3.2. Resolving resource inefficiency

After observation, we noticed that the key reason of low resource utilization of the IHEP local cluster was that there was no prediction of the number of users and submitted jobs. To guarantee each active user has job slots to run their jobs, a relatively small value was assigned to ‘max_user_run’. This attribute let the scheduler dispatch job slots to each user fairly, but when there were few users but many jobs in the queue, the attribute blocked the scheduler from dispatching more job slots to one user. So the cluster had idle job slots while there were long job queues. The solution to this problem was to tune ‘max_user_run’ dynamically according to the number of active users, running jobs, queuing jobs and idle job slots.

In this solution, we mean by active users those whose jobs are more than 1/20 times the queue’s idle job slots. The “max_user_run” attribute of each queue is equal to the number of idle job slots multiplied by the number of active users divided by the number of queued jobs. When the number of active users is more than 20, the Job Max-Running Number (JMRN) of the queue is:

\[
jmrn = \frac{\text{idle job slots} \times \text{active users}}{\text{queuing jobs}}
\]

When the number of active users is less than 20, however, the user Job Max-Running Number (JMRN) of the queue is:

\[
jmrn = \frac{\text{idle job slots}}{\text{active users}}
\]

These two simple formulas maximize the resource utilization of the IHEP local cluster. Currently, ‘1/20’ and ‘20’ are static values chosen after numerous tests at the IHEP local cluster. This static value can be changed manually according to the total resources and the numbers of users of the local cluster. Actually these two static elements might not be the best choice of JMRN, depending on the status of the cluster. Experience shows that when 20 is chosen as the static value, the utilization of resources is much better than before. We cannot currently get the optimal value of this element analytically, so that might be our next plan.

3.3. Improving system stability

We noticed that the biggest threat to system stability comes from unexpected errors happening to the work nodes. The failed work nodes block communication between the healthy work nodes and the scheduling server (Torque), which means that new jobs cannot be scheduled smoothly. Besides, the failed work nodes also cause many job failures. A large number of jobs were scheduled to the failed work nodes and jobs running on these work nodes failed quickly in an infinite loop, like a “black hole” in the cluster.

![Figure 3 self-inspection workflow](image)
To prevent the “black hole” node effect, we designed and implemented a self-inspection daemon which runs at each work node. The self-inspection daemon checks the memory usage, CPU usage, GPU card status, zombie processes, disk usage and so on. If any errors are detected, the self-inspection daemon first tries to fix it automatically. If there are fatal errors, such as hardware errors, which cannot be fixed by the daemon, it will send a message to the resource management in order to stop jobs from being scheduled at the node. As soon as the message is received, the failed work nodes are removed from the local cluster by the resource management and the administrator is notified to check the node manually. After the fatal errors are fixed, the self-inspection daemon tells the resource management to restore the recovered node to the cluster.

4. Other functions developed in SAPS

4.1. SAPS monitoring management

In order to simplify the administrators’ work, we designed and implemented a SAPS monitoring management module. We collect cluster running status from resource management and save them as JSON files. The monitoring module provides some monitoring web pages in HTML5 based on those JSON files. Figure 4 shows some screenshots from the monitoring management. Figure 4(a) shows the total resource utilization and the number of active users. Figure 4(b) shows the job cpu time distribution, where we can notice that most jobs are finished within 1 hour. Figure 4(c) gives the number of failed jobs: it provides a view of the job status for administrators. Figure 4(d) is part of the SAPS monitoring management, showing job scheduling statistics in recent period.

![Figure 4 SAPS monitoring management](image-url)
4.2. Alarm system

An alarm system guarantees the local cluster is running stably. The alarm system checks the scheduler on line and report job errors to users and administrators via email. The alarm email provides the user with all the information of the failed jobs including job id, job name, error time, work node, cputime, walltime, memory usage and possible error reasons. Figure 5 shows an example of the alarm email.

4.3. Report module

The report module, based on scheduler log analysis, records all job information, such as the number of jobs, failure rate, amount of resources consumed etc. The report can be reached from the web page or email. Figure 6 shows the web page of the cluster monthly report.

5. System Test

We have done some tests to compare SAPS to the original Torque and Maui. The tests were done at a cluster with 3340 job slots. 40 test users and 4000 test jobs were simulated. Both test users and test queues were from the real cluster. All the jobs were submitted at random times and each job was set to just sleep for a random length of time.

Figure 7 Experimental Results

Figure 7(a) shows the cluster resource utilization with the original Torque and Maui; Figure 7(b) shows the results when using SAPS. The blue area represents the total number of cpu cores in the test cluster; the red area is the number of cpu cores occupied by running jobs; the green bars show the number of queuing jobs; the blue dots show the total number of users; and the red dots show the number of queuing users.
The figure shows quite a satisfactory result. The number of queuing jobs was reduced sharply from 2000 to 500. The utilization of cpu cores was increased significantly from 74% to almost 100%.

6. Conclusion

SAPS has been successfully tested on our cluster at IHEP. The GPU scheduling with multi-core based on Torque and Maui dispatches GPU jobs according to their requirements. The rate of resource utilization at the local cluster has been significantly increased and the system stability has been improved greatly. In addition, SAPS provides monitoring, alarm and report functions which have simplified the administrators’ work and provide more services to users.

However, there are some functions which still need to be developed. The optimization of the attribute “max_user_run” just focuses on queues. A new finer granular way with the focus on users should also be considered, and an analytical way found to tune the value of this attribute.

7. Acknowledgement

We would like to thank all members of the IHEP-CC scheduling monitoring teams, and Prof. Sun Gongxing who helped with the system architecture, deployment and operations. We would also like to thank the BESIII team and other system administrators for their support.

This work was supported by the Xie Jia Lin foundation of IHEP under Contract No.Y5546140U2 and the National Natural Science Foundation of China (NSFC) under Contracts No. 111147521070.

References
[1] http://docs.adaptivecomputing.com/torque/6-0-0/help.htm
[2] http://docs.adaptivecomputing.com/maui/index.php
[3] http://bes3.ihep.ac.cn
[4] http://english.ihep.cas.cn/rs/fs/ybj/
[5] http://english.ihep.cas.cn/rs/fs/dyb/
[6] http://cms.web.cern.ch/
[7] http://atlas.ch/
[8] http://english.ihep.cas.cn/rs/fs/juno0815/
[9] http://english.ihep.cas.cn/ic/ip/LHAASO/
[10] Thain, Douglas; Tannenbaum, Todd; Livny, Miron (2005). "Distributed Computing in Practice: the Condor Experience". Concurrency and Computation: Practice and Experience 17 (2–4): 323–356. doi:10.1002/cpe.938.
[11] HTCondor License Agreement
[12] http://www.pbsworks.com/
[13] http://slurm.schedmd.com/overview.html
[14] http://www.top500.org/
[15] Giacinto Donvito; Davide Salomoni; Alessandro Italiano. “Testing SLURM open source batch system for a Tier1/Tier2 HEP computing facility”; Journal of Physics: Conference Series Volume 513 032027 doi:10.1088/1742-6596/513/3/032027
[16] Albeaus Bayucan; Casimir Lesiak; Bhroam Mann; Tom Proett; Dave Tweten.(1998) “Portabl Batch System Internal Design Specification” OpenPBS Public Home http://www.mcs.anl.gov/research/projects/openpbs/
[17] http://en.wikipedia.org/wiki/NASA