PBS Pro. What is a Scheduler?

July 10, 2020

1 Introduction

Most HPC systems are shared between many users - they are large and expensive pieces of equipment, of a size that most users will only need to use a small part of them. This means that many users will probably be using the system at any time - perhaps requesting more resources in total than the 33 000 cores that are available in Lengau. There are also a limited number of fat nodes, gpu-equipped cores and so on. How do the system administrators get to decide who should run first, and who can wait a little while?

2 Job-schedulers and computers playing Tetris

The answer is, of course, that they don't a special piece of software called the *HPC job scheduler* makes the decisions automatically. There are several different job-schedulers available. The one used by the CHPC is called PBS-Pro, developed by Altair. Other popular options are SLURM and Torque.

The task of managing the queue of all the tasks or jobs submitted by users is a very difficult one - a naive first-in, first-out approach could cause long wait times while computational resources stand idle. To illustrate this, suppose that a cluster has 10 nodes, and the following jobs are submitted (in order):

1. – 6 nodes
2. –5 nodes
3. –7 nodes
4. – 4 nodes
5. – 3 nodes
6. – 2 nodes
7. – 3 nodes

Lets assume for now that they all have equal walltime requirements. If the jobs are run strictly in order, only allowing the next job to run simultaneously if there is space for it, then the 5-node job will have to wait for the 6-node job, and after that, the 4-node job will have to wait for the 7-node job. Obviously, the best use of resources is to run the 4-node job to run at the same time as the 6-node job, then running the 5-node job, 2-node job and one 3-node job simulaneously, then finally running the 7-node job and the second 3-node job.

Of course, this is a highly simplified example. In practise, jobs vary in walltime (duration) as well, which makes the problem more complex. You might justifiably say that the job-scheduler is playing the most difficult game of Tetris ever devised!

But suppose that one HPC user submits *many* medium-sized jobs. Should this person be allowed to run job after job while all other users wait? Clearly this would also be unfair – users who have run several jobs recently should be afforded a *slightly* lower priority, until a few other users have had a turn to run jobs.

Also, some jobs may legitimately be more important than others, and should be afforded a higher priority. For instance, the CHPC provides a backup computational resource for the South African Weather Service (SAWS). If the HPC system owned by SAWS is down for maintenance, then it is a matter of national priority that their weather-forcasting simulations are run at certain times every day, even if normal CHPC users must wait an hour or two as a result.

Finally, some nodes may have to be taken offline for maintenance at times, without shutting down the whole cluster. The "game of tetris" becomes very complex indeed.

The job-scheduler is a sophisticated piece of software, but the problem is so difficult that it cannot do a perfect job, and can even be "tripped up" in certain scenarios. This may go some way to explain why the CHPC and other HPC centers place various limits on the users as to size and duration of jobs.

3 So What's in it for Me, the User?

So, what does all this have to do with reader, who probably just wants to run computations with as small a queuing time as possible?

Well, having explained in principle how the job-scheduler works, it is possible for users to legitimately "game" the system a little to favour them, or at least, reduce their waiting time in the queue.

Continuing the *tetris* analogy, the most difficult jobs to fit in are those requiring many nodes, and with a long walltime – a big block that takes up a lot of space and time, which everything else has to fit around. This is a

big part of the reason why access to the large queue is by request only, and users must prove that they can efficiently use the large amounts of resources that they are requesting.

What else can scupper the intrepid electronic tetris player? Well a smallish job with a long runtime is often surprisingly difficult to accommodate. This is why nearly all HPC centers (including CHPC) place a restriction on maximum walltime of at most a few days. Consider what happens in a tetris game when you get a long vertical block where your board is nearly filled, and you lack a slot to put it in. The best thing to do is to take advantage of parallel scaling to turn it into a job using a large number of cores, but running for only a short time (equivalent to turning the long vertical block so that it lies horisontally in tetris). In some cases, this is not possible, however.

4 Checkpoint Early, Checkpoint Often

Some jobs will not benefit from using more than one or two nodes. In these cases, you can assist the job-scheduler to cheat, by cutting the "long block" into many small short ones. Nearly all scientific software (with a few known exceptions, admittedly), allows the user to stop and save data, and restart it from where it left off. This is called *checkpointing* and is *always* a good idea, even for jobs of medium duration, because there is never a perfect guarantee that an accident may prematurely end a job or take the HPC system offline (HPC sysadmins do their best, but are only human). By checkpointing, you will ensure that you do not lose whatever progress you have made, and will be able to break a long job into several shorter ones. For long-duration computations, it is even possible to queue up a series of jobs so that once the first is completed and resources are available, the second will automatically begin, loading the checkpointed solution and continuing the solution. The third will only begin after the second has completed, again loading the last checkpoint, and so on, until the computation is complete.

5 Summary

So to finish off this brief introduction to job-sheduling, here's a quick recap: The task of allocating resources and time to running jobs is very complex, and is managed by a special piece of software known as the job scheduler. Even so, certain types of job, particularly those with a very long walltime, are difficult for the job scheduler to handle and may cause inefficient allocation of resources as well as longer queuing times for other users. For this reason, a limit is placed on allowable walltime. Users who have jobs that need a long walltime to complete are therefore advised to either use more cores to decrease the walltime, when parallel scaling permits this, or to use

checkpointing to break their jobs into smaller chunks which continue where the previous job left off to complete the computation.