

Multicore Deployment Task Force status report

WLCG Collaboration Workshop

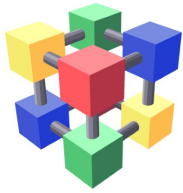
Barcelona, July 7th 2014

Alessandra Forti and Antonio Pérez-Calero
Yzquierdo

for the WLCG Multicore deployment Task Force

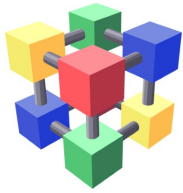


WLCG
Worldwide LHC Computing Grid



Outline

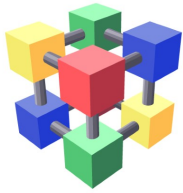
- Multicore jobs in WLCG
- The problem of multicore job scheduling
- The WLCG multicore deployment Task Force
- Results and current status
- Conclusions and Outlook



Multicore jobs in WLCG

Looking at the **restart of the LHC** data taking in 2015, experiments are developing **multicore applications** due to:

- **Hardware evolution:** over the last decade architecture design goes in the direction of adding processors to the CPU, while individual core performance will probably not increase significantly
- **Evolution of LHC conditions: higher data volumes** to be processed, with **increased event complexity** due to higher pileup, causing increasing
 - **processing time per event**
 - **memory usage**



WLCG Multicore Deployment TF

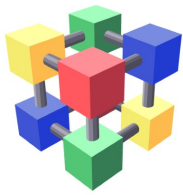
Evaluate:

- **Multicore capabilities of local batch systems**
- **Compatibility of approaches to multicore job distribution by different LHC VOs**

This talk: summary of the activities of this task force over the last months.

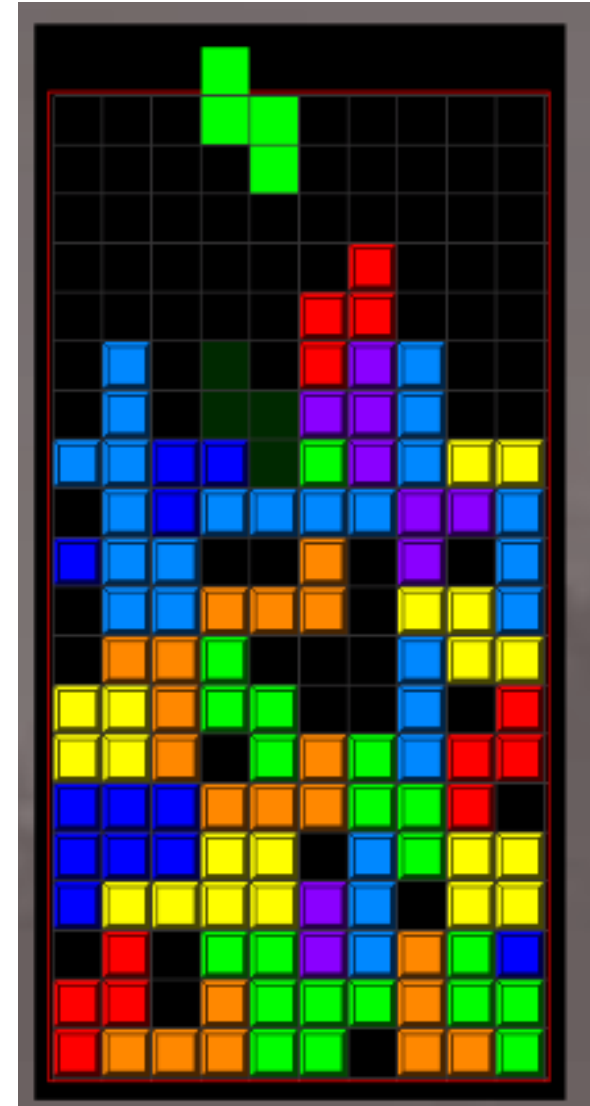
- **Acknowledgements: thanks to all the participating people from experiments and sites, which provided the content for this talk!**

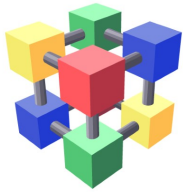
Project twiki: <https://twiki.cern.ch/twiki/bin/view/LCG/DeployMultiCore>



Multicore TF objectives

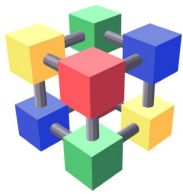
- Integrate **scheduling of both multicore and single-core jobs**, that will still be used by LHC experiments, as well as other VOs in shared sites.
- **Avoid splitting resources**, such as dedicated whole node slots and separated queues, which may introduce additional inefficiency and complexity in site resources configuration and management.
- **Maximize CPU usage**: minimize idle CPUs while there is job to be done





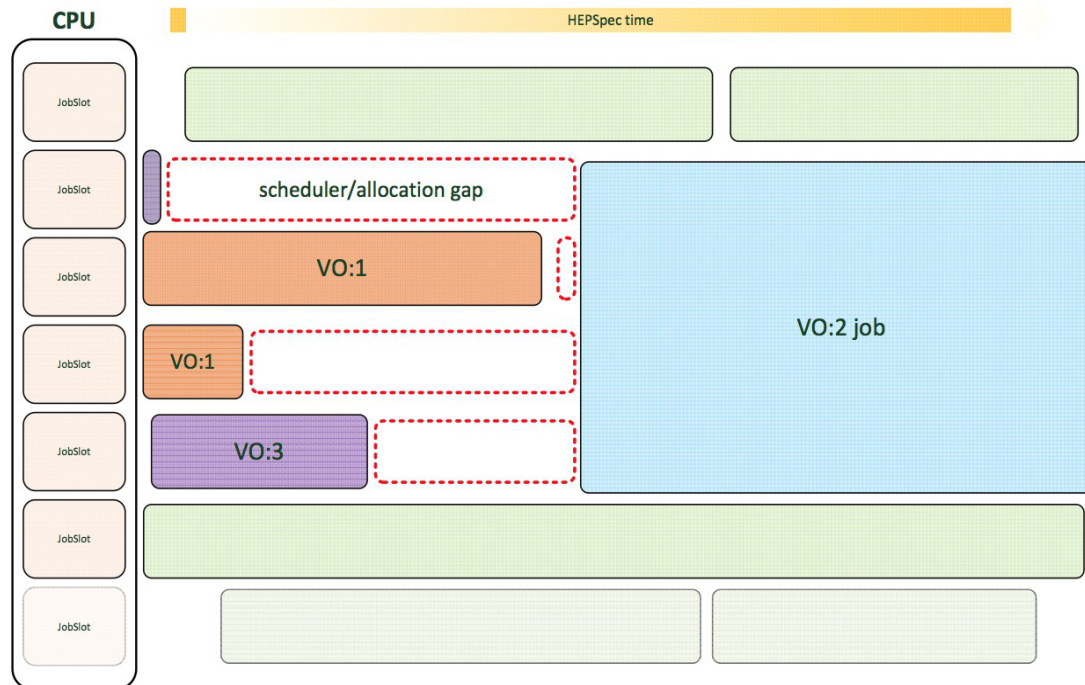
Review of batch systems

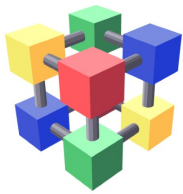
- We have **reviewed** batch systems in terms of their **functionalities** useful for multicore scheduling
- **Experience** related to:
 - **ATLAS**, initially, as multicore jobs in production **since January**
 - **CMS**, running at T1s **since May**
- **Mini workshops** dedicated to each technology
 - **HTCondor** (RAL), **UGE** (KIT), **Torque/Maui** (NIKHEF, PIC), **SLURM** (CSCS)
- Main conclusion: there is a solution for **most popular batch systems to support multicore jobs**
 - Native functionalities
 - In some cases, complementary scripts are needed
- **System configuration** (*tuning*) depends on site load composition and running conditions



Scheduling multicore jobs

- Key problem: in order for a multicore job to start in a non-dedicated environment, the machine needs to be sufficiently **drained**
- **Creating a multicore slot:**
 - Prevent single core jobs from taking freed resources
 - **draining = idle CPUs!**
 - Higher priority single core arrives and occupies slots
 - **wasted draining!**

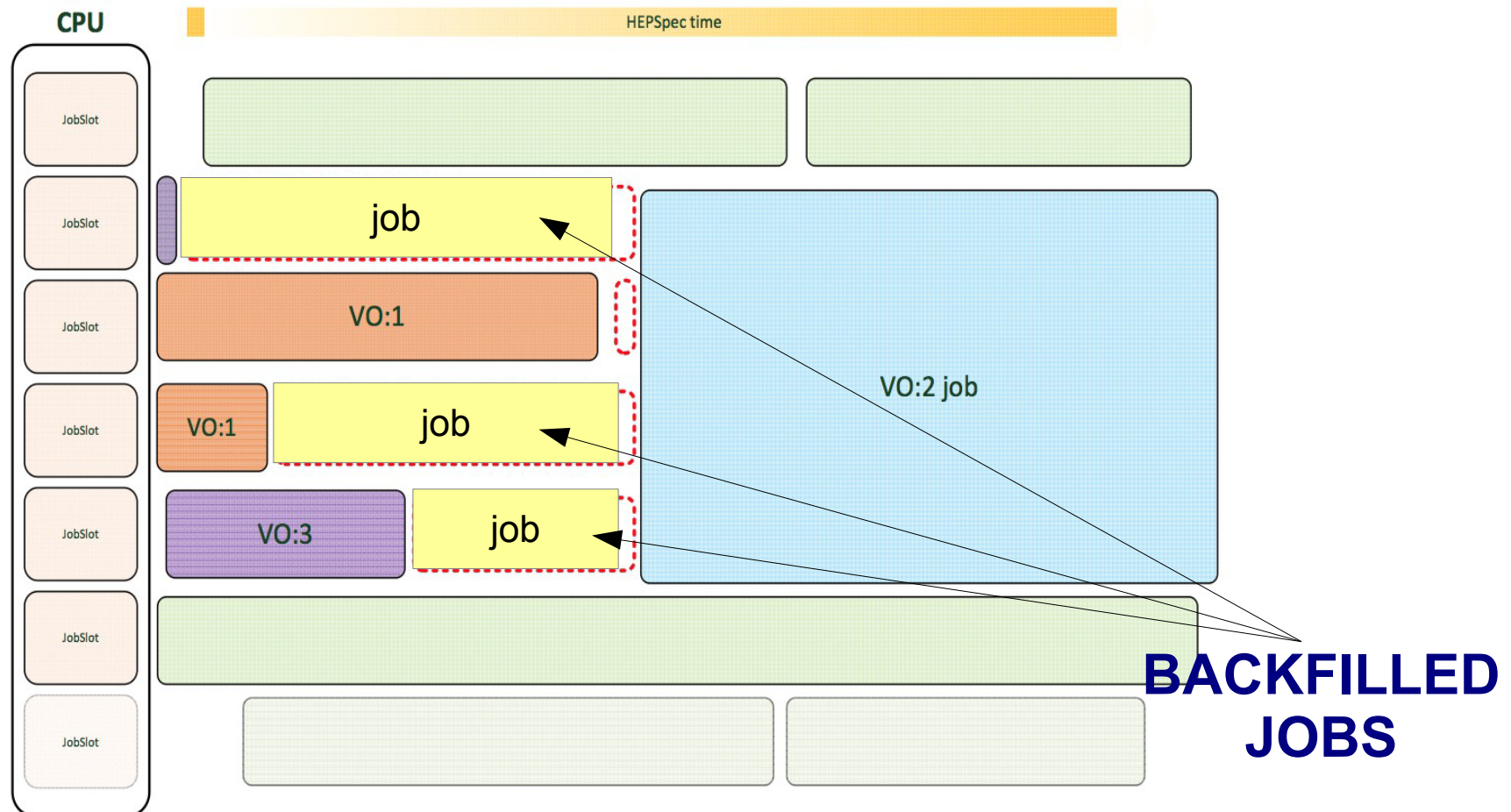


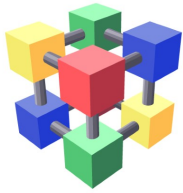


Scheduling with backfilling

Backfilling can be used to reduce the amount of idle CPUs caused by the WN draining:

- **Jobs of lower priority** are allowed to utilize the reserved resources only if their prospective job end (i.e. their declared **wallclock usage**) is before the start of the reservation

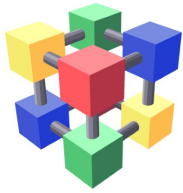




Scheduling with backfilling

The ability of the scheduler algorithm to perform **successful backfilling** depends on the concepts of **entropy** and **predictability**

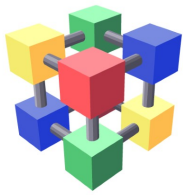
- **Entropy**: having a variety of jobs with different requirements in the queue. There should be a distribution of jobs resources requests in order to increase the likelihood of finding the right "piece" to fill each temporary hole in draining WNs
- **Predictability**: reasonably accurate prediction for jobs running time, so that the scheduler can make a decision on whether it should run this job in that hole or not.
 - How accurate this prediction needs to be?



Job running time estimation

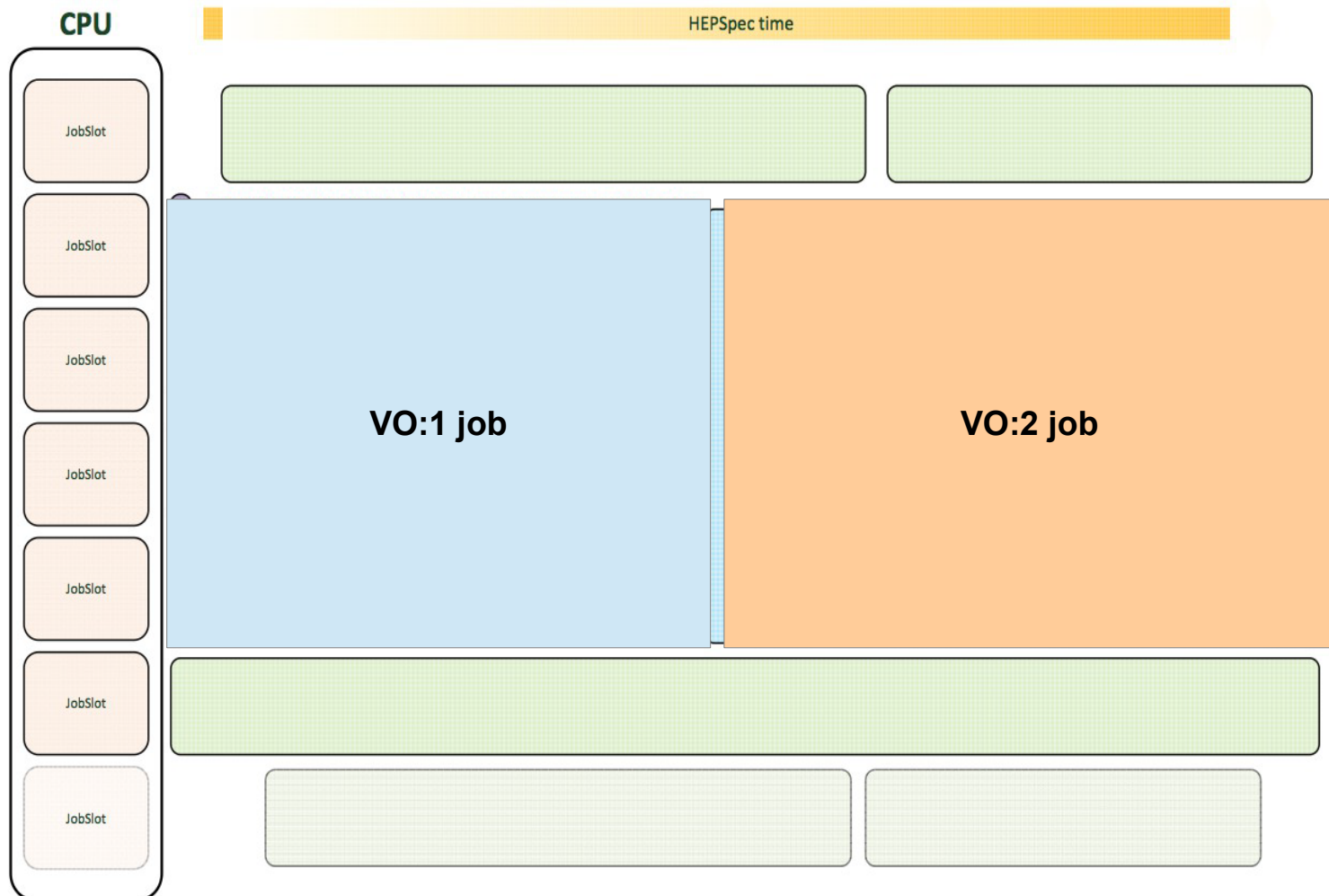
Providing a reliable estimation of job running times is however difficult for various reasons:

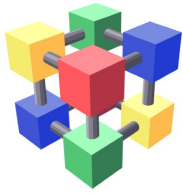
- **Inherent to the jobs themselves**, as the instantaneous luminosity and pile-up determine the complexity of events and thus the job running time
 - different for analysis, MC production and data reconstruction/reprocessing
 - there are currently ways to mitigate this, for example data reconstruction workload distributed in a number of jobs with approximately equal running time
- **Access to input data waiting times**: unpredictable in a complex environment such as the WLCG
- **Variance in CPU power** for WNs distributed across the grid and also within sites
 - This may not be so much of a problem if the actual different between the faster and lower machines at a given site still provides an estimation accurate enough to do some backfilling
- **The masking effect of pilots**: submission of jobs through pilots introduce some other effects, such as running more than one job per pilot, waiting for new jobs to appear, etc.



Conserving the slots

- There are two aspects of the problem: creating and conserving multicore slots
 - **Once the cost has been paid, avoid multicore slot destruction**

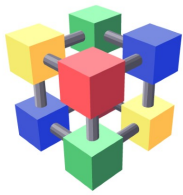




Conserving the slots

Keep the multicore slots alive:

- **Continuous and stable supply of multicore jobs**
 - so that the vacated slots can be filled with new multicore jobs
 - **avoid bursty submission patterns**, which force the system to continue and re-adjust the level of draining
- Different VOs should agree on a **common slot size** so that they can access the same slots in shared sites.
 - This is well understood and there is general consensus that there should exist at least a default value (for example 8)
- **Rank expressions/job priorities:** allocate multicore jobs to multicore slots, instead of getting partially filled by single core jobs.

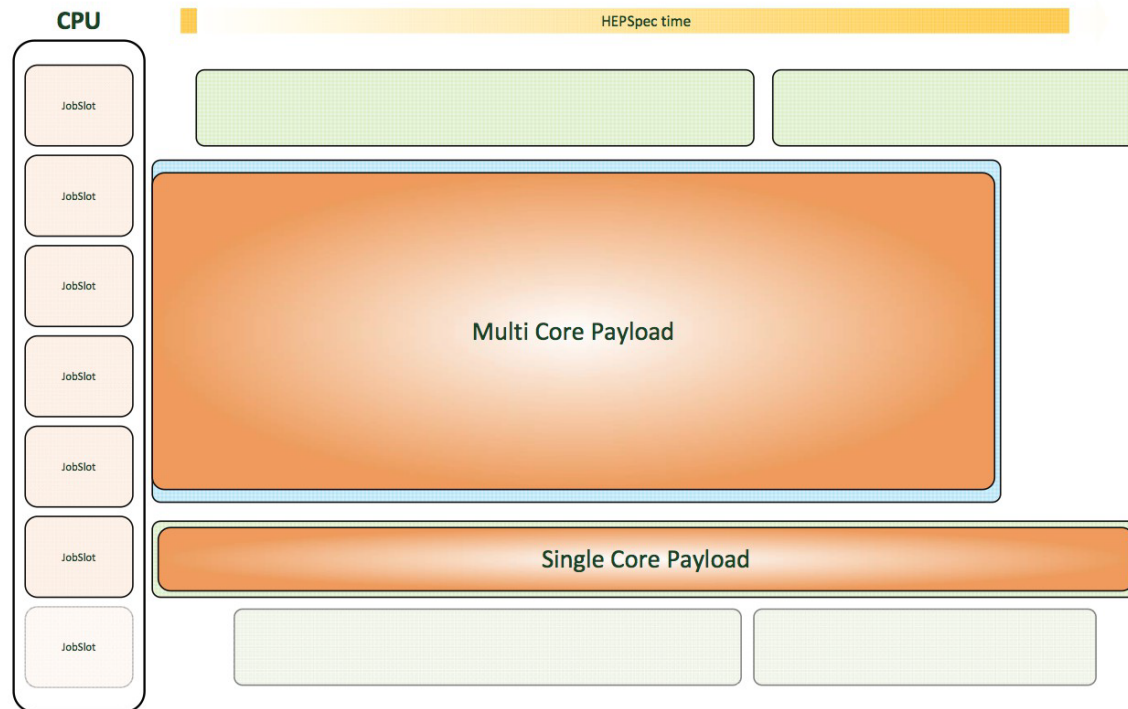


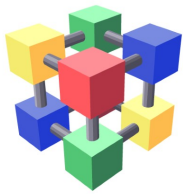
Multicore job submission models: ATLAS

ATLAS job submission principles:

- keep single core and multicore jobs separated
- one pilot pulls only one payload.

Its strong point would be the **entropy**. The experiment submission system is **being adapted** to provide the job running times to the batch systems.





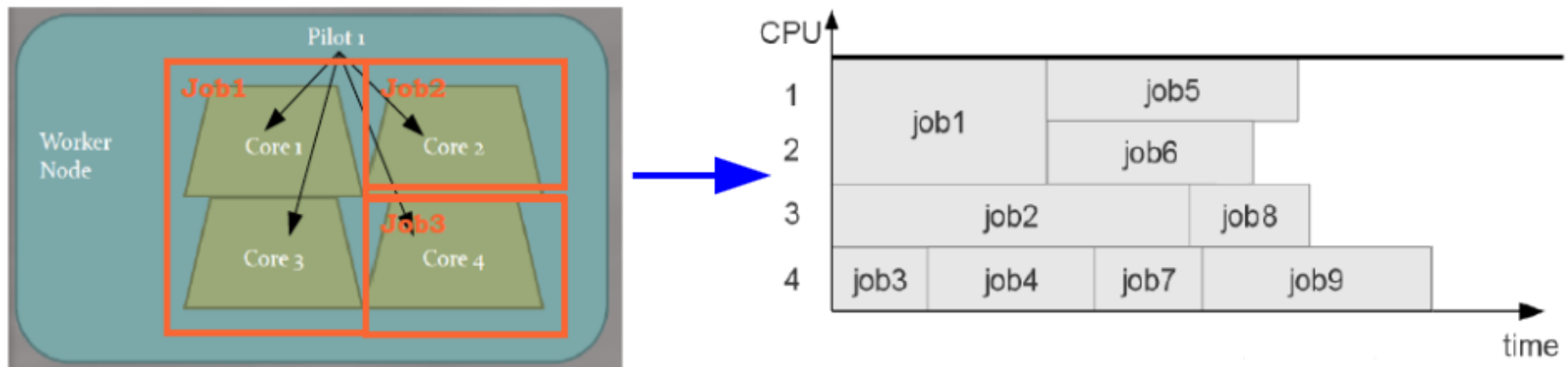
Multicore job submission models: CMS

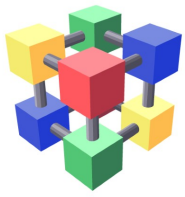
CMS job submission principles:

- **single core and multicore jobs into the same pilots** thanks to some internal machinery (glideinWMS partitionable slots) that can handle the mixture of jobs
- pilots **continue pulling jobs** until they exhaust the allowed walltime limits

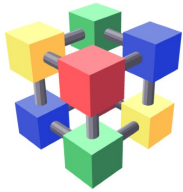
In relation to a backfilling strategy, the strong point of this model would be the **predictability**, although it would tend to **reduce the entropy** from the system.

However, by pulling multiple payloads, it contributes to **keeping the multicore slots alive**





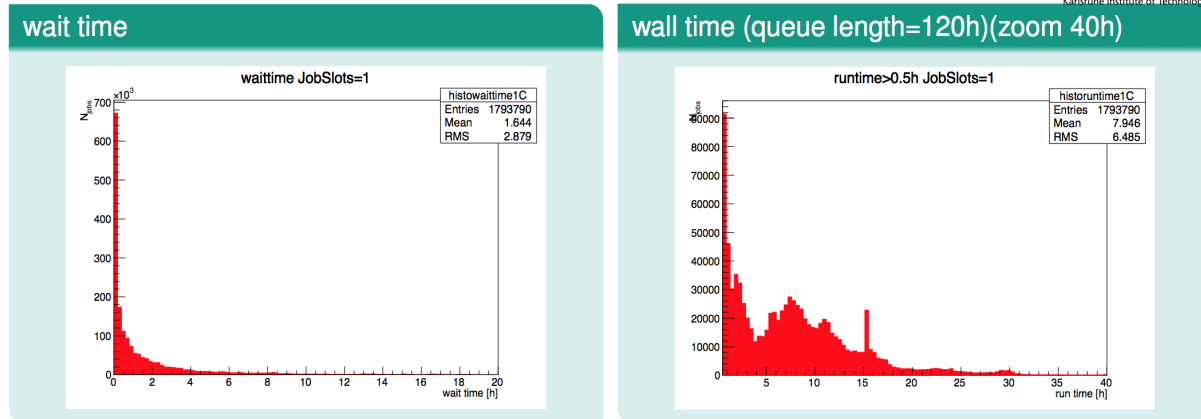
- Results



Initial results from ATLAS

- Most sites were initially exposed to ATLAS jobs, as they started earlier with MC production via multicore jobs. **Experience from KIT:**

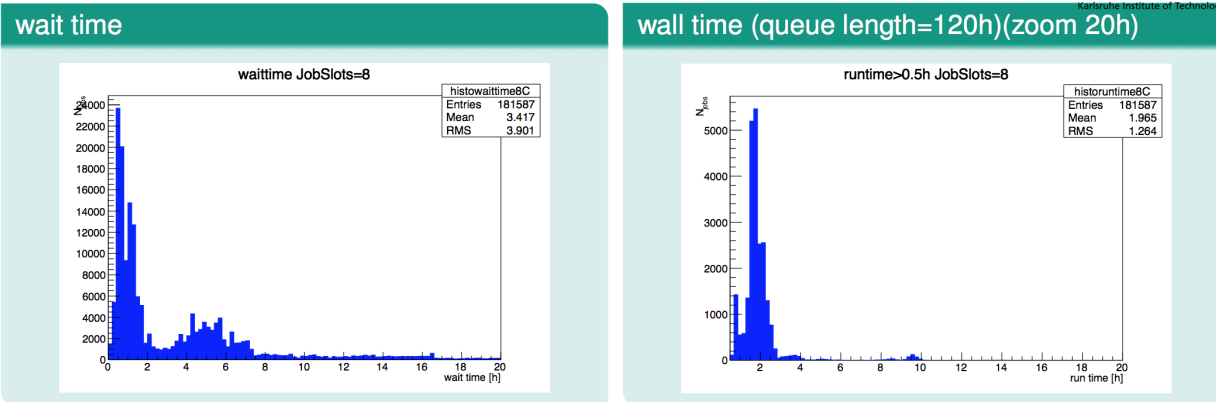
KIT: SingleCore Statistics 2014.Jan



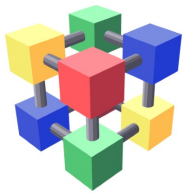
Multicore jobs: with longer waiting times combined with short running times

The cost of draining the slot to run a multicore job is not fully exploited by short running jobs

KIT: MCore Statistics 2014.Jan

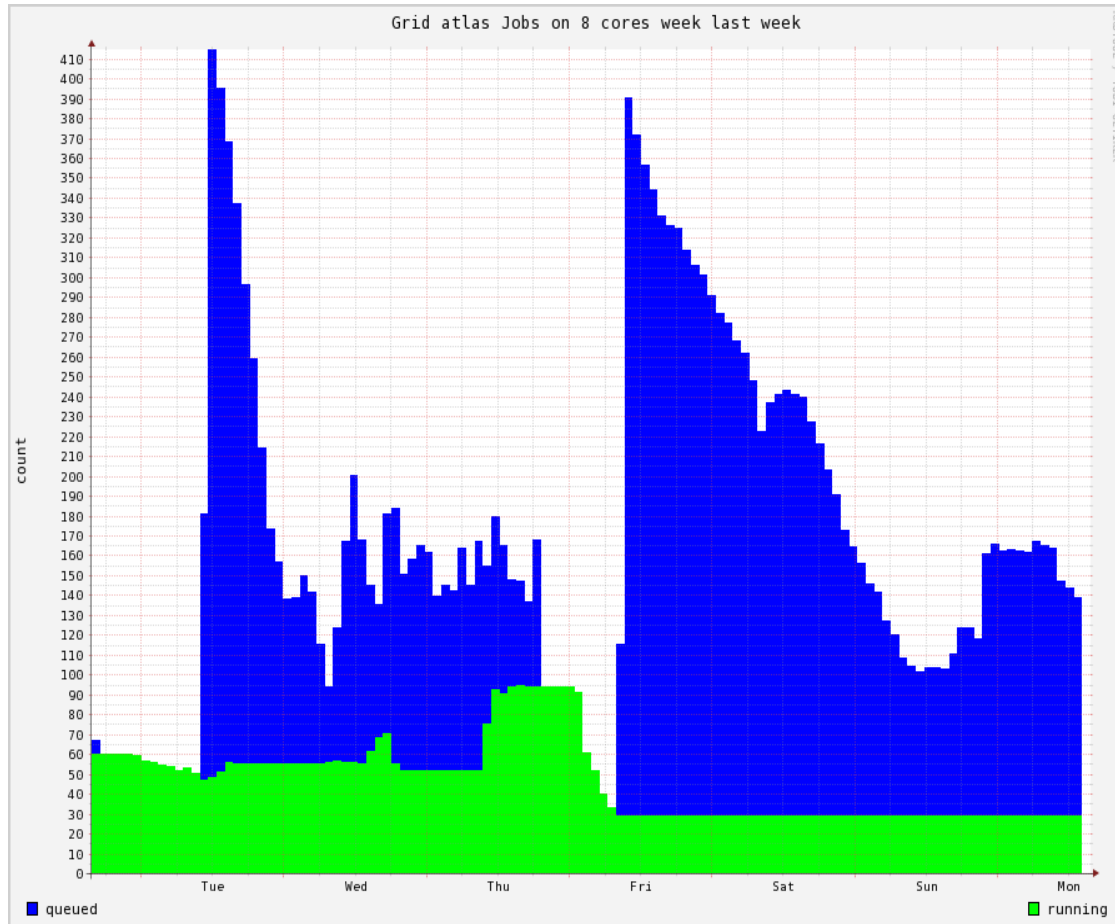


Ref: <https://indico.cern.ch/event/298062/contribution/3/material/slides/0.pdf>



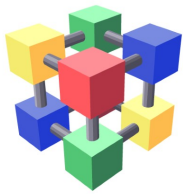
Initial results from ATLAS

- Most sites were initially exposed to ATLAS jobs, as they started earlier with MC production via multicore jobs. **Experience from KIT:**



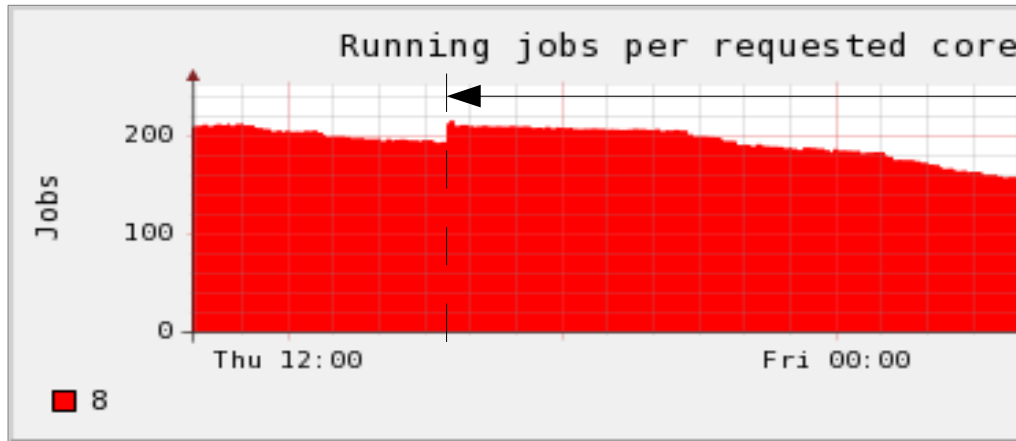
Bursty multicore jobs submission patterns observed

Ref: <https://indico.cern.ch/event/298062/contribution/3/material/slides/0.pdf>



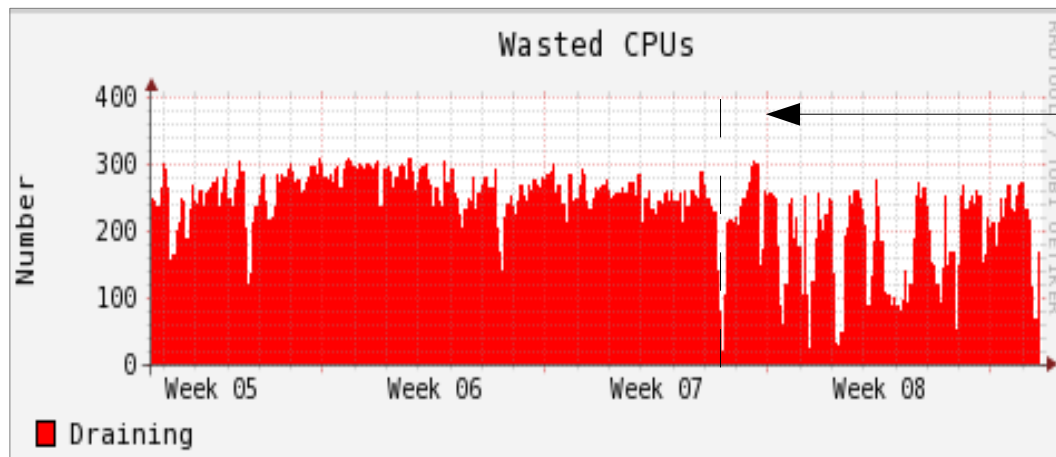
Initial results from ATLAS

- Most sites were initially exposed to ATLAS jobs, as they started earlier with MC production via multicore jobs. **Experience from RAL:**



Cancelled draining nodes.

Conclusion: need to constantly drain slots to maintain the number of running multicore jobs

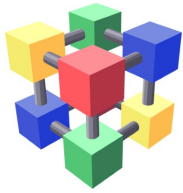


Constant draining means keeping idle CPUs all the time

Draining rate retuned in an attempt to reduce wastage

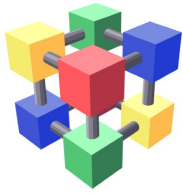
Conclusion: the draining rate has been tuned according to the amount of running and queued multicore jobs

Ref: <https://indico.cern.ch/event/298065/contribution/0/material/slides/1.pdf>



First conclusions from ATLAS jobs

- When **no backfilling is available** due to the lack of running time estimates, **draining nodes has a cost** in idle CPUs
- The **impact of the degradation** of CPU usage as a consequence of draining depends on the size and load composition of the site.
 - However, even if small, **could be extremely important**, as in general funding is linked to good results
- **Short multicore jobs** do not fully exploit the effort made in creating the multicore slot for them
- Job **submission patterns** affect tuning, performance and wastage of the system
 - Wavelike patterns require to **constantly tune the amount of draining needed**



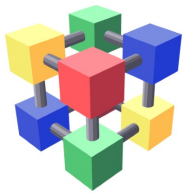
Multicore slot conservation

With no backfilling available, sites are opting for **conserving multicore slots** as the preferred strategy:

- It accommodates both **CMS** and **ATLAS** models
- It is a **native solution** for a number of batch systems (e.g. UGE at KIT), while other may require **additional components** (e.g. Mcfloat, see next slide).

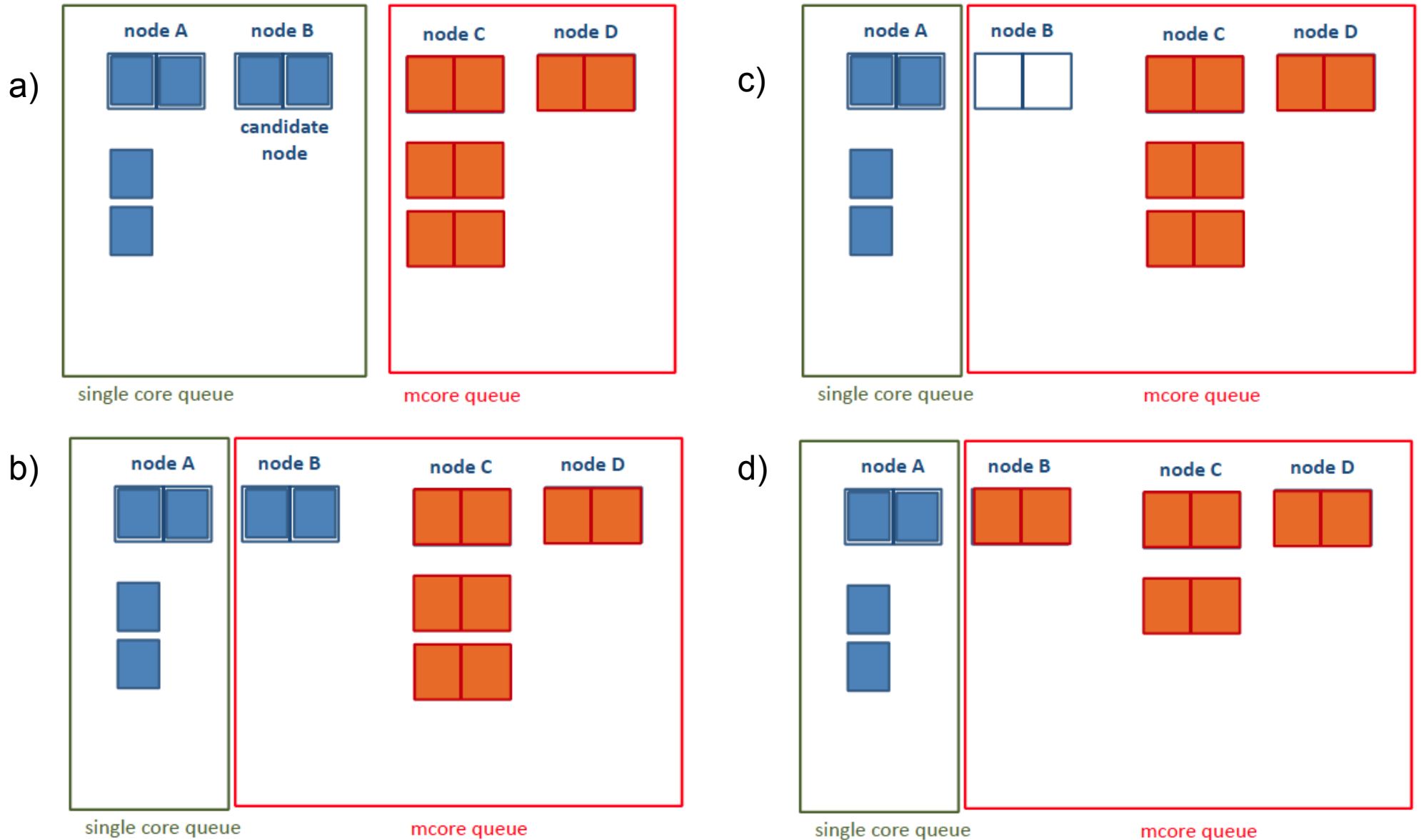
One implementation of such strategy is the **dynamic partitioning** of site resources

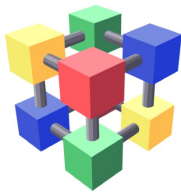
- Moving WNs between separated pools for single and multicore jobs
- The boundary between the two partitions is adjusted dynamically to load variations:
 - **no draining is needed to support a constant multicore job load**
 - **draining in a very controlled amount:** only a small percentage of the total number of cores in a site being drained simultaneously (e.g. 1-2% NIKHEF)
 - increasing the fraction of resources for multicore jobs may take quite a long time



Dynamic partitioning

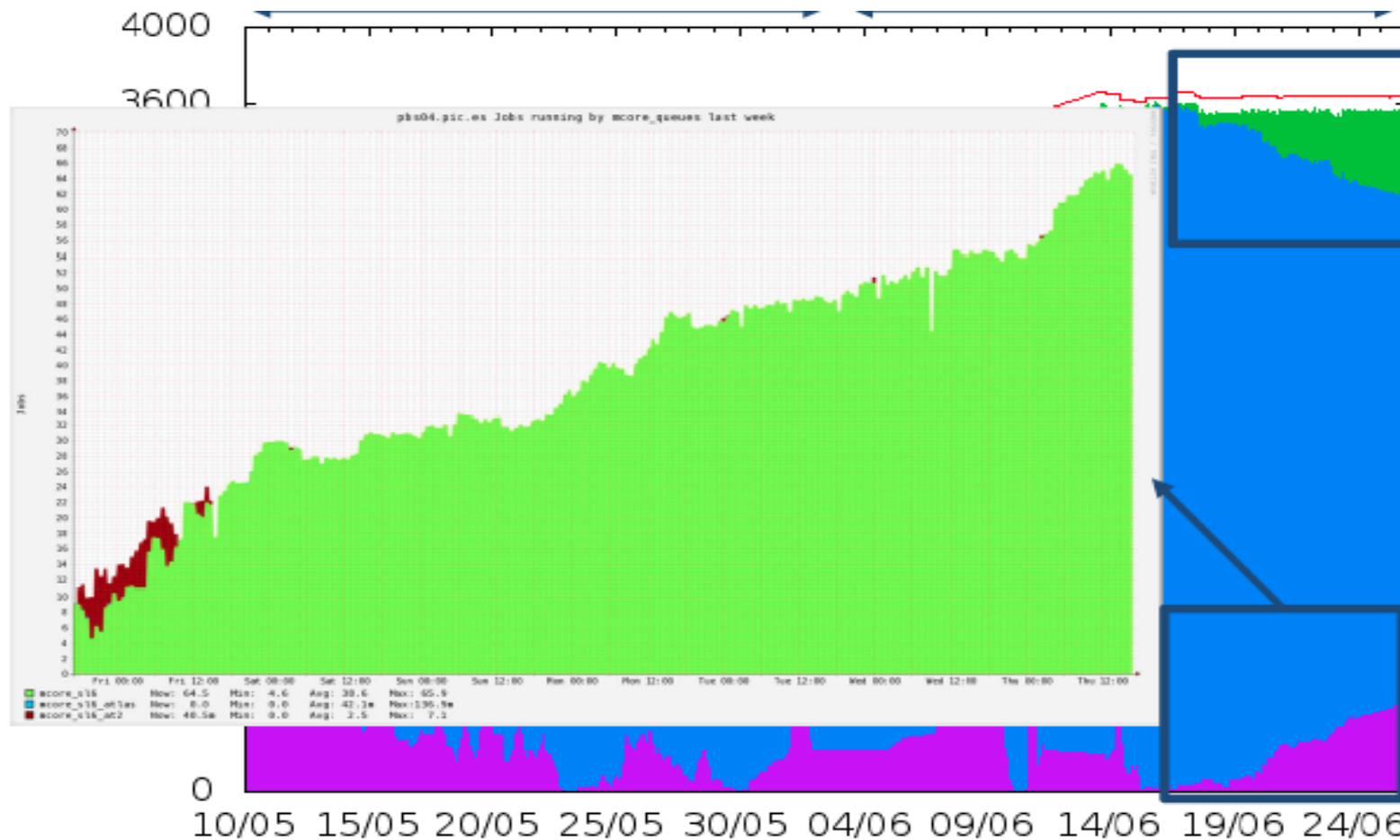
Mcfloat for Torque/Maui developed at NIKHEF, used also at PIC





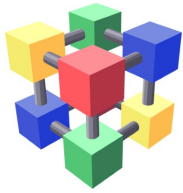
Results from CMS jobs

- **Main conclusion:** once sites have been exposed to ATLAS multicore jobs **no additional adjustments** have been necessary for CMS jobs (KIT, RAL and PIC reports)
 - thanks to the **agreement on the slot size** (8 cores)
- CMS multicore **job submission more stable** than that of ATLAS
- Results from PIC using mcfloat (second half of June)



CPU usage:
~98%

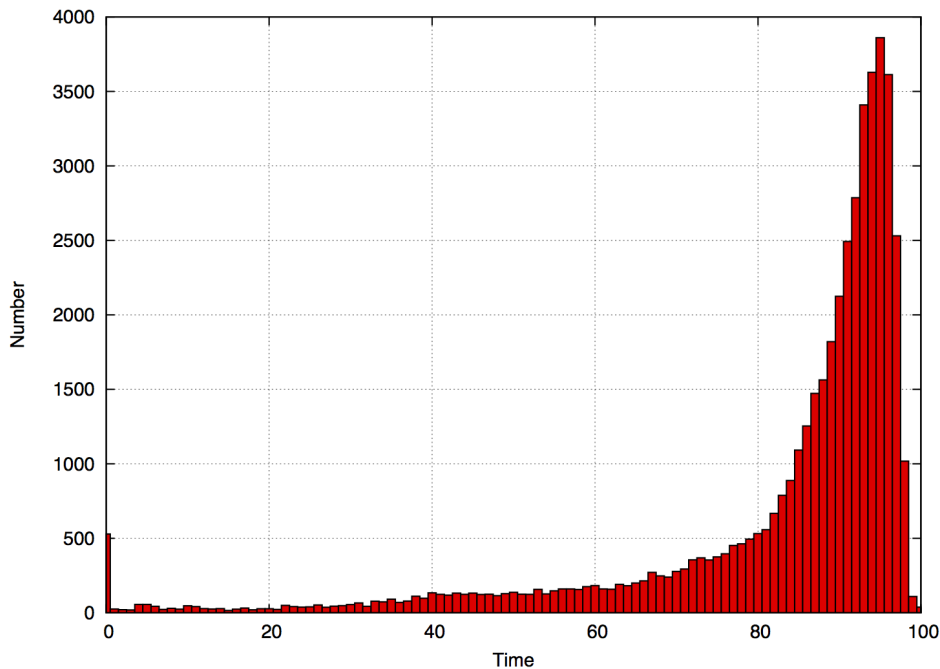
mcore jobs



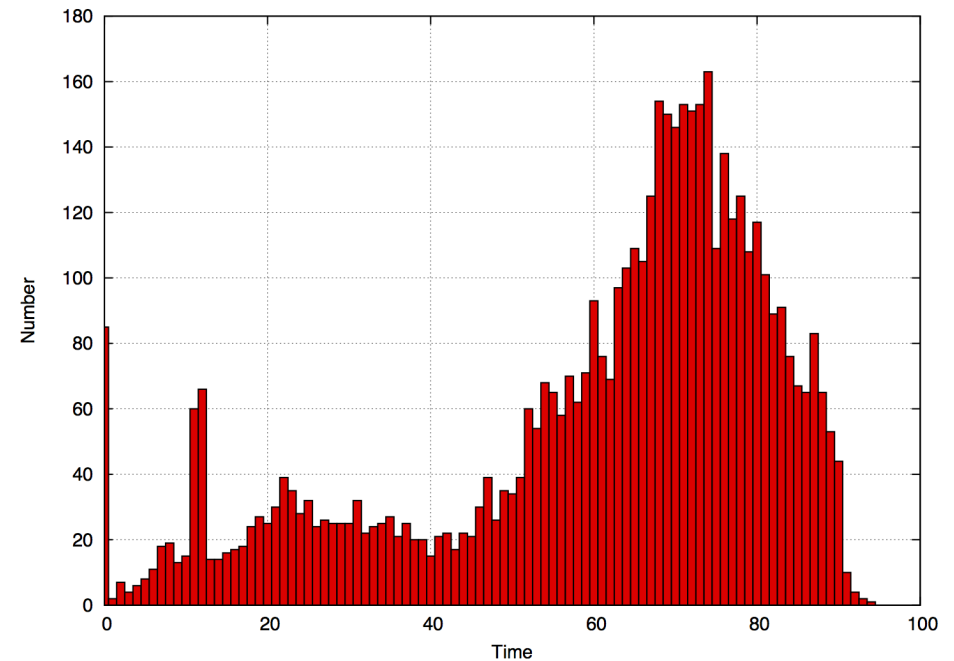
Results from CMS jobs

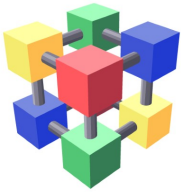
- **CPU efficiency results:** CMS multicore pilots, currently being filled with 8 single core jobs, presents additional CPU inefficiencies compared to single core jobs (report from RAL)
 - need better tuning of the internal mechanism

single core jobs: ~80%



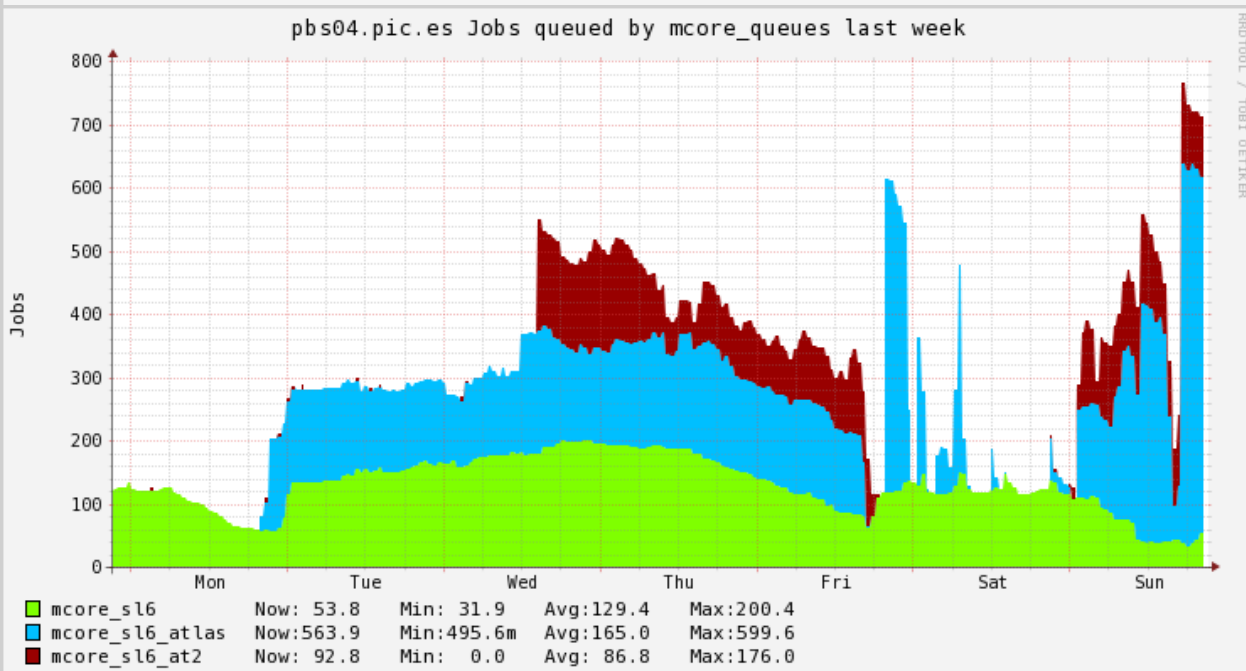
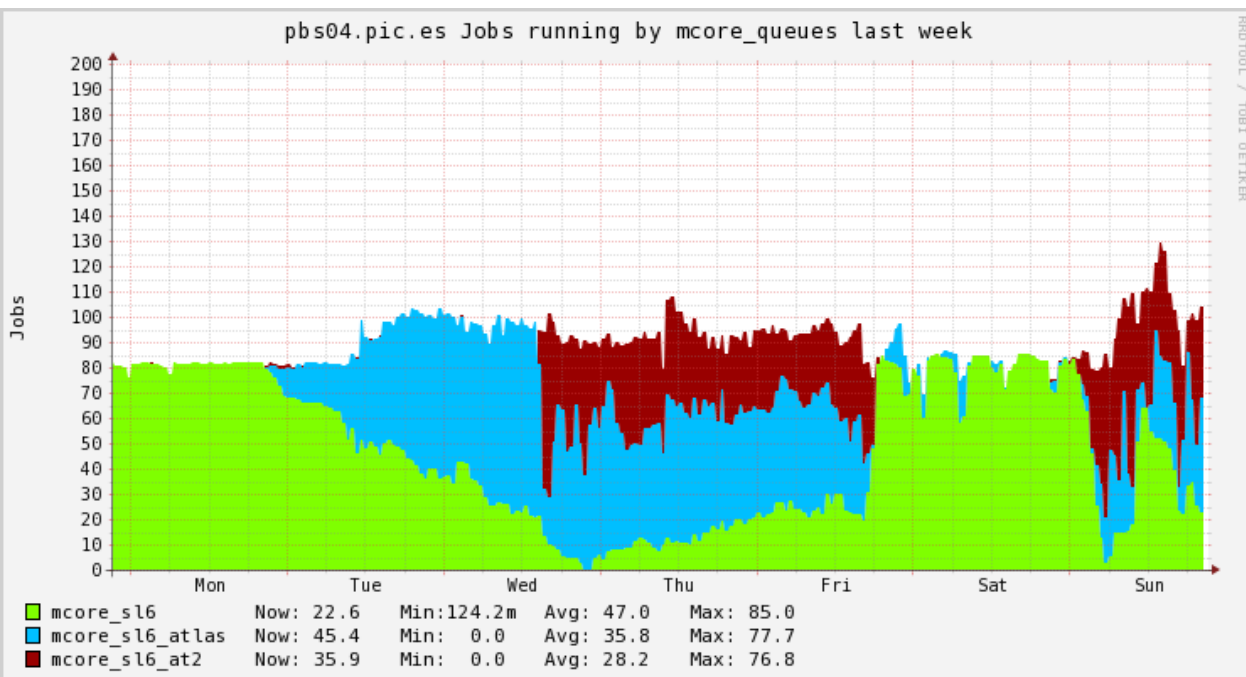
multi core jobs: ~60%

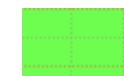




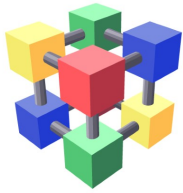


Fresh plots: combined submission

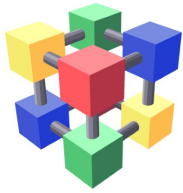
- Starting to see results from combined experience: e.g. **last week multicore jobs at PIC**



 CMS
 ATLAS T1
 ATLAS T2

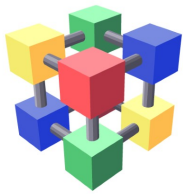


- **Conclusions and Outlook**



Conclusions

- Multicore jobs will be required to process and simulate data in the next LHC run
- The distribution and scheduling of multicore jobs across the WLCG is a problem in itself:
 - multitude of sites with diverse batch system technologies
 - different job submission models for each experiment (ATLAS and CMS)
- The WLCG multicore deployment task force has the mandate of coordinating these activities to make sure they can work together
- Results show that, since backfilling is not currently available, multicore slots have to be conserved to avoid unnecessary draining resulting in CPU wastage
 - A dynamic partitioning of the resources into separated WN pools is emerging as a viable solution
 - Big thanks NIKHEF (Jeff Templon) for the mcfloat tool!
 - **Torque/maui sites are encouraged to try this strategy**
- **Starting to get promising results of the combined CMS+ATLAS experience**



Outlook

- The immediate objective is to continue **testing both submission models (CMS and ATLAS) in shared environments at real scale**
 - Evaluate their compatibility
 - Analyze performance dependency on size of each site, the actual mixture of single core / multicore jobs, if the site is dedicated mainly to HEP or not, the actual batch system capabilities and its particular tuning, etc.
- For these tests, the ideal place are sites supporting both experiments with a diversity of batch systems to study.
 - CMS and ATLAS just started to send multicore jobs concurrently to shared Tier1s
- Multicore support result from the **interaction** of the VO submission models with local batch system capabilities and scheduler tuning
 - **It is and iterative process**, feedback exchange will be needed: **sites ↔ Vos**
- **Major milestone, to be ready by end of October, looks feasible**