Journal of Physics: Conference Series **513** (2014) 032074 doi:10.1088/1742-6596/513/3/032074

CMS multicore scheduling strategy

Antonio Pérez-Calero Yzquierdo[1,2], Jose Hernández[2], Burt Holzman[3], Krista Majewski[3] and Alison McCrea[4] for the CMS Collaboration.

[1] Port d'Informació Científica (PIC), Universitat Autónoma de Barcelona, Bellaterra (Barcelona), Spain.
[2] Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas, CIEMAT, Madrid, Spain.
[3] Fermi National Accelerator Laboratory, USA.
[4] University of California San Diego, USA.

E-mail: aperez@pic.es

Abstract. In the next years, processor architectures based on much larger numbers of cores will be most likely the model to continue "Moore's Law" style throughput gains. This not only results in many more jobs in parallel running the LHC Run 1 era monolithic applications, but also the memory requirements of these processes push the workernode architectures to the limit. One solution is parallelizing the application itself, through forking and memory sharing or through threaded frameworks. CMS is following all of these approaches and has a comprehensive strategy to schedule multicore jobs on the GRID based on the glideinWMS submission infrastructure. The main component of the scheduling strategy, a pilot-based model with dynamic partitioning of resources that allows the transition to multicore or whole-node scheduling without disallowing the use of single-core jobs, is described. This contribution also presents the experiences made with the proposed multicore scheduling schema and gives an outlook of further developments working towards the restart of the LHC in 2015.

1. Introduction: going multicore

The motivation for the development of multicore software by the CMS collaboration is twofold. Firstly, it is inspired by the evolution of computer hardware, as over the last decade single-core CPU capabilities have been pushed to its limitations. The evolution of processor architecture has been such that in order to improve the overall CPU performance, its design has moved to adding more processor units to the CPU (*cores*), while the individual core performance has not increased significantly.

Secondly, the future evolution of LHC running conditions is to be considered. Higher energy and luminosity during LHC Run 2 implies that increasing data volumes will need to be processed by the experiments. Additionally, increasing event complexity, due to a higher pileup, will result in higher processing time per event and more intensive memory usage, as compared to the conditions during LHC Run 1.

Multicore applications present several advantages in this environment, as they aim at fully exploiting multicore CPU capabilities adapting the code to new architecture designs. Processing events in a threaded way also reduces memory consumption per core, which avoids running into memory limitations, as memory is shared between threads. Further advantages come from the experiment's Workload Management System (WMS), as the number of jobs to be handled by the

system is reduced and output files of larger size are produced, which then require less managing and merging operations.

Adapting LHC experiments computing to use multicore CPUs requires software modifications at the level of the application itself but also in the scheduling tools, both grid-wide and at the site level. These software modifications represent a transition to a new era for High Energy Physics computing, from sequential programming to parallel processing. The integration of elements from two domains that have evolved separately, Grid Computing and High Performance Computing, constitutes in a way a new paradigm of *distributed parallel computing* (see, for instance, [1]). This software evolution will happen in parallel to the upgrades of the LHC and the upgrade of the detectors, all needed to continue pushing the boundaries of High Energy Physics in the coming years.

This document is organized as follows: section 2 describes the strategy being developed for CMS multicore application and job scheduling. The approach to simultaneous scheduling of single core and multicore jobs has been tested in a controlled environment. The results of this experience are summarized in section 3. Finally, section 4 presents the conclusions from this study and the outlook for future developments towards the implementation by CMS of an efficient multicore software solution by the restart of the LHC in 2015.

2. CMS strategy for multicore jobs

The CMS collaboration is exploring ideas for new data processing software at different levels of parallelization. In a first approach, processing of different events can be simultaneously run in parallel processes. In a deeper level, data modules inside an event can also be processed in parallel. Finally, both ideas may be combined, as parallel processing of data modules, not necessarily from the same event, may be performed simultaneously. Parallel threads share common data in memory, such as detector geometry, conditions data, etc. Even parallelization at event level implemented as forked subprocesses has been shown to provide promising results, with a remarkable reduction in memory usage (25-40%), for a small CPU inefficiency, mainly due to merging of the output files [2]. Further details of each approach, as well as the development status of these efforts can be found in [3]. A first version of a multithreaded application to be used by CMS for production-size tests is expected to be ready by the end of 2013.

Multicore jobs will be intensively used in the near future, thus CMS needs to develop scheduling strategies to handle them. However, single core programs will likely still be used within CMS computing, mainly for analysis jobs. Therefore, a successful strategy must include the integration of both multicore and single core jobs scheduling. Scheduling strategies must also aim at minimizing idle CPU time, reducing any inefficiencies introduced by the scheduling infrastructure. An intensively automated scheduling system would require minimal human intervention to manage computing work-flows, hence reducing the need for dedicated manpower. Job scheduling must also provide tools for status monitoring to be used by system operators. Finally, it must allow proper accounting of used resources and the implementation of priority policies for different types of jobs, privileged user groups, etc.

A successful scheduling strategy must also take into account the grid sites' preferences. Many of the data centers supporting CMS computing share their resources with other Virtual Organizations (VOs). CMS scheduling system should not force splitting sites' resources by imposing requirements such as dedicated queues or whole-node batch slots, as they would introduce inefficiency and additional complexity in resources configuration and management.

The CMS workload management infrastructure is currently built on glideinWMS [4, 5], a grid-wide batch system, derived from HTCondor [6]. The key concept of this system is the use of the so called *pilot jobs* to schedule user jobs in a pull mode. Pilots jobs are sent to all grid sites matching job resources request, entering the queues at the local batch systems. If resources are allocated at one or several grid sites, the set of running pilots define a virtual pool of computing

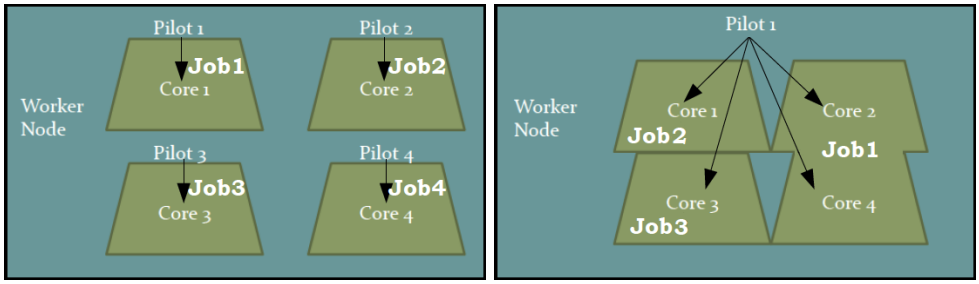

Figure 1. Current CMS WMS based on single core pilots (left) and the proposed multicore pilot strategy (right). See text for detailed description.

resources. The grid-wide WMS then manages user jobs, assigning them to the first pilot that makes them run. Further details of the use by CMS of such a model for computing workload management can be found in [7, 8].

The current scheduling model and its proposed evolution are shown in Fig.1. Workflows are currently managed by pilots which request a single slot in the local batch systems, corresponding to a single core in a workernode (WN). These pilots then pull single core jobs, allocating one user job per WN core (Fig.1, left). This model does not make any use of the multicore capabilities of the hardware where it is made to run.

In contrast, multicore pilots are designed to control several batch slots. Previous tests for the allocation of multicore jobs used pilots taking dedicated whole-node slots [2]. This system performed correctly but, as previously discussed, is not the model preferred by sites as it forces them to split their resources into specialized queues. The new schema is based on multi-slot pilot jobs featuring a dynamic partitioning of the allocated resources. Pilots take N slots from the local batch system and then arrange M internal slots according to user job's requirements. In the example shown in Fig.1, right, a single pilot managing 4 cores in a WN has split its resources in such way that it can pull a job requesting 2 cores plus 2 additional single core jobs. This model thus satisfies the condition of being able to manage single core and multicore jobs simultaneously. Multicore pilots are required to run multicore applications, but even its use to schedule single core jobs is advantageous, as the number of pilots required to manage the whole experiment workload can be greatly reduced.

The use of multicore pilots is supported by batch system configurations commonly found in CMS grid sites, assigning one batch slot to one WN core.

3. Testing the proposed scheduling strategy

The scheduling strategy described in the previous section needs to be implemented and tested. For that purpose, a testbed infrastructure equivalent to that used for CMS production jobs has been setup at CERN, based on glideinWMS 2.7 and HTCondor 7.8.6. Jobs have been sent to remote resources located at PIC, the Spanish Tier-1 grid site supporting CMS, ATLAS and LHCb experiments. Local components include CREAM Computing Element as grid middleware, Torque/Maui as local batch system and scheduler, as well as two 8-core WNs running Scientific Linux 6 and configured as 8 batch slots per machine, i.e. one slot per CPU core, for a total of 16 job slots.

The objective of the experience is to verify the technical capabilities of such a setup with respect to handling single and multicore jobs simultaneously and to identify potential inefficiencies in CPU usage intrinsic to the job allocation machinery itself, not the running application. For this reason, jobs running an application 100% efficient in CPU usage were

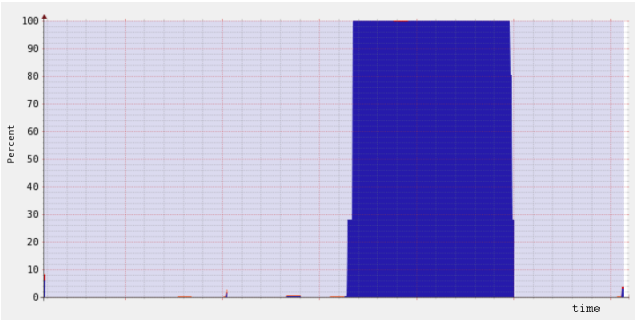

Figure 2. CPU usage (%) versus time in an example of the stress command loading an 8-core CPU up to 100% for 1000 s.

```
Job ID           Queue      NDS    TSK   S Time
-------------------------------------------------
23744131.pbs02.p cms_mcor     1      4    R 63:38   td457+td457+td457+td457
23744132.pbs02.p cms_mcor     1      4    R 32:21   td458+td458+td458+td458
23744133.pbs02.p cms_mcor     1      4    R 31:45   td457+td457+td457+td457
23744134.pbs02.p cms_mcor     1      4    R 31:12   td458+td458+td458+td458
23744135.pbs02.p cms_mcor     1      4    Q   --     --
23744136.pbs02.p cms_mcor     1      4    Q   --     --
23744137.pbs02.p cms_mcor     1      4    Q   --     --
23744138.pbs02.p cms_mcor     1      4    Q   --     --
```

Figure 3. Pilot jobs taking multiple slots from the local batch system. Four running pilots jobs have been assigned 4 batch slots each.

employed. These jobs executed the *stress* command [9], causing CPU loads by means of performing square root calculations of random numbers for a given time interval. As an example, the command

```
stress --cpu 8 --timeout 1000s
```

generates 8 stress threads saturating the CPU of one of the 8-core WNs in the test environment for 1000 seconds, see Fig.2. In order to test the scheduling strategy via multicore pilots, test jobs were set to require 1, 2 and 4 CPU cores, executing stress threads to load the corresponding number of cores to 100% usage.

It is not in the scope of this study to reproduce a realistic situation of a typical grid site, with a much larger pool of resources being accessed simultaneously by multiple VOs, each of them sending jobs with different requirements. Such complexity was simulated in a study by ATLAS [10], which describes different dynamic regimes arising from diverse local scheduler configurations when they are required to accept both single and multicore jobs. This study, however, does not contemplate our proposal of encapsulating single core jobs into multicore pilots. This key feature of the CMS model could simplify job scheduling by sending jobs in a unified way to the local WMS. Its impact on such a realistic scenario is nevertheless still to be quantified.

Multicore pilots have been tested in 2, 4 and 8 slot configurations. From the point of view of the local batch system, pilots are just jobs requesting a given number of slots. Figure 3 shows an example of pilots requesting 4 slots. Each running pilot gets assigned 4 slots on the same machine, the remaining pilots then wait in the queue.

As the job queue gets populated with jobs of diverse resources requirements, it is essential

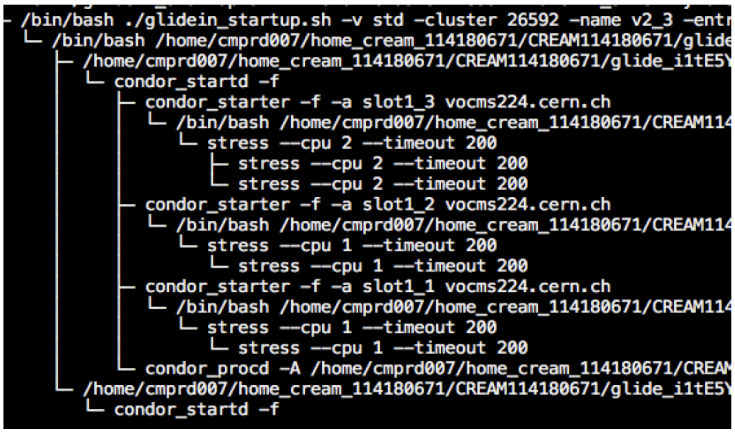

Figure 4. Example of a process structure for a pilot managing multiple jobs of different requirements at the same time.

that pilots can pull different types of jobs and run them simultaneously. The tree of processes in one of the WNs running the test jobs reveals the structure under the control of the pilot process. In the example shown in Fig.4, a 4-core pilot has arranged itself into 3 internal uneven slots, pulling then one 2-core plus 2 single core jobs.

In a first CPU usage efficiency test, see Fig.5, 8-core pilots were employed. Job queue was filled with a random mixture of 1, 2 and 4 core jobs. Pilots configured themselves internally according to queue content at their startup. Each pilot arranged initially into 8 1-core slots and started pulling single core jobs. Pilots ran single core jobs efficiently until none remained in the queue. Inefficiencies at startup and at job completion were observed (Fig.5(a)). Once single core jobs were exhausted, 2-core jobs were next to be run. Pilots were unable to pull 2-core jobs in the initial configuration, however, internal slots in a pilot may claim idle resources, as jobs now require more cores in order to run. Slots were thus rearranged from 8 1-core to 4 2-core slots. Some CPU inefficiency was observed during core reallocation to internal slots, as well as at job completion (Fig.5(b)). Once 2-core jobs were finished, 4-core jobs were next to be run, thus slots needed to be rearranged once more, from 4 2-core to 2 4-core slots. With the internal slots reconfigured, 4-core jobs started running and continued until queue was empty. Inefficiencies during core reallocation to internal slots (once per slot) and at job completion were detected (Fig.5(c)). Finally, pilots remained in the system for some time, waiting for more possible jobs to enter the queue.

For the second CPU efficiency test, shown in Fig.6, 8-core pilots were again used. However, this time the queue was initially filled only with 4-core jobs. The queue content caused each pilot to configure itself as 2 4-cores slots. These 4-core jobs were executed, running efficiently until none remained in the queue. Some inefficiency caused by the pilot draining was observed (see Fig.6(a)), as the last 4-core job was running while the other slot was already empty.

Once pilots were running, single core jobs were added to the queue, being pulled only after 4-core jobs were exhausted. Initially, as the optimal configuration needed to run 4-core jobs, each pilot had arranged itself internally into 2 4-core slots. In the current stage of configurable-slot pilot development, as shown in the previous test, internal slots can absorb idle resources from free slots. However, they are not able to subdivide themselves at running time. This meant that each 4-core slot could only pull a single core job instead of four. This caused CPU usage to drop, as Fig.6(b) clearly shows. Again, CPU inefficiency derived from pilot draining can be observed.

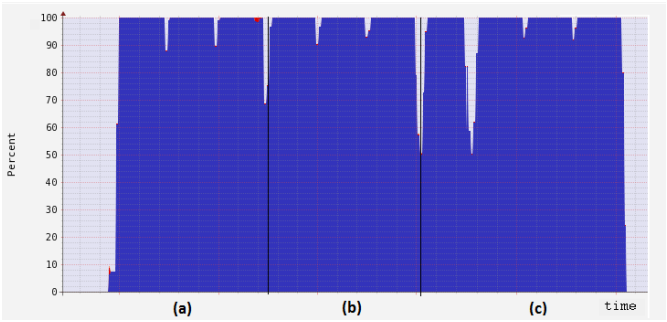

Figure 5. CPU usage (%) versus time for the first scheduling efficiency test. See text for a detailed description.

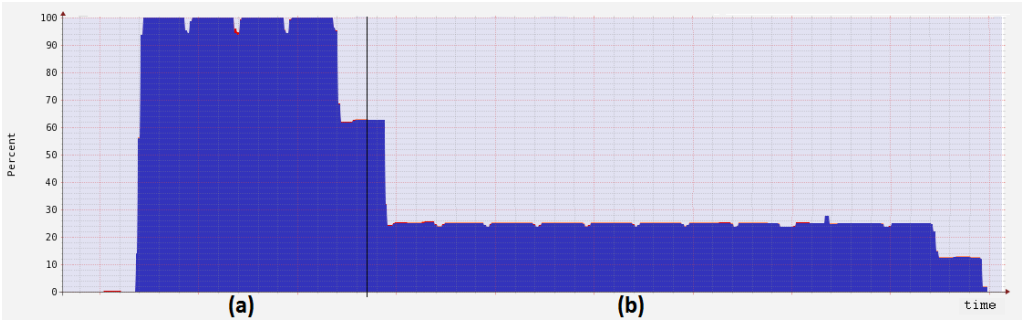

Figure 6. CPU usage (%) versus time for second scheduling efficiency test. See text for a detailed description.

4. Conclusions and timeline for CMS multicore software deployment

As described in the previous sections, CMS multicore scheduling strategy is currently under development, based on the idea of multicore pilots with dynamic allocation of internal slots. The principle has been successfully tested, as multicore pilots have been shown to be able to handle a mixture of single core and multicore jobs. However, several sources of scheduling inefficiencies have been identified in these tests:

- Not specific to multicore pilots: at the beginning of pilot lifetime and at job completion once a job is finished, until another job is pulled and starts to run. At the end of pilot lifetime, when jobs are all finished but pilots remain in the batch system waiting for more potential jobs to appear.
- Exclusive to multicore pilots: during the internal slot reconfiguration to incorporate idle resources once the jobs that caused the initial configuration are exhausted from the queue. Draining inefficiency while finishing long jobs, as pilots are using only a fraction of the allocated cores.

Additionally, at this stage of software development, pilot internal slots can't yet be subdivided into smaller pieces at running time, which is needed in order to adapt pilot internal configuration to the current queue content. Inefficiency thus arises as jobs are assigned more resources than they require. Optimal allocation of resources to internal slots must be achieved dynamically, not only at start-up time. Pilots must rearrange themselves accordingly or be forced to exit the batch system, so that fresh pilots may start with a new optimal configuration.

There is room for improvement in the presented job scheduling strategy regarding CPU efficiency, both in terms of fine tuning and new features:

- Pilot performance can be controlled through the parameters that define pilot lifetime: the relation between job duration and total pilot lifetime should be tuned in order to minimize inefficiencies at job start and completion, draining, etc.
- Further developments and ideas are being considered: pilots could request a range of slots instead of a fixed number to increase the likelihood for multicore pilots to get the resources they demand. Also, methods for improved communication between job queue, pilots and local batch systems are being discussed in an organized WLCG taskforce [11]. Other ideas include ordering job scheduling according to their expected run time, as scheduling shorter ones at the end of pilot lifetime would increase the probability of job completion, and the possibility of pilots returning slots to the local batch system when they are not expected to be used.

The current timeline for CMS multicore strategy deployment is the following: a first version of a multi-threaded application is expected to be ready by the end of the year 2013. The new scheduling schema using multicore pilots should be implemented and tested for production jobs at large scale during 2014. It will first be used to manage production single core jobs, then production multicore parallelized jobs. The final objective is to provide CMS computing with multicore applications and a multicore job scheduling strategy ready for the LHC restart by 2015.

Acknowledgments

The Port dInformació Científica (PIC) is maintained through a collaboration between the Generalitat de Catalunya, CIEMAT, IFAE and the Universitat Autonoma de Barcelona. This work was supported partly by grants FPA2007-66152-C02-00 and FPA2010-21816-C02-00 from the Ministerio de Ciencia e Innovación, Spain. Additional support was provided by the EU 7th Framework Programme INFRA- 2007-1.2.3: e-Science Grid infrastructures Grant Agreement Number 222667, Enabling Grids for e-Science (EGEE) project and INFRA-2010-1.2.1: Distributed computing infrastructure Contract Number RI-261323 (EGI-InSPIRE).

References

[1] Elmer P *et al* 2013, The Need for an R&D and upgrade program for CMS software and computing, `http://arxiv.org/abs/1308.1247`
[2] Hernandez J *et al* 2012, Multi-core processing and scheduling performance in CMS, `http://indico.cern.ch/contributionDisplay.py?contribId=199&sessionId=4&confId=149557`
[3] Jones C *et al*, 2013, Stitched together: transitioning CMS to a hierarchical threaded framework, `http://indico.cern.ch/contributionDisplay.py?contribId=158&sessionId=3&confId=214784`
[4] Sfiligoi I *et al* 2009, The pilot way to grid resources using glideinWMS, `http://dx.doi.org/10.1109/CSIE.2009.950`
[5] GlideinWMS Homepage: `http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/doc.prd/index.html`
[6] HTCondor Homepage: `http://research.cs.wisc.edu/htcondor/`
[7] Grandi C *et al* 2013, CMS computing model evolution, `https://indico.cern.ch/contributionDisplay.py?contribId=102&sessionId=5&confId=214784`
[8] Sfiligoi I, 2013, CMS experience of running glideinWMS in high availability mode, `https://indico.cern.ch/contributionDisplay.py?contribId=114&sessionId=9&confId=214784`
[9] Stress command: `http://linux.die.net/man/1/stress`
[10] Crooks D *et al* 2012, Multi-core job submission and grid resource scheduling for ATLAS AthenaMP, J. Phys.: Conf. Ser. 396 032115 doi:10.1088/1742-6596/396/3/032115.
[11] WLCG Machine/Job Features task force, 2013, `https://twiki.cern.ch/twiki/bin/view/LCG/MachineJobFeatures`