

Efficient provisioning for multi-core applications with LSF

Stefano Dal Pra¹

¹ INFN-CNAF, viale Berti-Pichat 6/2, 40127 Bologna, Italy

E-mail: stefano.dalpra@cnaf.infn.it

Abstract.

Tier-1 sites providing computing power for HEP experiments are usually tightly designed for high throughput performances. This is pursued by reducing the variety of supported use cases and tuning for performances those ones, the most important of which have been that of single-core jobs. Moreover, the usual workload is saturation: each available core in the farm is in use and there are queued jobs waiting for their turn to run. Enabling multi-core jobs thus requires dedicating a number of hosts where to run, and waiting for them to free the needed number of cores. This drain-time introduces a loss of computing power driven by the number of unusable empty cores. As an increasing demand for multi-core capable resources have emerged, a Task Force have been constituted in WLCG, with the goal to define a simple and efficient multi-core resource provisioning model. This paper details the work done at the INFN Tier-1 to enable multi-core support for the LSF batch system, with the intent of reducing to the minimum the average number of unused cores. The adopted strategy has been that of dedicating to multi-core a dynamic set of nodes, whose dimension is mainly driven by the number of pending multi-core requests and fair-share priority of the submitting user. The node status transition, from single to multi core et vice versa, is driven by a finite state machine which is implemented in a custom multi-core director script, running in the cluster. After describing and motivating both the implementation and the details specific to the LSF batch system, results about performance are reported. Factors having positive and negative impact on the overall efficiency are discussed and solutions to reduce at most the negative ones are proposed.

1. Introduction

During January 2014, the INFN-T1 farming group joined the WLCG Multi-Core Deployment Task Force (MCTF [1]), whose main goal was to facilitate and enable at Tier-1 sites the execution of multi-core applications meeting requirements settled by major LHC experiments, namely ATLAS and CMS.

Eventhought current batch systems are commonly able to accept and manage a great variety of jobs, this kind of activity requires special configurations in a typical WLCG farm. This is because the most common workload is provided by single-core applications, with every available slot being almost always in use, and there are many queued jobs pending for their turn to run. This implies that a multi-core job could only start when enough free slots are available at the same time in the same node, which is an extremely unlikely event for jobs requiring several slots in the same host.

As the most general case, given by a mixed workload of n -core jobs, with generic n , is too hard and complex to be efficiently handled, the MCTF agreed to work assuming the simplified



case of $n = 8$. Here and in the following multi-core stands for eight-core.

2. Traditional solutions and their drawbacks

Providing multi-core resources requires some sort of *node reservation*, meaning that single-core execution is to be avoided or prevented there, in order to enhance the probability of having enough free slots at once. Two of them are frequently adopted:

host partitioning. This is the most simple and straightforward solution. A fixed subset of the Worker Nodes of the site are exclusively dedicated to multi-core only activity. This is however unacceptable at the INFN-T1, since too many resources would remain unused when not enough multi-core activity is in progress or because the subset would be undersized during a boosted multi-core demand.

advanced reservation. This is a standard feature provided by several batch systems, LSF included. Assuming that the batch system knows the maximum expected end-time for each running job, it can select the node where enough slots are first going to be free. The node can exclusively be reserved to a specific multi-core job during a suitable time window, when a sufficient number of currently running jobs are expected to be done.

Unfortunately, the duration of most of the running jobs is too uncertain, varying from a few seconds (extremely frequent with *empty pilot* jobs) to a large upper runtime limit defined at queue level. That means that whenever a node gets reserved, a *draintime* begins, during which no single-core jobs can be dispatched there, causing free slots to be unusable until there are enough of them to host a multi-core job. Furthermore, the batch system would systematically trigger an advanced reservation for each pending multi-core job, adding more and more cpu power loss due to draining times.

The available common methods to run a mix of single and multi core jobs are not general enough to be satisfactory in our case, so a different model had to be designed.

3. The dynamic partitioning model

A desirable feature would consist in having a varying number of nodes dedicated to multi-core, according on needs. This would be equivalent to an auto-resizing host partition, whose behaviour could be defined according to the following principles:

Drain one, run many. Once selected for multi-core and put on drain, a node should then run multi-core jobs as long as possible, so to minimize the negative impact due to the draintime.

Follow the demand. Depending on the number of pending multi-core jobs, more nodes should be dedicated, up to a defined upper limit, or up to equilibrium.

Avoid emptiness. If no multi-core jobs are being dispatched to a dedicated node, this should be put back at work with ordinary single-core activity.

4. Implementation with LSF

A dynamic partitioning model such as the one described above has been implemented with LSF, which can be instructed to adopt special behaviours by customizing some of its functionalities and adding a few more scripts providing needed information and implementing the partition logic. These are:

elim The External Load Information Manager is a custom script, launched by the lsf `lim` daemon on each Worker Node that can be member of the mcore partition. It executes an endless loop, printing to its `stdout` name and value of one or more custom defined *External Load Index* at regular times.

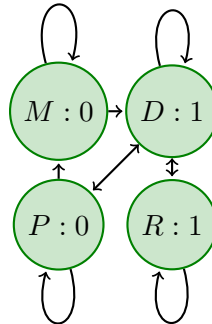


Figure 1: The status transition map.

To obtain a dynamic partition, we define a `mcore` Index, having 0 or 1 as possible values, hence behaving like a flag. WNs whose flag equals 1 are dedicated to multi-core. This means that:

- single-core jobs cannot be dispatched to nodes having `mcore==1`
- multi-core jobs must be dispatched to nodes having `mcore==1`

esub We need each new job to become *mcore aware* once submitted. This can be done by customizing the External Submitter script *esub*, which is executed at the submission host (the CREAM-CE, in our case). It distinguishes the multi-core jobs from the single-core ones by inspecting their submission parameters, then modifying their *resource request* in order to require `mcore==1` in case of multi-core jobs and `mcore!=1` for the other ones.

director A third script, the *director*, implements the logic of the partitioning model. It runs every six minutes and decides which nodes are to be added to, or removed from the `mcore` partition. The result is written in a Json file on a shared filesystem, readable by every node in the cluster. The *elim* running on each WN decides the value of its own `mcore` flag status by inspecting this file.

status The director makes use of status information, which are provided by a couple of simple C programs based on the `lsf/lsbatch.h` api. These are:

- the resource request of each pending or running job.
- the current set of unavailable or closed nodes.

configuration It is possible to tune the behaviour of the dynamic partition by configuring the hostgroups available for multi-core, specified as a list of racks in our case, the maximum number of nodes that can be in drain status at a time, the maximum acceptable number of unused slots, the higher acceptable ratio of unused slots, and several other minor details.

hosts We selected for `mcore` WNs having 16 or 24 slots, thus being capable to run up to two or three multi-core jobs.

4.1. Transitions

The director considers each node as being member of one of four disjoint sets:

- *M*: available for `mcore`. This is initially the whole set of hosts defined by the configuration file. Only single-core jobs can be dispatched to nodes in this set, and they have their `mcore` flag set to 0.
- *D*: Assigned to `mcore`. A node in this set is in drain time. It might have running single-core jobs but no more single-core can be dispatched there. It might have up to eight free slots. Nodes in this sets have their `mcore` flag set to 1.

- *R*: Running only multi-core jobs, draintime finished, no free slots. Nodes in this sets have their mcore flag set to 1.
- *P*: Purged from mcore. A node in this set comes from the *D* set, where it was found to have free room for multi-core jobs after many (default being three) consecutive checks. There are multi-core jobs still running, however single-core only can be dispatched to Nodes in this set. The mcore flag is reset to 0.

4.2. Dynamic of the partition

The dynamic partitioning works like a finite state machine, represented by Fig. 1. At any given time, a Worker Node is in the state identified by the set it belongs to.

- At $T = 0$, all WNs are w_i in the set $M = \{w_1, \dots, w_N\}$
- When $Q_m > 0$ multi-core jobs are queued, k WN are moved from M to $D = \{w_1, \dots, w_k\}$ by the director. The k value is determined by a chain of three simple formulas considering the number of pending multi-core jobs, the number of currently unused slots and the configured thresholds and limits.
- When a node completely finishes its draintime, i.e. it is full of multi-core only jobs, it is moved from D to R .
- When a node $w_i \in D$ has free room for multi-core and no jobs starts there after a timeout, it is moved from D to P .
- When more nodes dedicated to multi-core are needed, they are moved from $P \cup M$ to D , starting with those in P . The reason for doing so is that such nodes are supposed to still have running multi-core jobs. A draintime has been “paid” for each one of them. Moving back nodes from P to D helps recovering this “cost”, that would be definitely lost elsewhere.
- The elim script on each node w_i updates its mcore status:

$$mcore(w_i) = \begin{cases} 1 & \text{if } w_i \in D \cup R \\ 0 & \text{if } w_i \in M \cup P \end{cases}$$

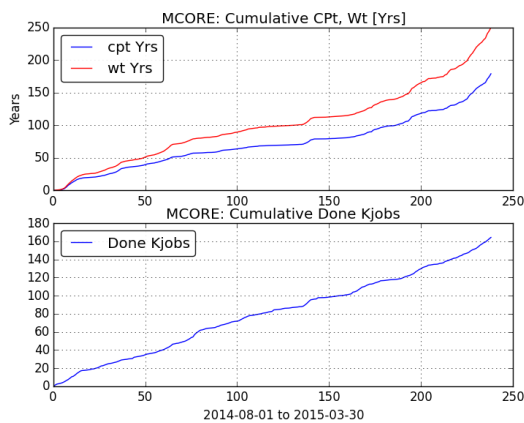


Figure 2: MCORE Done jobs

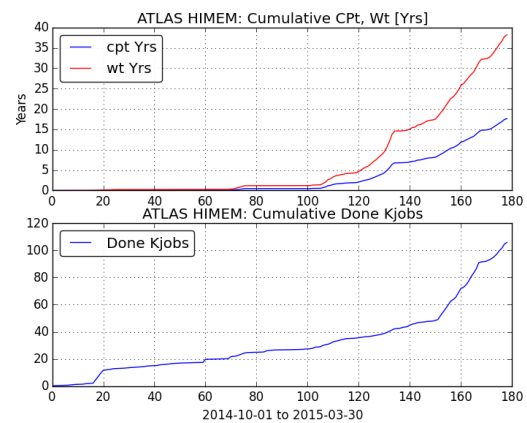


Figure 3: ATLAS HIMEM Done Jobs

5. Deployment and improvements: high memory jobs

The dynamic partitioning system begun its activity in the production batch system in 2014 August first, and proved to work successfully (Fig. 4) with no need for manual interventions.

During October, a new requirement came from the ATLAS experiment to also provide resources to the so called *himem* jobs. These are special single-core jobs requiring up to two times the RAM needed by the ordinary ones.

To deal with this case, himem jobs are treated like multi-core ones requiring two slots in the mcore partition. This way one core remains unused, however the working one has the needed amount of RAM. Furthermore, LSF is configured to run no more than four such jobs per node. Doing so, room is always granted for at least two 8-core jobs on the nodes with 24 slots.

This solution has the benefit of reducing the number of unused slots, at the cost of delaying a little the start of 8-core jobs at the node.

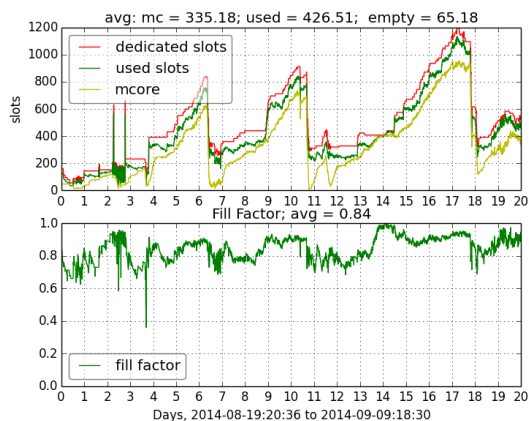


Figure 4: Dynamic partition early days, August 2014, multi-core. The space between red and green line represents unused slots. Sudden submission dropdown have negative impact in the average efficiency (see Fig. 6). Different configurations have impact on the reactivity of the system, i.e. how quickly the partition grows or shrinks.

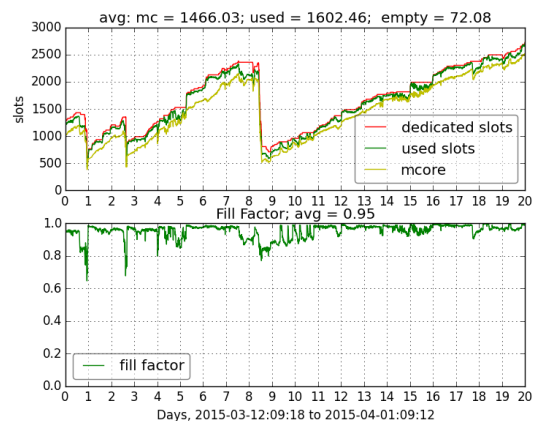


Figure 5: Dynamic partition, March 2015, multi-core and high-memory. The number of unused slots over time is much reduced respect to the case of Fig. 4. By inspecting the *used slots* line it can be noted how the fill factor decreases when there is lack of himem jobs. The partition reaches greater size and dropdown are less frequent.

Month	cpu	VO	n	CPT_days	WCT_days	%Eff	E[CPT] [h]	E[WCT] [h]
2015-03	1	atlas	338737	57508.103	60272.685	95.413	4.075	34.163
2015-03	1	cms	207049	57664.471	74034.616	77.890	6.684	68.654
2015-03	2	atlas	52900	3529.259	7683.582	45.932	1.601	13.944
2015-03	8	atlas	23848	13799.668	18293.281	75.436	13.888	18.410
2015-03	8	cms	2798	8511.604	11948.709	71.235	73.009	102.491

Table 1: Activity by core and VO, March 2015. Efficiency is computed as $\frac{\sum CPT}{\sum WCT}$. Himem jobs are given twice the CPU and RAM assigned to single-core jobs, but only take advantage of the extra RAM. this is why their efficiency appears less than 50%.

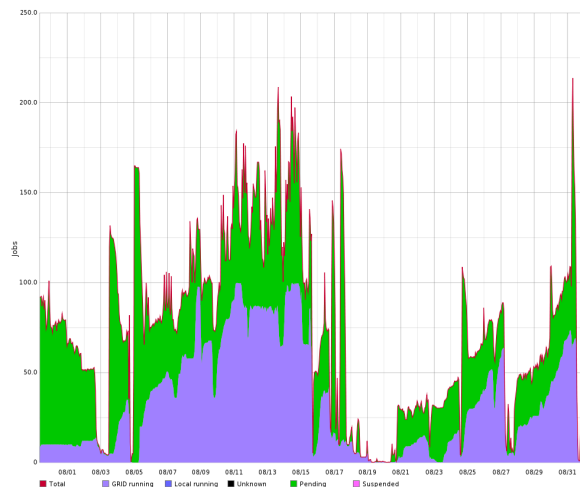


Figure 6: Atlas, multi-core, 2014 Aug. The submission flow suddenly interrupts several times. Nodes in the mcore partition are put back at work with single-core jobs. Short after, submission flow restart, triggering need for new draining.

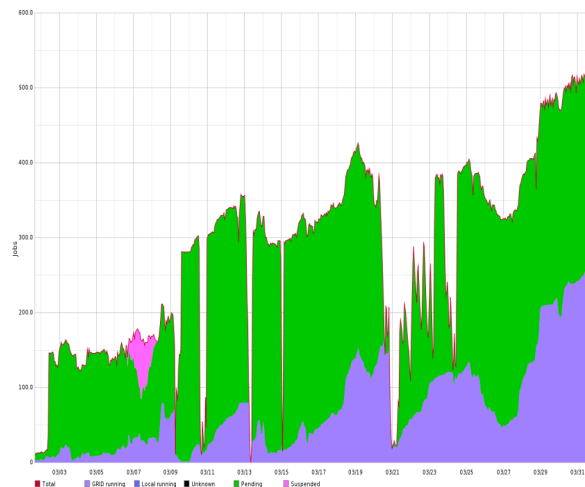


Figure 7: Atlas, multi-core and high-memory, March 2015. The submission flow appears to be more regular and the volume of running jobs more than doubled. Recovering after a dropdown requires days.

6. Results and comments

The dynamic partitioning system started in August 2014, enabling access to compute resources for 8-core jobs. Two months later it was adapted to also enable high-memory jobs. Fig. 2 and 3 reports cumulative CPUtime and WallclockTime for multi-core and high-memory activity. Tab. 1 reports *per core* accounting data for March 2015.

In order to evaluate performances, a *Fill Factor* was considered, defined as $FF = \frac{used\ slots}{dedicated\ slots}$, with optimal value $FF = 1$. Fig. 4 shows a quite poor behaviour, with an average fill factor $FF = 0.84$ and an average partition size of 335 slots, 65 of which unused. On the other hand, the partition grows and shrinks itself according to the multi-core jobs submission flow. Moreover, during time periods of almost steady submission flow, the fill factor increases near to 1.

Fig. 5 reports much better performances, with $FF = 0.95$ and an average partition size of 1466 slots, 71 of which unused. The improvement is due to a more regular submission rate and to high-memory jobs, running in the mcore partition with 2 assigned slots each.

6.1. On the dynamic of the partition

By inspecting Fig.4 and Fig.5, it can be noted how the slope of the “dedicated_slots/day” trend represents the analogous of a “step response” in a dynamic linear system, and can grow as fast as 200 to 350 slots/day. This implies a consequent limit on how quickly the number of running multi-core jobs can grow over time. this is a much lower limit respect to the single-core case, where a rump-up of ~ 400 jobs/hour can be easily observed, having noticeable consequences. If a VO stops submitting jobs for a while, its priority in the batch system increases while the running jobs are fading. When the VO restart submissions, its single-core jobs are dispatched as soon as a slot gets free (i.e. another job finishes). Multi-core jobs, instead, have to wait longer, because 8 single-core jobs have to finish *in the same node*, which requires much more time than that needed to have 8 single-core jobs started *wherever possible*.

6.2. On the mixing-ratio problem

Both ATLAS and CMS declared the intention to divide their CPU quotas 50% or more between multi-core and single-core. From the site point of view, this may be enforced by properly

configuring sub-shares in the LSF batch system. Doing so, however, would require different job types to have different submitting users, which is not how the experiments are planning to do. Should this requirement be unmet, the only feasible alternative relies at the submitter side, which has to carefully mix submissions so that to drive the percentage of their running jobs to float around the optimal desired value. This in turn, requires careful consideration to the partition size, which actually limits the (much slower) multi-core dynamic. In this sense, frequent interruptions in its submission rate can seriously prevent the possibility for the desired percentage to be met. Worth to mention is the fact that if one experiments plan to only submit multi-core, then the overall dynamic of running jobs will be slower, as said, up to the point that the currently adopted HIST.HOURS [3] parameter in LSF for the calculations of the fairshare job priorities might be adapted in order to remain effective even with slower dynamics.

6.3. future improvements

The most desirable feature on a dynamic partition is given by its grow rate; the higher, the better. Currently, hosts to put on drain are randomly selected from the M set. However, enough information is available to the director to permit better choices, such as selecting nodes running jobs having the overall lowest “life expectation”. A further improvement would be achieved by preventing longest single-core jobs to be dispatched to nodes participating to mcore. Having jobs declaring their maximum lifetime at submission time would be of great benefit in this respect.

7. Conclusions

A dynamic partitioning system for the LSF batch system has been designed and implemented at INFN-T1. It works smoothly and provides resources for multi-core and high-memory jobs. The efficiency of the the system has been measured by monitoring and logging the number of dedicated and unused slots over time. The partition director exhibits more efficient behaviour with smooth job submission flows and suffers in the event of sudden interruption and restart, because of the higher need for draining times (Fig. 6). High-memory jobs are helpful to reduce the number of unused slots on the draining nodes. Having more independent multi-core submitting agents also helps in preventing partition collapse. The grow rate of the partition limits the grow rate of the number of running multicore jobs, and this encourages the submitter to provide a reliable smooth multi-core job submission rate.

References

- [1] <https://twiki.cern.ch/twiki/bin/view/LCG/DeployMultiCore>
- [2] S. Dal Pra “Job Packing: optimized configuration for job scheduling”, HEPiX Spring 2013 Workshop, <https://indico.cern.ch/event/220443/session/5/contribution/9>
- [3] LSF Admin guide <http://www-304.ibm.com/support/customer/sas/f/plcomp/platformlsf.html>