Batch efficiency

Ulrich Schwickerath, Ricardo Silva, Christian Uria
CERN IT, 1211 Geneve 23, Switzerland
E-mail: `Ulrich.Schwickerath@cern.ch,Ricardo.Silva@cern.ch`

Abstract. A frequent source of concern for resource providers is the efficient use of computing resources in their centers. This has a direct impact on requests for new resources. There are two different but strongly correlated aspects to be considered: while users are mostly interested in a good turn-around time for their jobs, resource providers are mostly interested in a high and efficient usage of their available resources.

Both things, the box usage and the efficiency of individual user jobs, need to be closely monitored so that the sources of the inefficiencies can be identified. At CERN, the Lemon monitoring system is used for both purposes. Examples of such sources are poorly written user code, inefficient access to mass storage systems, and dedication of resources to specific user groups. As a first step for improvements CERN has launched a project to develop a scheduler add-on that allows careful overloading of worker nodes that run idle jobs.

1. Introduction

CPU resource providers have a high interest in keeping their CPUs busy. If a large fraction of the available CPUs is idle, it is difficult to justify funding for additional resources. In general, though, the resource provider has little influence on what the user jobs do on the nodes. The most basic change resource providers can do is attempt to improve the box usage by simultaneously running more jobs on each box. However, there are limits to this approach, because on the execution nodes disk and memory space and I/O bandwidth are limited as well. These limits become specially visible with multicore machines.

Inefficient use of computing resources by user jobs can have different reasons; while some jobs or job classes are CPU intensive, other jobs need to access and process a large amount of data, and are thus limited by I/O. The resource providers have no possibility to distinguish these job classes. However, dedicated queues targeted at CPU and I/O bound jobs provided at the batch system level allow users to tell the resource providers the type of job by submitting it to the appropriate queue. Unfortunately, even this approach cannot eliminate the inefficiencies caused by jobs which must wait for some external events - for example, a typical problem which has been observed at the CERN batch farm is that users submit jobs which wait for tape recalls via the CASTOR mass storage system. Even if users attempt to pre-stage their data before submitting the jobs to the batch farm, jobs can be scheduled on the worker nodes before the data has arrived on disk, for example if there are a lot of such requests pending.

The tape recall system at CERN knows about all incoming requests. From the position of a request in the queue and the available resources it is possible to get an estimated time of arrival. This information can be used in turn by a batch scheduler plugin to pick a fitting CPU bound job and dispatch it on the worker node with the job which is doing nothing while waiting for its data.

A mechanism doing exactly this has been developed and put in place at CERN for the shared batch resources. Fig. 1 illustrates the basic idea.

2. The batch system at CERN

The batch system at CERN is an installation with almost 2000 batch worker nodes (at the time of writing). About half of these resources are public[1], shared among the different user communities at CERN. Fair share scheduling is used on these shared nodes. The scheduler plugin previously described works on this part of the cluster. *Platform Computing Corporation's Load Sharing Facility*, LSF [1], is used to manage the load on the whole farm. The batch nodes have two Intel CPUs with two or four cores each and 2 GB of memory per core. In general, there is one job slot assigned per core and experience shows that at least one out of eight simultaneously running jobs on a node can be waiting for an external event. Therefore, we assign an additional job slot per box, and let the LSF scheduler decide if it is used or not, based on the usage of the machine.

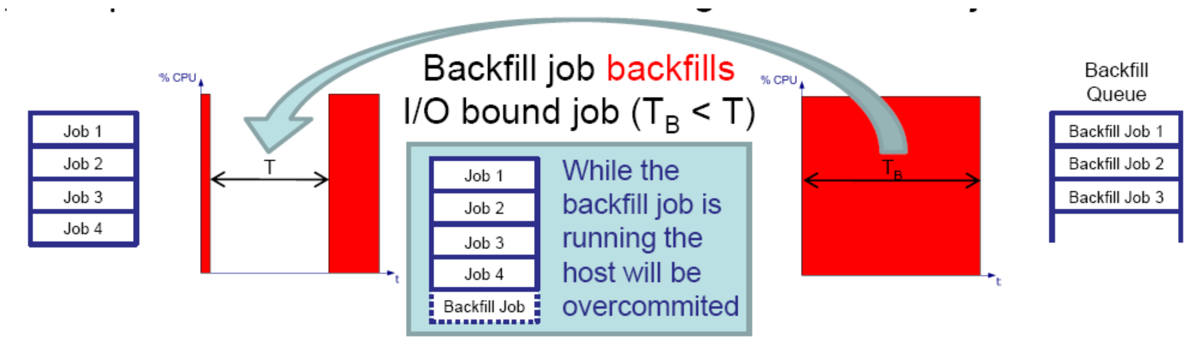

Figure 1. Job back-filling, making use of information provided by the tape system.

2.1. Batch worker node usage at CERN

The CPU usage on the public batch farm during the past year is illustrated in fig. 2. As can be

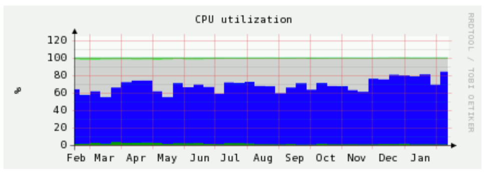

Figure 2. Average CPU usage on the public batch farm at CERN.

seen from this figure, the average CPU usage has not been optimal during the last year. The reasons for this are many-fold, as mentioned before. The aim of the project described in this document is to improve the average CPU usage per machine, any previous knowledge on the nature of the user jobs.

2.2. User job efficiencies

An obvious reason for low machine usage is low job efficiency. Fig. 3 shows the job efficiencies for all user jobs for the four LHC experiments for jobs which were run at the CERN batch

[1] *Public* means that these nodes can be used by all users who have access to the interactive Linux cluster at CERN, *lxplus*

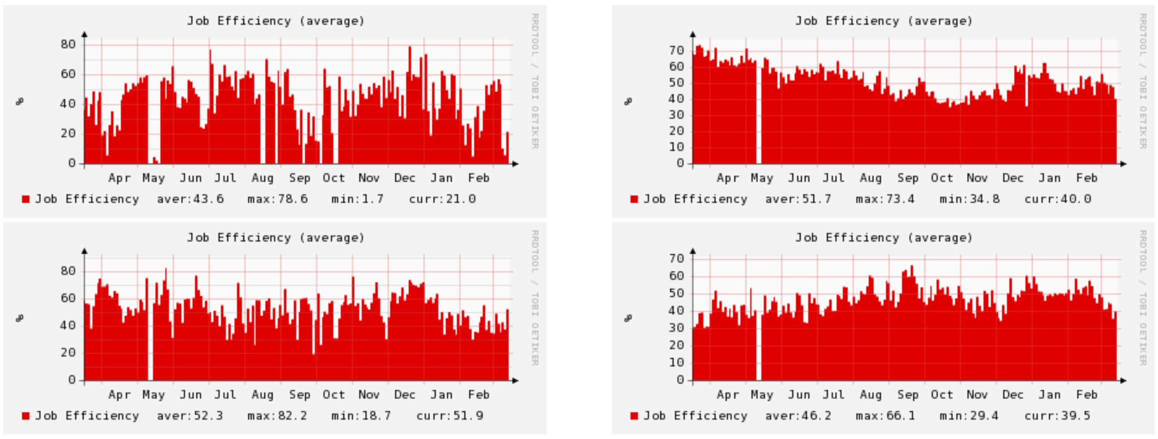

Figure 3. User job efficiencies for the four LHC communities.

farm. The CPU/runtime ratios are generally poor. Reasons for this have to be followed up by the experiments. Single users often submit very different jobs, and some accounts are shared by different parties. More information about individual jobs is needed to be able to really chase these cases effectively. This has been attempted in a different project which addresses job instrumentation, see [2].

3. Job types

User jobs can have very different efficiencies, depending on what they are doing. Here, we want to focus on the two extreme cases, purely CPU bound jobs, and purely I/O bound jobs.

3.1. CPU bound jobs and CPU queues

Within this report, a CPU bound job is defined to be a job which has a CPU/runtime ratio above 80%. This number is motivated by past experience. It is a free parameter which can be used later to tune the algorithm.

For this class of jobs dedicated queues are introduced. The queues differ in the maximum CPU time which can be used by the jobs running in it. The maximum run time is fixed by the requirement that jobs shall have an efficiency of at least 80%.

For CPU bound queues it is important to have an efficient idle job detection. The LSF product comes with a plugging which allows to raise an exception for a idle jobs. The site admin has the possibility to provide a script, which can act on such jobs.

It is not a good idea to kill such jobs off immediately, however. Experience shows that some jobs do a bit of I/O at the beginning, and then start to run very efficiently. Therefore, a check is done if the job can still archive it's goal of a total 80% CPU usage, taking into account the remaining run time, see Fig. 4. If achieving this goal is no longer possible in the remaining allowed runtime the job is automatically terminated.

3.2. I/O bound jobs

All jobs which are not CPU bound are assumed to be I/O bound. There are no special queues for them; all jobs submitted to non-CPU queues are assumed to be I/O bound. In general, all queues have a CPU and a run time limit defined, but in this case the run time limit is much larger (up to four times) than the CPU time limit. A run time limit is set for operational reasons, for example in the case when it is necessary to drain a machine.

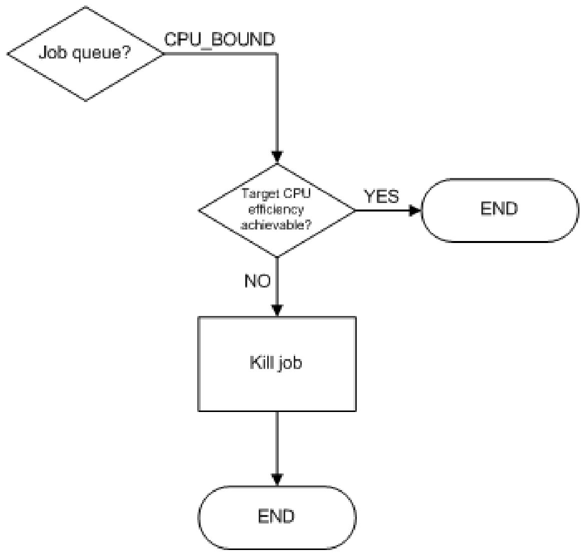

Figure 4. Flow chart indicating actions to be taken for CPU bound jobs for which an idle job detection has been raised.

4. Back-filling scheduler

The public queues for I/O bound jobs make use of the automatic idle job detection as well. An exception is raised when the CPU/run time ratio is lower than 50% for more than 30 minutes. This exception is used as an entry point for the algorithm depicted in fig. 5.

The jobs are generally not killed. Instead, the algorithm retrieves a list from the mass storage system to check if the job may be waiting for a tape recall. If this is the case, it obtains the estimated time of the arrival for this data, and examines the jobs waiting in the CPU bound queues for execution. The runtime limit for the queue is used to check if any of these jobs fits into the gap, and if the CPU bound jobs resource requirements are matched by the worker node, the algorithm dispatches the job to the worker node, in addition to the jobs which are already running there via the normal scheduling. The idea is depicted in fig. 1. To avoid overloading of boxes, only one CPU bound job can be dispatched per worker node this way at a time, even if several jobs are on a singe box have idle exceptions due to tape recalls. Also, querying the mass storage system too frequently must be strictly avoided because this could significantly degrade the system. Instead of asking the mass storage system if there is a pending request related to a specific worker node or job, the mass storage system has been configured to provide a list of all currently pending requests, together with the estimated time of arrival of the data. The list is made available via a web server, and parsed by the scheduler. No active queries are done.

In case no suitable CPU bound job is waiting in the system for execution, or in case the idle job is stalled for a different reason, no action is taken.

5. Results

Before production deployment, the scheduler has been run for some time on the production system in no-action mode. In this phase, 28% of the jobs with an idle job exception were identified by the algorithm as back-filling candidates. Parameters which influence the likelihood of finding possible back-filling candidate jobs are:

- the CPU/run time ratio above which a job is called CPU bound, (80%)
- the CPU/run time ratio limit for triggering an idle job exception, (50%)
- the time interval used for the idle time job exception, (30min).

The parameter values indicated were chosen conservatively to result in minimum impact on user jobs. We expect future tuning of the parameters to increase the likelihood of back-filling and further increase our resource CPU efficiency.

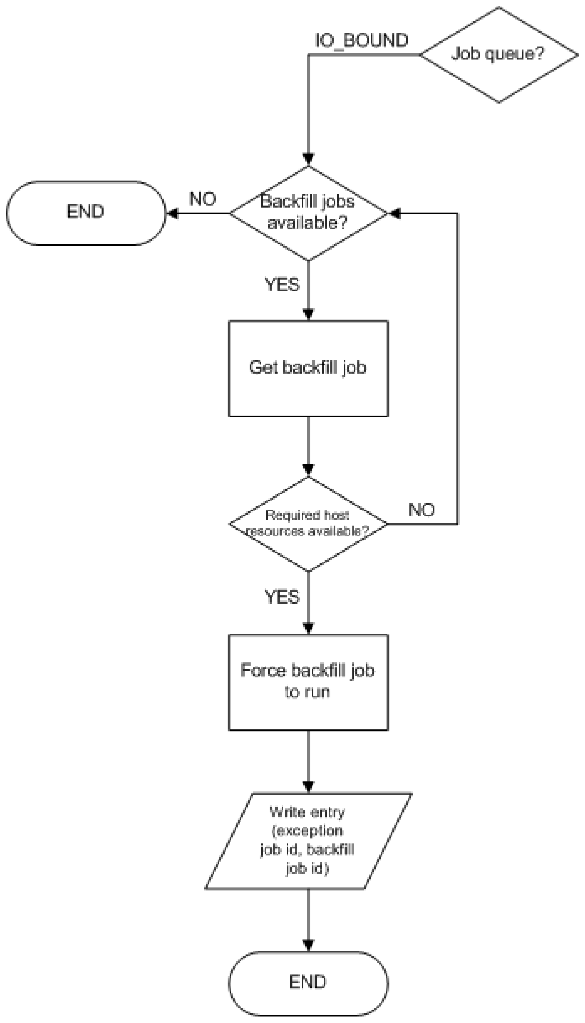
IO_BOUND
Job queue?
END
NO
Backfill jobs available?
YES
Get backfill job
Required host resources available?
NO
YES
Force backfill job to run
Write entry (exception job id, backfill job id)
END

Figure 5. Flow chart indicating actions to be taken for I/O bound jobs for which an idle job detection has been raised.

The system has been released into production in December 2008 with the above settings, and is running since then. The resulting effect is difficult to measure because of the chaotic nature of the jobs run by the users. Nevertheless, an consistent increase of the average box can be observed in the CERN monitoring figures, see fig. 2 since December 2008.

6. Summary

A mechanism has been developed and deployed at CERN which can help to increase the worker node usage by cross talking to the mass storage system, and carefully over-commiting worker nodes with idle jobs. User jobs need to be categorized into CPU and I/O bound, in order to make the algorithm work. CPU bound jobs shall be submitted into dedicated queues. Due to this algorithm and the fact that these queues have slightly higher priority, users tagging their jobs by submitting to these queues are rewarded with a smaller turn around time.

7. References

[1] Release Notes for Platform LSF Version 7.0, Nov. 2006, PLATFORM corporation.
[2] Monitoring the efficiency of user jobs, *James Casey, Daniel Rodrigues, Ulrich Schwickerath, Ricardo Silva*, Proceedings CHEP 2009, to be published