Efficient provisioning for multicore applications with LSF

Stefano Dal Pra
INFN–CNAF, stefano.dalpra@cnaf.infn.it

The multi–core problem

- As of 2014, The INFN–T1 farm, composed by 15000 cores, manages single–core jobs only.
- Every slot is almost always in use.
- There never is room for multi–core (8–slots) jobs!

The solution: dynamic partitioning

- mcore partition: a varying number of nodes dedicated to multicore and high–memory (2 slots) jobs.
- The partition automatically grows and shrinks on a per–need basis.
- When a WN is assigned to the mcore partition, LSF stops dispatching single–core jobs to it.

The implementation

- elim script runs on the WN and defines the status of the `mcore` flag.
- esub script runs at the submission host for each submitted job and modify its parameters: the multi–core requires `mcore==1`, any other requires `mcore!=1`
- director script implements the logic of the partitioning model. It runs every 6 minutes on the LSF master and selects which WNs are to join or leave the partition.
- Status information are essential. These are queried to LSF by a couple of C programs, specific for the needed data.
- tunable: no hard–coded settings.

Features

- Dedicates WNs to (only) multicore jobs on need.
- Reclaim them back to ordinary single-core jobs when they remain free after a configured time.
- Designed to work with {8,4,2}–core jobs.

Mcore status

WNs are moved across the following sets:

- M: available for multicore, only s–core jobs
- D: assigned to the mcore partition.
- R: running only multi–core jobs.
- P: purged from the mcore partition. only single–core are dispatched to it, there still are running multi–core jobs

Dynamic of the mcore partition

- At $T = 0$, all WNs are $w_i \in M = \{w_1, \dots, w_N\}$
- 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.
- When a node is full of multi–core, it is moved from D to R.
- When a node $w_i \in D$ has free room for a multi–core and no jobs starts there after a timeout, it is moved from D to P.
- When more multi–core nodes are needed, they are moved from $P \cup M$ to D, beginning with P.
- The elim script on each node w_i updates its mcore status:

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

$$\begin{tikzcd}
M:0 \arrow[r] \arrow[loop above] & D:1 \arrow[loop above] \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] \\
P:0 \arrow[u] \arrow[loop below] & R:1 \arrow[loop below]
\end{tikzcd}$$

Figure: The Status Transition Map

Conclusions

- Dynamic partitioning is an efficient solution to provision MultiCore resources
- More efficient with smooth job submission flows
- Suffer with sudden interruption and restart, because of the higher need for draining.
- High–memory jobs are helpful to reduce the number of unused slots on the draining nodes.
- Having more independent multicore submitters also helps in preventing partition collapse.

Multicore Activity (Aug. 2014 and Mar. 2015)

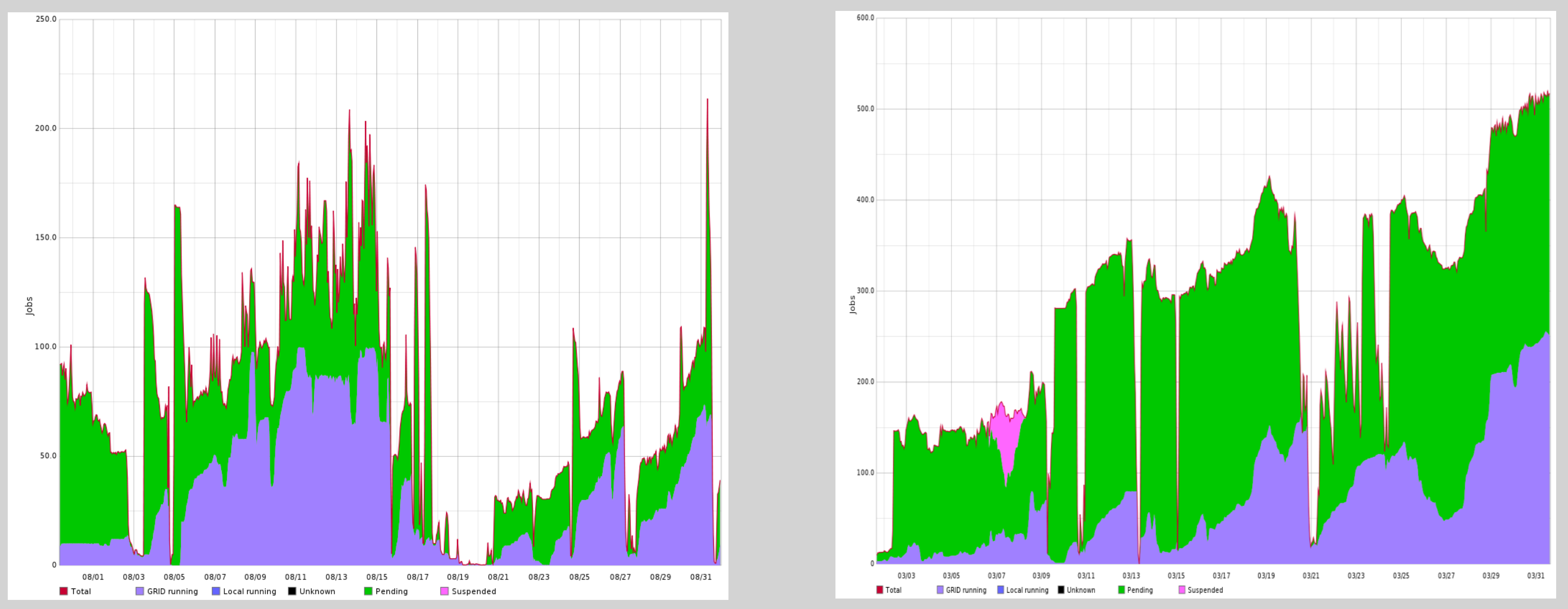

Figure: multi–core, running and pending jobs. When the submission flow suddenly stops, nodes in the mcore partition remain empty and are put back at work with single–core jobs. Short after, submission flow restart, triggering need for new draining. This is unefficient and depends on how smooth the multi–core submission flow is. Having many independent submitters may help reducing impact of discontinous submission flow.

Mcore partition size and usage, Aug. 2014 and Mar. 2015

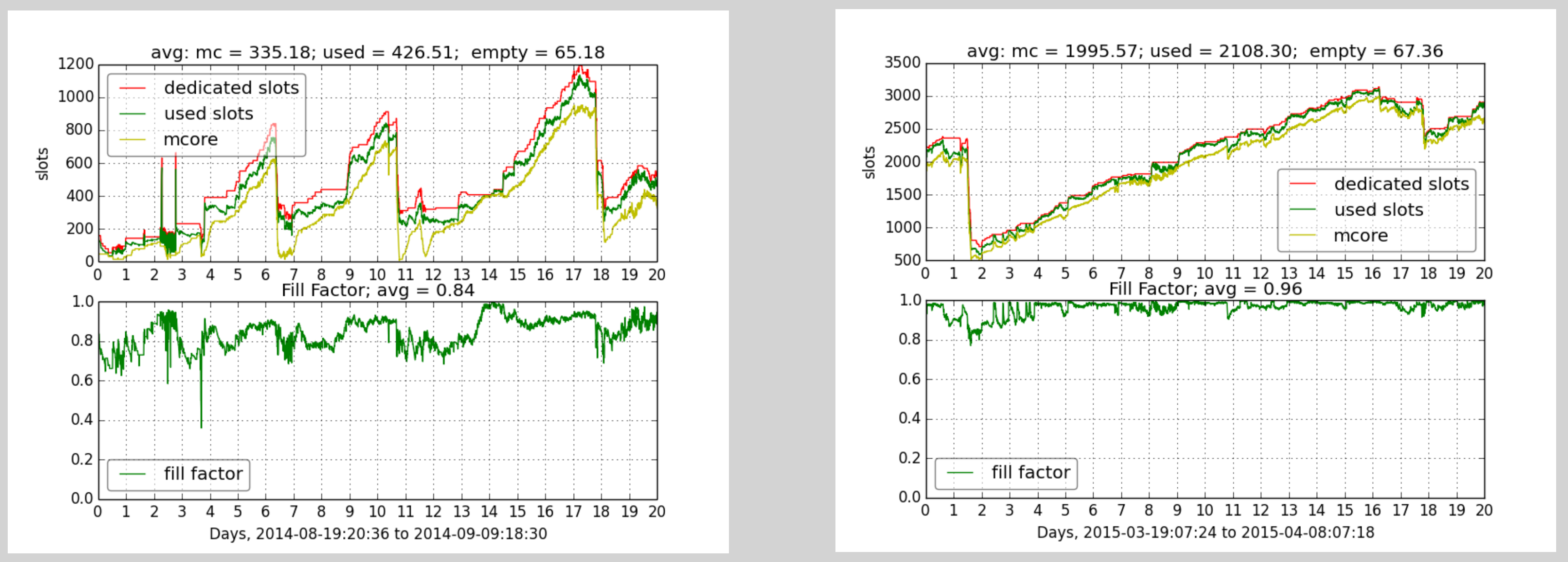

Figure: Dynamic partition size, August 2014 and March 2015. The space between red and green line represents unused slots. Sudden stop in the submission flow have negative impact in the average efficiency. Different configurations have impact on the reactivity of the system, i.e. how quickly the partition grows or shrinks. The second chart shows an improved Fill Factor thanks to HIMEM jobs and more regular submission flow.

HIMEM (2 slots) and MCORE (8 slots) mix, 7 days

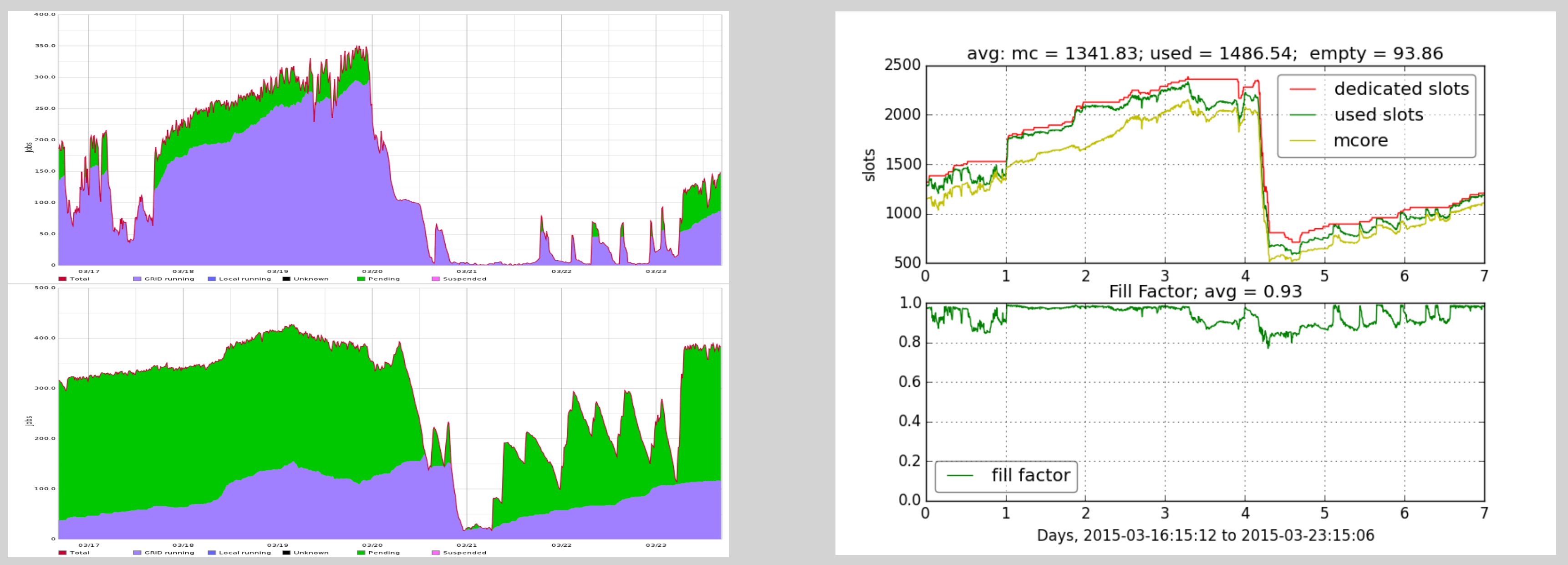

Figure: HIMEM (upper–left), MCORE (bottom–left). HIMEM are single–core jobs requiring up to twice the ram needed by a common job. They are modified by the esub to require 2 slots in the mcore partition. This permits to reduce the number of unused slots. By inspecting the *used slots* line it can be noted how the fill factor reduces when the himem job submission flow stops.

Production by number of used cores

Month	*cpu*	*VO*	*n*	*CPT_days*	*WCT_days*	*%Eff*	*E[CPT] [h]*	*E[WCT] [h]*
2015-03	1	atlas	338737	57508.103	60272.685	67.217	4.075	4.270
2015-03	1	cms	207049	57664.471	74034.616	39.111	6.684	8.582
2015-03	2	atlas	52900	3529.259	7683.582	43.272	1.601	3.486
2015-03	8	atlas	23848	13799.668	18293.281	70.298	13.888	18.410
2015-03	8	cms	2798	8511.604	11948.709	28.167	73.009	102.491

Table: Activity by core and VO, March 2015. Noticeably, the average efficiency of multi–core jobs ($cpu = 8$, atlas) is higher than that of single–core. This partially compensates the cpu powerloss due to draining.

References

- `https://twiki.cern.ch/twiki/bin/view/LCG/DeployMultiCore`
- A. Forti et al. "Multicore job scheduling in the Worldwide LHC Computing Grid", CHEP–2015, `http://goo.gl/gbwSh7`
- S. Dal Pra "Job Packing: optimized configuration for job scheduling", HEPiX Spring 2013 Workshop, `http://goo.gl/kad3cg`
- LSF Admin guide `http://goo.gl/tZ0fmj`