Jobs masonry in LHCb with elastic Grid Jobs

Jobs masonry in LHCb with elastic Grid Jobs

F Stagni[1], **Ph Charpentier**[1]
[1]PH Department, CH-1211 Geneva 23 Switzerland

On behalf of LHCb Collaboration

E-mail: `federico.stagni@cern.ch`

Abstract. In any distributed computing infrastructure, a job is normally forbidden to run for an indefinite amount of time. This limitation is implemented using different technologies, the most common one being the CPU time limit implemented by batch queues. It is therefore important to have a good estimate of how much CPU work a job will require: otherwise, it might be killed by the batch system, or by whatever system is controlling the jobs' execution. In many modern interwares, the jobs are actually executed by pilot jobs, that can use the whole available time in running multiple consecutive jobs. If at some point the available time in a pilot is too short for the execution of any job, it should be released, while it could have been used efficiently by a shorter job. Within LHCbDIRAC, the LHCb extension of the DIRAC interware, we developed a simple way to fully exploit computing capabilities available to a pilot, even for resources with limited time capabilities, by adding elasticity to production MonteCarlo (MC) simulation jobs. With our approach, independently of the time available, LHCbDIRAC will always have the possibility to execute a MC job, whose length will be adapted to the available amount of time: therefore the same job, running on different computing resources with different time limits, will produce different amounts of events. The decision on the number of events to be produced is made just in time at the start of the job, when the capabilities of the resource are known. In order to know how many events a MC job will be instructed to produce, LHCbDIRAC simply requires three values: the CPU-work per event for that type of job, the power of the machine it is running on, and the time left for the job before being killed. Knowing these values, we can estimate the number of events the job will be able to simulate with the available CPU time. This paper will demonstrate that, using this simple but effective solution, LHCb manages to make a more efficient use of the available resources, and that it can easily use new types of resources. An example is represented by resources provided by batch queues, where low-priority MC jobs can be used as "masonry" jobs in multi-jobs pilots. A second example is represented by opportunistic resources with limited available time.

1. Introduction

HEP experiments need a large, unprecedented amount of computing and storage resources. This is the "raison d'être" of distributed computing, often referred to as grid computing resources. But grid resources are not infinite, and in the grid world, providers pose limits to their usage. Within this paper, we show a simple but effective way to exploit the time limitations of grid resources. The paper will demonstrate that, using this simple but effective solution, LHCb manages to make a more efficient use of the available resources, and that can make easy use of new types of resources. Section 2 explains the jobs masonry problem, section 3 explains how LHCb managed to have elastic simulation jobs, and its effects are shown in section 4. Conclusions are given in section 5.

2. Grid jobs and the jobs masonry problem

Users of the HEP community have been using for long the term "Grid Job", or simply "Job", as a way to indicate a computation unit. A Job can be, for example, the reconstruction of particle tracks from what it is usually dubbed as a "raw" file: a reconstruction job is, in fact, taking one or more files as input, and producing an output of one or more files, in a certain amount of time. Developers of the software used for reconstruction jobs have to pay particular attention to how long it takes to reconstruct particles. Another example of a Grid Job is represented by MonteCarlo (from now on: MC) simulation jobs, which simulates collisions inside a detector. We believe that a simple definition for a Job could be: a computation unit, having a start time and an end time. Time is important. It is important, first and foremost, because computing resources are not infinite. It is also important because, often, computing resources are shared: for example the providers, that are often indicated as "sites", share the same hardware for a number of different purposes. Thus, a single job is normally forbidden to run for an indefinite amount of time.

This concept is persisted via different technologies, the most known one being time limits on batch queues. Different types of resources have different time limits. When Grid jobs run on a Grid computing resource, it is important to have a good estimate of how long such a job will take before the job starts: otherwise, the job might be killed by the batch system, or by whatever system is controlling the jobs execution. When a job is killed, the computation time used is wasted and this leads to a situation in which VOs often overestimate the amount of time needed for their jobs, so that jobs are not killed. On the opposite, if a computing resource can be used for an amount of time that is too short for the execution of any job, such computing power is wasted. Or, in other words, the user (the Virtual Organization) is not making an efficient use of the available resources.

When a carpenter builds a wall, (s)he has to pay attention to the *masonry problem*, which means choosing bricks of correct lengths. When a VO wants to exploit at best their computing resources, the VO needs to choose jobs of the correct length. In other words, the VOs need to be good carpenters.

3. Elastic simulation jobs

Within LHCbDIRAC, the LHCb extension of the DIRAC [1][2] interware, we developed a simple way to exploit computing power, even for resources with limited time capabilities: we added elasticity to production MC jobs. All simulation jobs request (in the JDL) a short CPU time, which is enough to produce 25 events (we think that it makes little sense to have jobs that produce less than 25 events). MC jobs can then easily be matched, by pilot jobs, almost "everywhere". But, the actual amount of events produced will be determined only at run time. With our approach, independently of the time available, LHCbDIRAC will always have the possibility to start a MC job, whose length will be adapted to the available amount of time: the same job, matched on different computing resources, with different time limits, will produce a different amount of events. Figure 1 puts in graphic the elastic jobs concept.

In order to know how many events a MC job will be instructed to produce, LHCbDIRAC simply collects three values:

- $CPUe$, which is the estimated CPU time, in HS06·s, necessary for simulating one event.
- $CPUPower$, the power of the machine that is running the job, in HS06.
- $CPUTime$, which is the time left, in real time seconds, for the job before being killed.

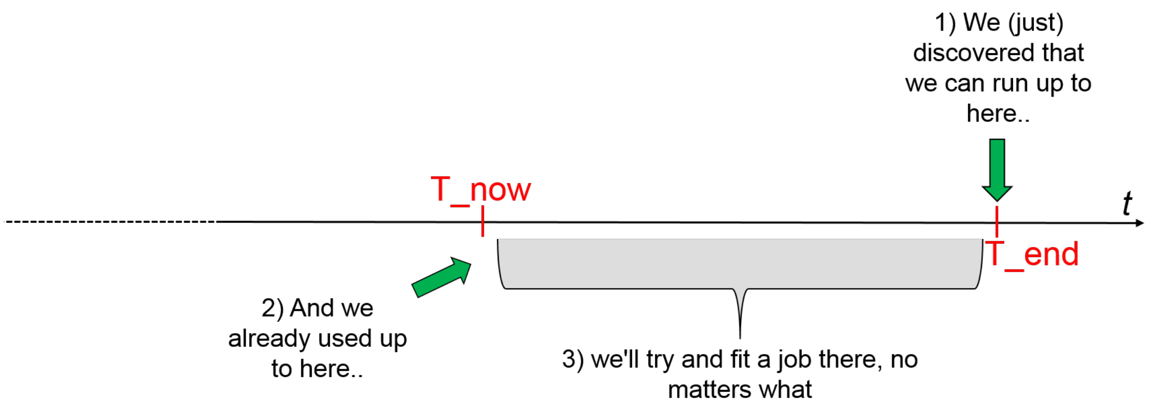
1) We (just) discovered that we can run up to here..
T_now
t
T_end
2) And we already used up to here..
3) we'll try and fit a job there, no matters what

Figure 1. Elastic jobs will use all the available CPU, no matters what

Knowing these values, we can estimate the number of events the job is able to produce with this simple formula:

$$eventsToProduce = \frac{CPUTime * CPUPower}{CPUe} * 0.8 \tag{1}$$

where the constant 0.8 is used as safety margin to cope with uncertainties in CPU power estimations.

3.1. Knowing CPUe
Simulation productions go through a testing phase before being submitted. During this test phase, a limited amount of non-elastic simulation jobs are created and submitted to a site chosen for testing. Each of this job will produce a fixed amount of events. The testing phase of a simulation production is necessary for several reasons, but for the scope of this paper, we will consider only deriving CPUe. Productions undergoing a testing phase are monitored by a dedicated agent. When all the jobs are finished, an evaluation takes place: if all jobs failed, the production request is rejected, but if jobs are successful, we evaluate CPUe. At this point, the CPUe is added and the destination of the jobs is changed. From that moment on, all the jobs submitted will be elastic jobs.

3.2. Knowing the CPU power
Knowing the power of the machine is a highly discussed subject, and it is not the goal of this paper to go in details about it. We can just remind that the power of a machine is not fully deterministic: just to give few examples, for a same machine, technologies like HyperThreading can be enabled or disabled, there might be power saving settings, and also BIOS settings can influence the effective power. Virtualization is another influencing factor.

Within LHCb (DIRAC) we go for the simplest solution: each and every pilot job runs a quick benchmark script, that has been proven to be rather reliable. We don't use BDII information because, first of all, not all LHCb computing resources are WLCG resources, and second because Worker Nodes (WNs) inside the same site often have different configurations. When available, we would like to use MachineFeatures, and soon we will for those sites that provide it already.

3.3. Knowing the CPU time left
The CPUtime left is the CPU time, in seconds, that the provider (e.g. the batch system) allows the job to run before killing it. Within LHCbDIRAC, we can collect the CPUtime by either

interrogating the batch system, if the batch system is supported, or by looking into JobFeatures, if JobFeatures is available. These checks are done within the Job Agent (inside the pilot) before matching any job. The use of JobFeatures is, for now, restricted to only Virtual Machines, and it is not yet completely integrated in the pilot.

4. Practical effects of elastic simulation jobs

Simulation productions have to produce at least a requested amount of events, so in case not enough events have been produced, simulation productions are automatically extended.

On average, LHCb simulation jobs are now shorter than non-elastic jobs.

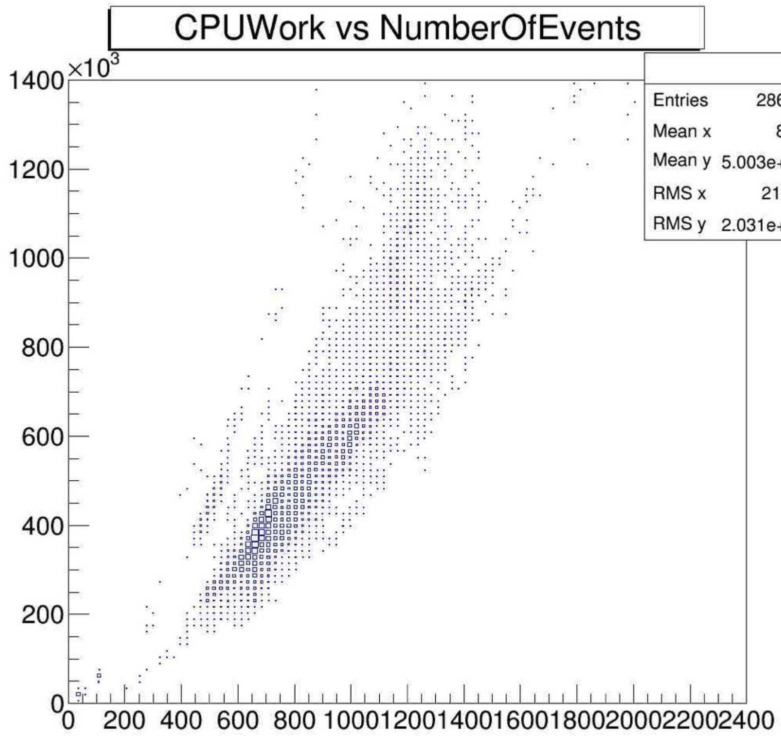

Figure 2. $CPUWork$ vs number of events produced by each simulation job, for a single production. The CPU work (which is the result of $CPUPower$ multiplied per $CPUTime$) is on the Y-axis, and the number of events is on the X-axis

Figure 2 shows a quick analysis of the jobs of one simulation production, consisting of about thirty thousands jobs. As can be seen, jobs were producing different number of events depending on the power of the machine and the CPU time left. The detailed structure of the dependency is under investigation as it is set a linear relation.

Figure 3 instead shows the machine power as calculated by the pilots compared to the power calculated by the simulation jobs. As can be seen, there are singularities that can be attributed to specific sites probably with different settings.

5. Summary and prospects

Starting from April 2014, all LHCb simulation jobs have been elastic. We believe that elastic jobs are a simple concept, quite easy to put in practice, that has rather important implications. The main implication is that simulations jobs can run almost everywhere: there are very little

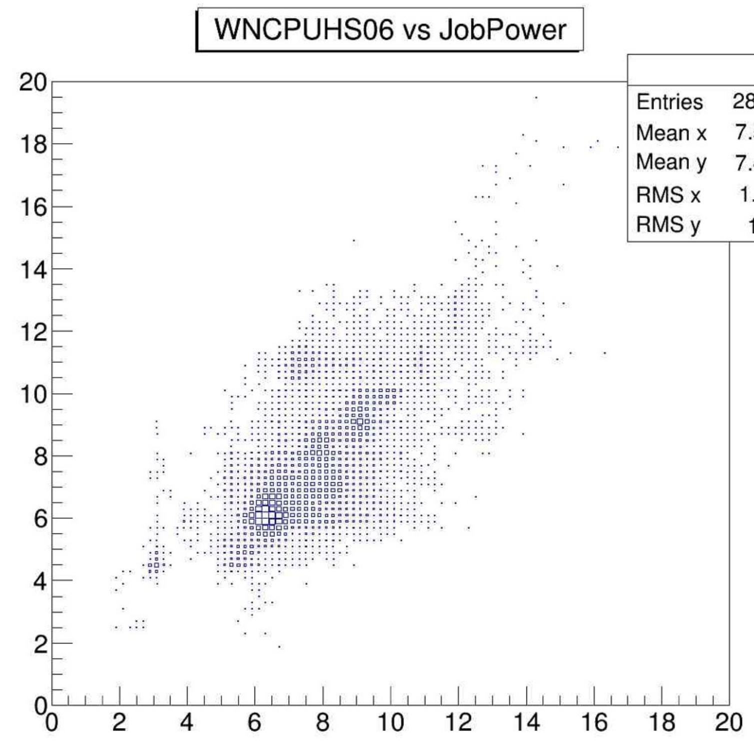

Figure 3. The machine power as calculated by the pilots (on the Y-axis), compared to the power calculated by the simulation jobs (on the X-axis), for a single production

resources, in fact, that won't have enough CPU available for running them, even for producing only a handful of events (an example is represented by volunteer computing resources). Due to the fact that simulation jobs are low priorities jobs (lower than any other type of jobs), they can be used as back-filling jobs when no other jobs can be matched. There are few more steps that we want to implement: the first is a better control on the job termination. We can in fact start a simulation job without specifying the number of events to simulate, and send a signal to the job when it should stop generating events. In order to do that, we need a tighter integration between LHCbDIRAC and the LHCb framework (Gaudi). We can send signals to the simulation application to stop producing events (i.e. after completing the currently produced event). For precise calculations we needs anyway to know CPUe. We may also consider run-time adjustments of CPUe.

References

[1] Tsaregorodtsev A and the Dirac Project 2014 *Journal of Physics: Conference Series* **513** 032096 URL `http://stacks.iop.org/1742-6596/513/i=3/a=032096`

[2] Casajus A, Ciba K, Fernandez V, Graciani R, Hamar V, Mendez V, Poss S, Sapunov M, Stagni F, Tsaregorodtsev A and Ubeda M 2012 *Journal of Physics: Conference Series* **396** 032107 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032107`