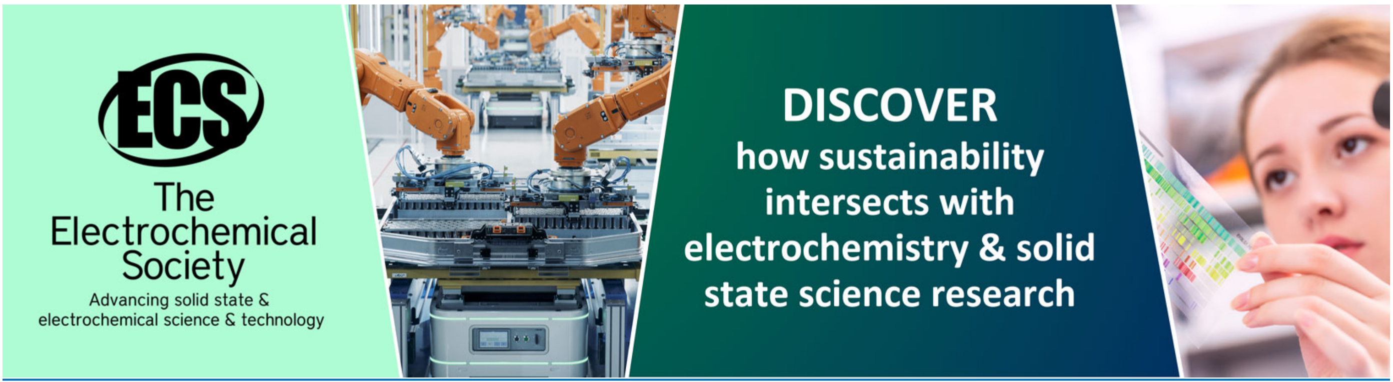
Journal of Physics: Conference Series

PAPER • OPEN ACCESS

Multicore job scheduling in the Worldwide LHC Computing Grid

To cite this article: A Forti *et al* 2015 *J. Phys.: Conf. Ser.* **664** 062016

View the article online for updates and enhancements.



Multicore job scheduling in the Worldwide LHC Computing Grid

A Forti[1], A Pérez-Calero Yzquierdo[2,3], T Hartmann[4], M Alef[4], A Lahiff[5], J Templon[6], S Dal Pra[7], M Gila[8], S Skipsey[9], C Acosta-Silva[2,10], A Filipcic[11], R Walker[12], C J Walker[13], D Traynor[13], S Gadrat[14]

[1] School of Physics and Astronomy, University of Manchester, Oxford Road, Manchester, M13 9PL, UK.
[2] Port d'Informació Científica (PIC), Universitat Autónoma de Barcelona, Barcelona, Spain.
[3] Centro de Invest. Energéticas, Medioamb. y Tecnológicas, CIEMAT, Madrid, Spain.
[4] Karlsruhe Institute of Technology, Steinbuch Centre for Computing, Karlsruhe, Germany.
[5] Rutherford Appleton Laboratory, Harwell Oxford, Didcot OX110QX, UK.
[6] National Institute for Subatomic Physics, Science Park 105, 1098 XG Amsterdam, Netherlands.
[7] INFN-CNAF, viale Berti-Pichat 6/2, 40127 Bologna, Italy.
[8] Swiss Center for Scientific Computing, ETH Zentrum, RZ, Clausiusstrasse 59, CH-8092 Zurich, CH.
[9] School of Physics and Astronomy, Kelvin Building, University of Glasgow, Glasgow, G12 8QQ, UK.
[10] Institut de Física d'Altes Energies, IFAE, Edifici Cn, Universitat Autònoma de Barcelona, Barcelona, Spain.
[11] Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia.
[12] Ludwig-Maximilians-Universitat, Munchen, Fakultat fur Physik Schellingstrasse 4, 80799 Munich, Germany.
[13] Queen Mary University of London, School of Physics and Astronomy, Mile End Road, London E1 4NS, UK.
[14] Centre de Calcul de l'IN2P3, 43 Bd du 11 Novembre 1918, F-69622 Cedex, France.

E-mail: `alessandra.forti@cern.ch, aperez@pic.es`

Abstract. After the successful first run of the LHC, data taking is scheduled to restart in Summer 2015 with experimental conditions leading to increased data volumes and event complexity. In order to process the data generated in such scenario and exploit the multicore architectures of current CPUs, the LHC experiments have developed parallelized software for data reconstruction and simulation. However, a good fraction of their computing effort is still expected to be executed as single-core tasks. Therefore, jobs with diverse resources requirements will be distributed across the Worldwide LHC Computing Grid (WLCG), making workload scheduling a complex problem in itself. In response to this challenge, the WLCG Multicore Deployment Task Force has been created in order to coordinate the joint effort from experiments and WLCG sites. The main objective is to ensure the convergence of approaches from the different LHC Virtual Organizations (VOs) to make the best use of the shared resources in order to satisfy their new computing needs, minimizing any inefficiency originated from the scheduling mechanisms, and without imposing unnecessary complexities in the way sites manage their resources. This paper describes the activities and progress of the Task Force related to the aforementioned topics, including experiences from key sites on how to best use different batch system technologies, the evolution of workload submission tools by the experiments and the knowledge gained from scale tests of the different proposed job submission strategies.

1. The WLCG multicore deployment task force

Running multicore was a long standing problem in WLCG. The problem had three aspects: the first one involved the application, as the HEP code is inherently sequential and required additional development to run on multicore slots; the second was at experiments submission level, it is easy enough to send one job requesting N cores, it is not so easy to integrate large scale submission to a distributed system; the third front was at the site level, different batch systems have different capabilities to handle multicore and sites can be shared and multicore and single core jobs have to coexist. While the first aspect was clearly in the experiments domain, the more general grid and site submission required coordination between the different parties. The WLCG multicore TF [1] was set up with the explicit mandate to find a set of easy to implement recommendations to schedule multicore without waisting resources at the WLCG sites by reviewing the two different experiments approaches and the capability of the different batch systems. The problem was of course complicated by WLCG being a distributed system and having to deal with a variety of sites different by size, batch systems, Computing Elements, number of VOs supported, on top of the differences between the two experiments philosophies.

2. The scheduling problem

Scheduling multicore jobs is more complicated than scheduling single core, which in itself is a complicated problem that may depend on fairshares, priorities and resource requirements, but that can still be reduced to when one slot is freed the job at the top of the queue occupies it. Conversely a multicore job requires an amount of resources corresponding to multiple single core slots and when it arrives at the top of the queue it has to wait for the resources it required to be freed by the batch system. This process is called *draining* (Fig. 1) and it's an expensive operation because, depending on other types of workloads running, it can take several hours, and in some cases few days, to free these slots which during all this time remain idle. Furthermore the place at top of the queue is not fixed. The batch system rearranges jobs in the queue when new jobs arrive with different priorities. If new single core jobs have a higher priority they jump in front of our multicore job and occupy the partially freed slots wasting all the idle time further and the draining process has to restart from scratch once the multicore job is back at the top. This is also negative for the multicore jobs users since their jobs take a very long time to run if they run at all. In the LHC experiments case this is made even more difficult by requesting all the slots for a job on the same node. Taking all this into account it is clear that preserving the slots as a way to reduce the draining becomes a very important part of running multicore.

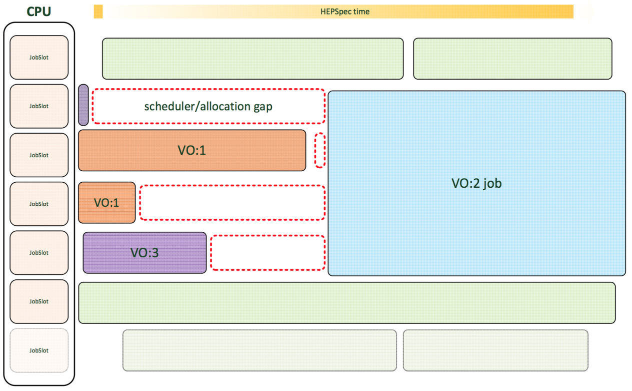


Figure 1. Resource draining required to create slots to which the multicore job can be allocated.

2.1. ATLAS and CMS models

Both ATLAS and CMS VOs rely on the concept of *pilot jobs* for resource allocation, pilots being submitted ahead of the job proper in order to reserve resources. However, each of the collaborations has proposed a particular model on how to perform the mixed scheduling of single and multicore jobs. Therefore, from the point of view of the sites, they present different job submission patterns. As Fig.2 shows, ATLAS submits single and multicore jobs in single and multicore pilots respectively [2]. Moreover, a single job (*payload*) is run for each pilot. When jobs end, pilots are terminated as well, resources being return under control of the site. In contrast, CMS model foresees to employ multicore pilots exclusively. Multiple jobs of different core counts can be executed for a single pilot, which is capable of internally readjusting its partitioning in a dynamic way [3].

The CMS model, presenting a single and unified resource request, is intended to allow the experiment more control of its distributed resources. Prioritization among different tasks is removed from the site responsibility, as is the mixing of single and multicore jobs. For ATLAS, scheduling the mix of jobs is entirely a site responsibility. However, even considering the CMS model, sites providing resources to several VOs would still need to handle a mixed load of requests, so only sites exclusive to CMS would be relieved completely from such a task.

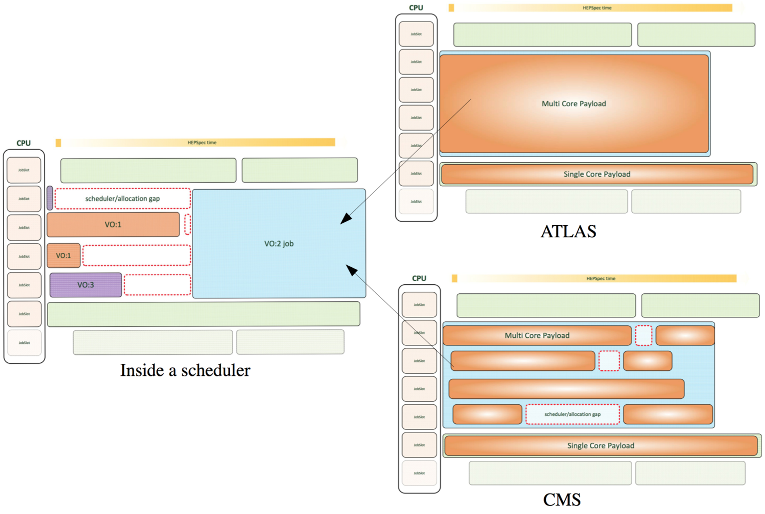


Figure 2. Scheduling models for Atlas and CMS.

In general, the CMS model, allows for good predictability of pilot running times, as the internal fluctuations in the type of workflow being executed are hidden from the site within the pilot. A potentially wide distribution of job running times is averaged by the multi-payload pilot model, sites only perceiving pilot running times. ATLAS submission model is less predictable, both in terms of alternating periods of high and low activity and also with respect to job running times. However, returning the resources after each payload is run produces a faster rotation rate in the use of CPUs, so a site has increased flexibility to satisfy the requests from all its VOs in a shorter time interval.

2.2. Initial observations

Initial observations from sites - reported also by other sites at later stages - were quite indicative of the problems to solve.

- Multicore require continuous draining of slots. Once the process of draining starts the problem is to contain the waste
- Short jobs, less than 6 hours don't exploit enough the slots that have taken so much effort to drain. On the other hand at shared sites too long jobs, longer than 24h, can be a problem because they don't allow a quick enough turn over for all the users to have a go at running. There must be a balance. At dedicated sites this is a much more muted problem.
- Much longer queuing times for multicore due to all the draining cycle. Sometimes not running for days.
- Bursty submission is unanimously considered the most problematic. Sites can come up with different strategies that will limit the waste while running constantly but the ramp down will give back the slots to single core jobs causing peaks of draining (wasted idle slots) during the subsequent ramp up.

In Fig. 3 it is shown the effect of much longer waiting times for multicore as compared to single core and the effect of too short jobs always compared with single core experienced at GridKA

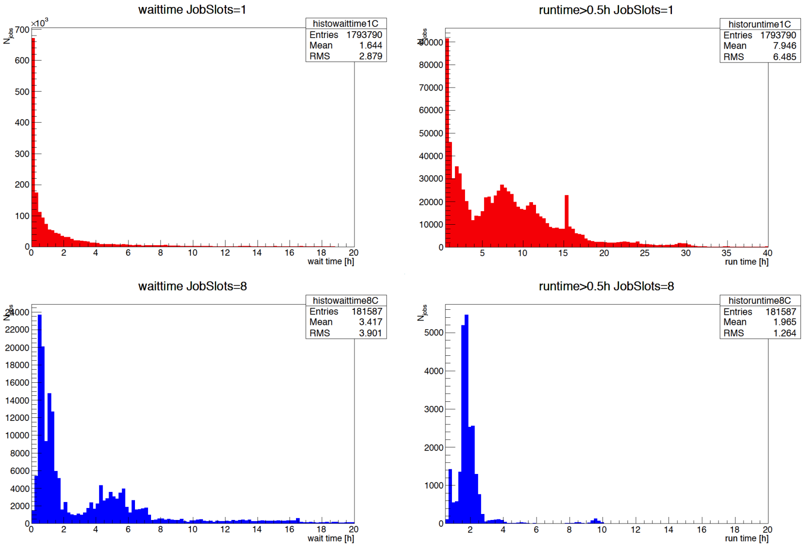


Figure 3. Initial observations of multicore scheduling at KIT: waiting time (left) and running time (right) for single core (top) and multicore (bottom) jobs.

Figure 4 shows instead RAL experience. It was noted that a constant draining of WNs, called $defragmentation$ in HTCondor [4] jargon, was needed in order to sustain a continuous load of multicore jobs. As the first picture shows, if the defragmentation procedure is disabled, the scale of running multicore jobs decreases. Uncontrolled defragmentation however produces excessive node draining, hence unused cores. The second picture shows how the amount of wasted CPU cores decreased as the first methods to control the defragmentation rate were introduced.

As we have seen, once the multicore slots are assigned the major task is to keep them occupied and not let single core take them back. The most important strategy to keep the slots alive is for experiments to maintain a continuous and stable supply of multicore jobs, this with an agreed multicore size at each site (default 8) will help the batch systems to replace multicore with multicore. The experiments should also optimize the length of jobs. As we have seen too long may cause problems at shared sites and too short don't exploit the resources enough causing extra waste.

Sites on the other hand should try to allocate slots used by multicore jobs to multicore jobs so that the batch system doesn't have to drain to replace a finished job. The decision to have the same multicore slot size at each site was taken to help with this reallocation. If they can't reallocate the slots, sites should at least try to rank/prioritize the multicore slots over the single core ones.

3. Implementations and solutions

Different strategies have been adopted by different sites, what follows are the solutions the task force recommends for the most common batch systems.

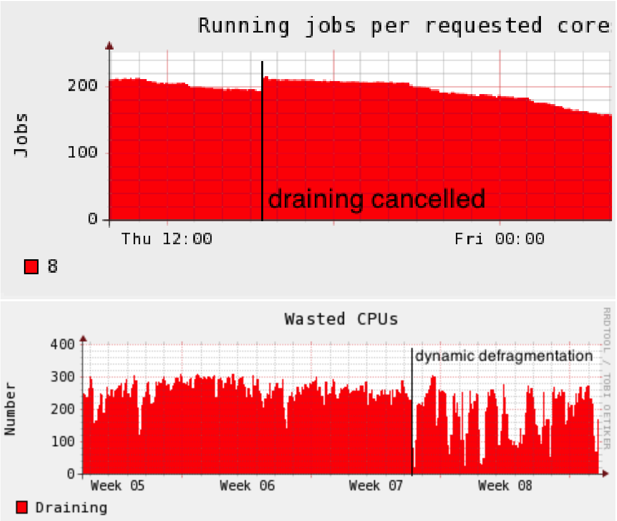


Figure 4. Initial observations of multicore scheduling at RAL: 8-core running jobs before and after the HTCondor defragmentation daemon is switched off (top) and CPU wastage due to defragmentation of the nodes before and after dynamic defragmentation rate is introduced (bottom).

3.1. Dynamic partitioning of Torque at NIKHEF and PIC and LSF at CNAF

Tier-1 sites at NIKHEF, PIC and CNAF have implemented multicore slot conservation introducing algorithms to produce a dynamic partitioning of CPU resources. NIKHEF and PIC employ the `mcfloat` tool [5] for their Torque/Maui [6] batch systems, while CNAF has implemented an analogous strategy for LSF [7, 8]. By means of separating their WNs into different pools only accessible to each type of jobs, higher priority single-core workloads are prevented from destroying the multicore slots. When multicore job pressure can't be satisfied with the current multicore pool, WNs are put to drain from single core jobs, then moved to the multicore pool. The boundary between both pools is adjusted dynamically with policies designed to avoid a) too many draining WNs, hence empty slots, at any point in time and b) empty multicore slots when the supply of such jobs consistently decreases. Figure 5 shows the observed performance in each of these three sites. Allocation of resources is successfully achieved in every case keeping CPU wastage to a negligible level.

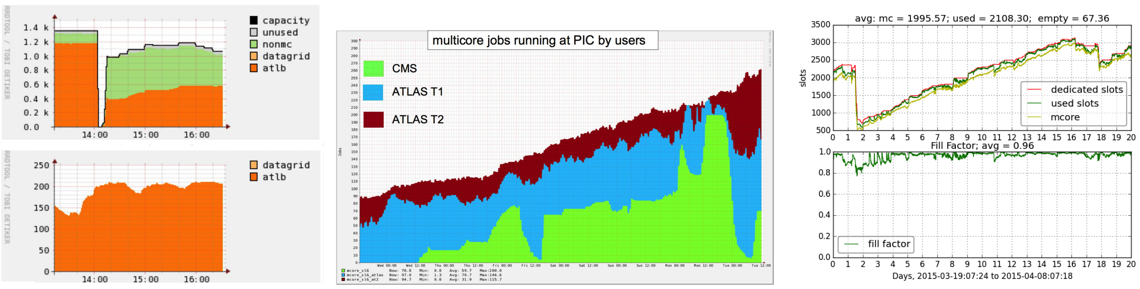


Figure 5. Dynamic partitioning of resources as a strategy to drain nodes and conserve and recycle multicore slots at NIKHEF (left), CNAF (center) and PIC (right).

3.2. HTCondor at RAL

The solution at RAL is also using native solutions with some adjustment. As we have seen one of the initial observations was that continuous draining is necessary to be able to run multicore jobs. To achieve that they have enabled the *defrag daemons*, which tells HTCondor to drain a configurable number of nodes and slots, with a cron job to adjust these numbers according to the number of multicore jobs queued with the classAd expression DEFRAG_MAX_CONCURRENT_DRAINING. While they don't do partitioning - their WN are shared by multicore and single core jobs - they used GROUP_SORT_EXPR another classAd expression to evaluate multicore jobs in the queue before single core, by doing so if a multicore job is likely to be replaced by another multicore job. They also rank WNs by how many 8-slots can be freed, i.e. a 32 slot WN will be picked in preference to a WN with only 8 slots. Figure 6 shows ATLAS and CMS both running at RAL (which is also another multi-VO site) and the peaks of draining slots corresponding to a drop in the number of queued jobs and consequent ramp up and to an increase of queued multicore. Draining waste has been confined between 0.5-2.5% with the peaks usually being narrow.

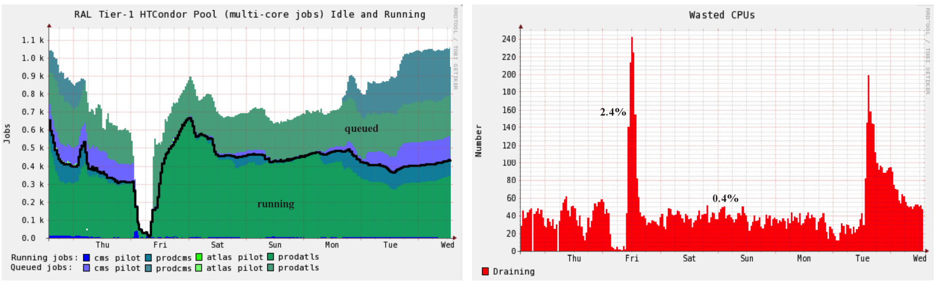


Figure 6. Multicore job scheduling with dynamic defragmentation for HTCondor at RAL: running and queued multicore jobs (left) and percentage of wasted CPU (right).

3.3. Dynamic scheduling SGE at GRIDKA

While other sites have been willing to complement their batch systems with their own scripts the goal at GridKA was to use only SGE native features to minimize the amount of resources wasted with draining and let the experiments decide how they should use their fairshare ranking and prioritizing their workload in their framework. They configured a SGE Parallel Environment for the jobs to request the appropriate amount of resources and capped the number of slots that can be drained at the time by using *maxreservation*. Their worker nodes are not partitioned and can run single-core and multicore. With this setup they achieved a degradation of less than 2%. Figure 7 shows both ATLAS and CMS running at GridKA. They are a good show case of how it is not possible to ignore single core jobs from other VOs at shared sites.

4. Backfilling, memory and passing parameters to the batch systems

Backfilling is a common technique native to many batch systems: jobs of lower priority are allowed to utilize the reserved resources only if their prospective job end (i.e. the declared wallclock usage) is before the start of the reservation. Figure 8 shows how this works. Successful backfilling relies on two concepts:

- Entropy: there should be a distribution of jobs resources requests in order to increase the likelihood of finding the right "piece" to fill each temporary hole in draining WNs.
- Predictability: job running times estimates, so that the scheduler can make a decision on whether it should run this job in that hole or not.

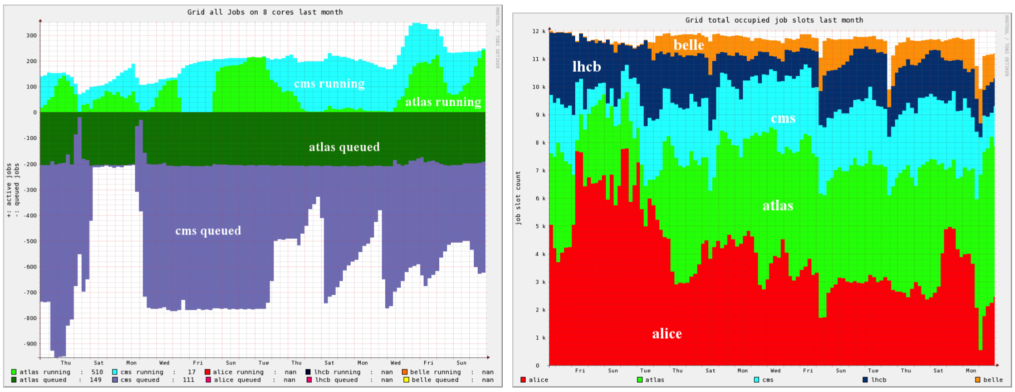


Figure 7. Multicore job scheduling with dynamic reservation for SGE at GRIDKA: running and queued multicore jobs (left) and example of farm utilization for all VOs (right) for a period of a month.

Backfilling requires the jobs ability to tell at submission time the estimate of the walltime to the batch system. This hasn't been achieved for many reasons: a) inherent the job themselves, the same application can take shorter or longer depending on the type of event it is processing; b) the variety of job resources a job can access in a distributed system and within a site; c) the late binding phylosophy of the experiment frameworks; d) there is no standard set of parameters to pass to different batch systems and the CEs behaviour is also not standard.

The task force has taken on board the work to do to resolve c) and d) thus extending it's original mandate including also the handling of high memory jobs. This has required reviewing the whole chain for memory and time parameters used by the jobs and review of the meaning of memory and time parametrs in the whole chain from the experiment framework, to the CEs, to the batch systems, to the kernel which ultimately determines what can or cannot be done with them.

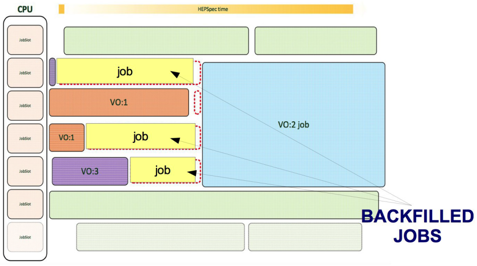


Figure 8. Backfilling is used to minimize the wastage of resources while draining them to allocate multicore jobs.

5. Accounting

The main problem with accounting was that batch systems register the amount of CPU time used by all the slots requested by the multicore job but the walltime as the realtime used by the job. While this is the correct thing to do, the way WLCG calculates the efficiency for the jobs is dividing the job CPUtime by the walltime and this resulted in efficiencies being off by a factor 8. To fix this required modifying the way APEL (WLCG accounting system) clients collect the information from the batch systems and the accounting portal to report the new information being correctly displayed. To update the clients and their configuration became part of the multicore deployment campaign and was done in cooperation with EGI, who is responsible for the APEL system. The new portal to display the information correctly is still in development.

6. Status of deployment to sites

At the moment of writing, the two experiments have different targets of deployment. CMS priority for 2015 is multicore prompt data reconstruction [10] which requires T0 plus 50% of

T1 CPU resources and the deployment target has been achieved [3]. ATLAS has 95% of the resources enabled. Figure 9 shows the increasing fraction of resources being used by ATLAS multicore jobs over the past few months progressing towards the target of 80% of its production at all sites.

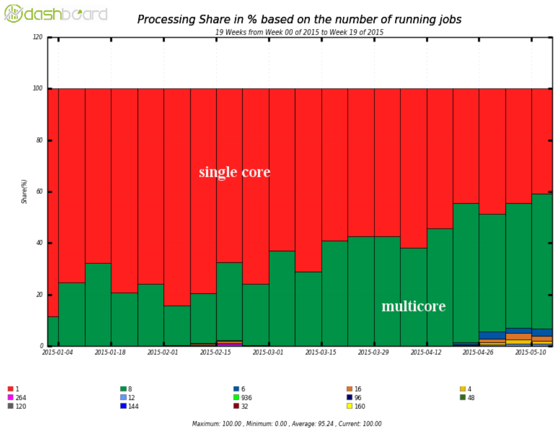


Figure 9. Fraction of CPU resources used by ATLAS as a function of the job core-count.

7. Conclusions and outlook

Over the past year we have reviewed the experiements models to run multicore, the batch systems capabilities, and found a set of strategies for the main batch systems to run the jobs for both experiments at shared sites. A deployment campaign for enabling multicore at sites and the proper accounting has resulted in the required resources being enabled. There is still some work to do to optimize both the experiments submission and the sites scheduling where needed, but the infrastructure to run multicore is setup and already in use. We will continue to work also on the extra tasks that have been assigned, namely passing memory and time parameters to the batch system and how this integrates with the changed way the kernel handles cpu and memory limits and to test backfilling [11] at sites that wish to implement this strategy.

References

[1] WLCG Multicore Deployment Task Force, 2015, `https://twiki.cern.ch/twiki/bin/view/LCG/DeployMultiCore`

[2] 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.

[3] Perez-Calero Yzquierdo A *et al* 2015, Evolution of CMS workload management towards multicore job support, these proceedings and `https://indico.cern.ch/event/304944/session/4/contribution/409`

[4] HTCondor Homepage: `http://research.cs.wisc.edu/htcondor/`

[5] Templon J *et al* 2015, Scheduling multicore workload on shared multipurpose clusters, these proceedings and `https://indico.cern.ch/event/304944/session/6/contribution/281`

[6] TORQUE and Maui Homepage: `http://www.adaptivecomputing.com/products/open-source/`

[7] Platform LSF Homepage: `www.ibm.com/systems/platformcomputing/products/lsf/`

[8] Dal Pra S, Efficient provisioning for multicore applications with LSF, these proceedings and `https://indico.cern.ch/event/304944/session/9/contribution/455`

[9] Son of Grid Engine Homepage: `https://arc.liv.ac.uk/trac/SGE`

[10] Hufnagel D *et al* 2015, The CMS Tier-0 goes Cloud and Grid for LHC Run 2, these proceedings and `https://indico.cern.ch/event/304944/session/5/contribution/119`

[11] Passing parameters project, 2015, `https://twiki.cern.ch/twiki/bin/view/LCG/BSPassingParameters`