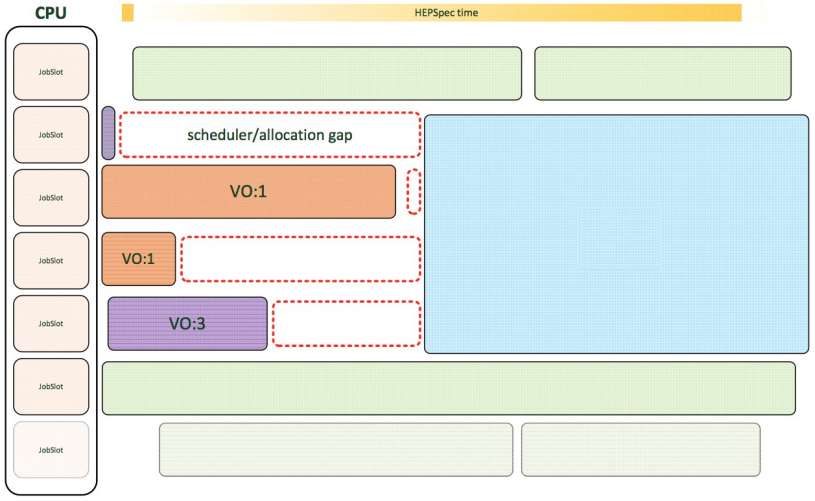
First Phase Wrap up

Alessandra Forti
Antonio Perez-Calero Yzquierdo
08 April 2014

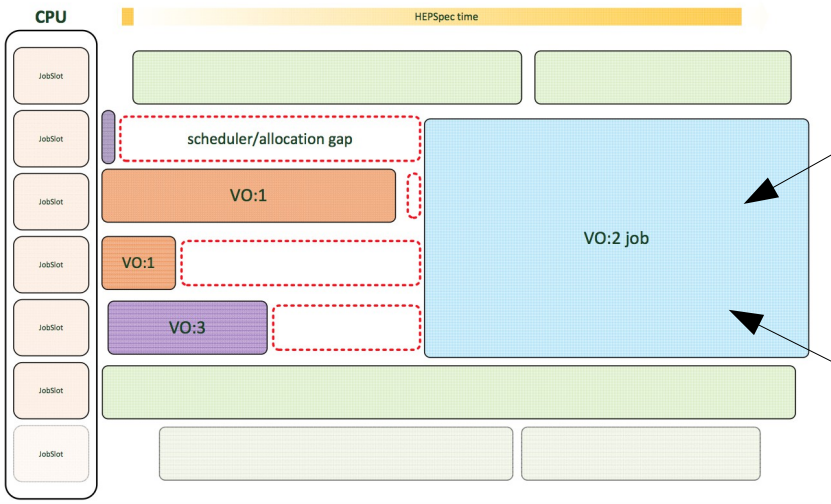
Multicore scheduling

- Single core and multicore jobs will have to coexist in the majority of sites.
  - To allocate non dedicated resources for multicore jobs draining is required.
  - As the majority of sites is currently configured without backfilling or reservations single core jobs with higher priority tend to occupy the freed slots making draining a painful and wasteful process until a sufficiently big number of slot is freed and a multicore job is at the top of the queue.
- Creating mcore slots
- Conserving mcore slots

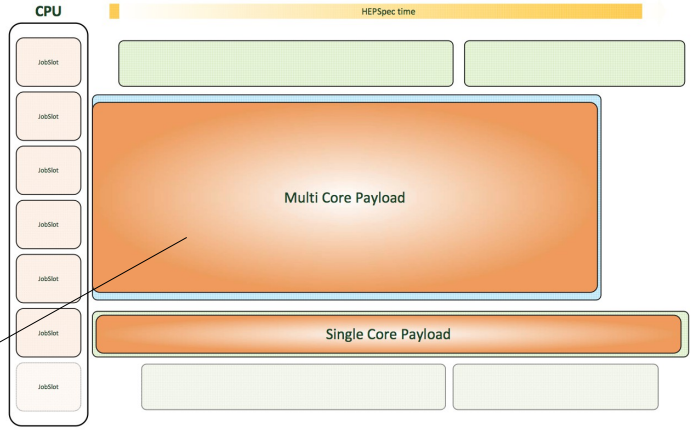


Atlas model

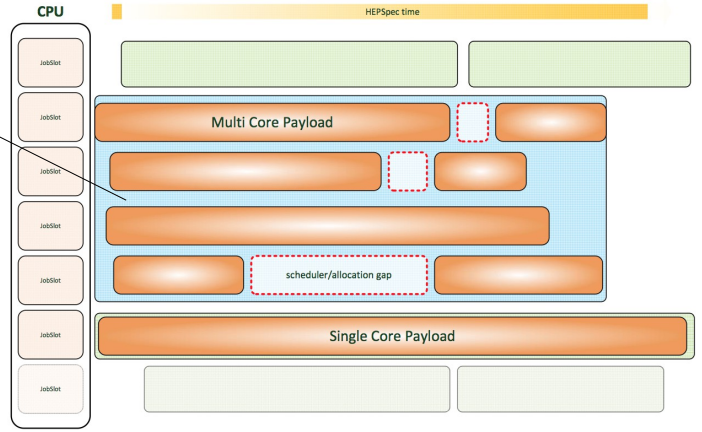
Atlas model considers the scheduling a site problem and prefers to submit both single core and multicore.



Inside a scheduler



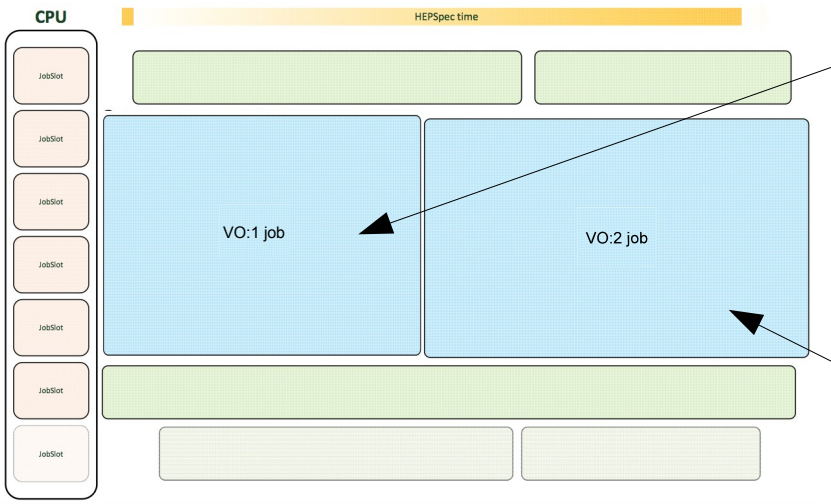
Atlas



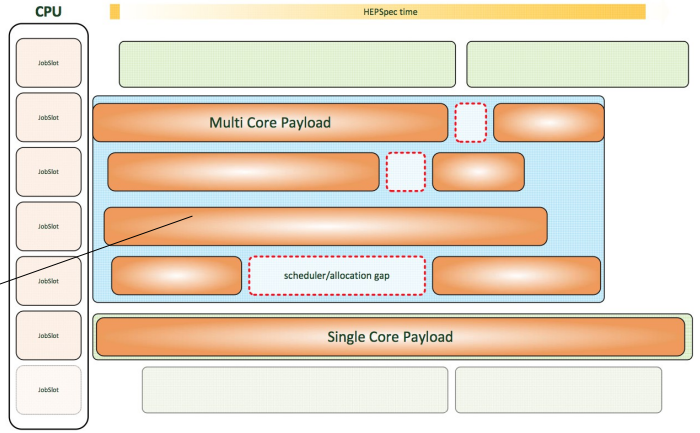
CMS

CMS model

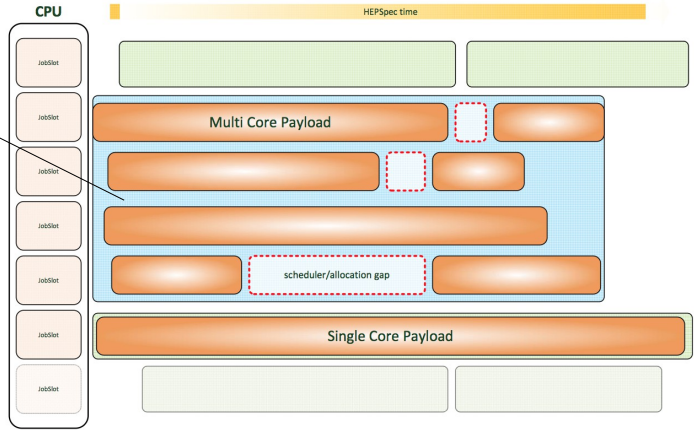
CMS model would have all Vos agree on a single pilot size. The aim is to preserve as long as possible the slots that have been assigned.



Inside a scheduler



Atlas



CMS

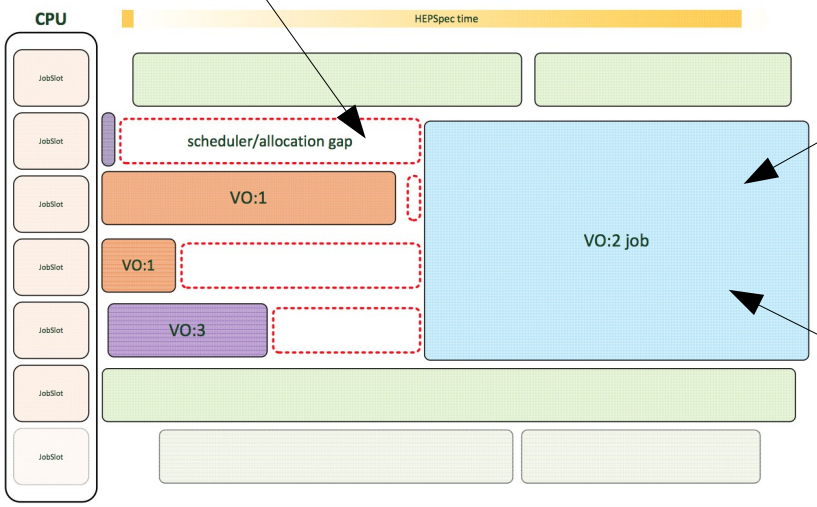
Backfilling

- 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
  - Most batch system are designed to do this
    - job request 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
    - job running times estimates, so that the scheduler can make a decision on whether it should run this job in that hole or not.

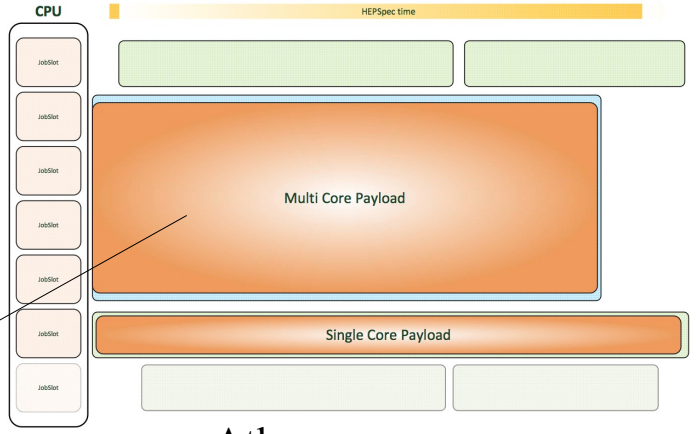
Functionality	Torque/Maui	SLURM	HTCondor	USGE/OSGE	Son of GE	LSF
Efficient Backfilling	tunable	tunable	not out-of-the-box, but similar behaviour can probably be configured	yes	yes	yes

Most sites would like....

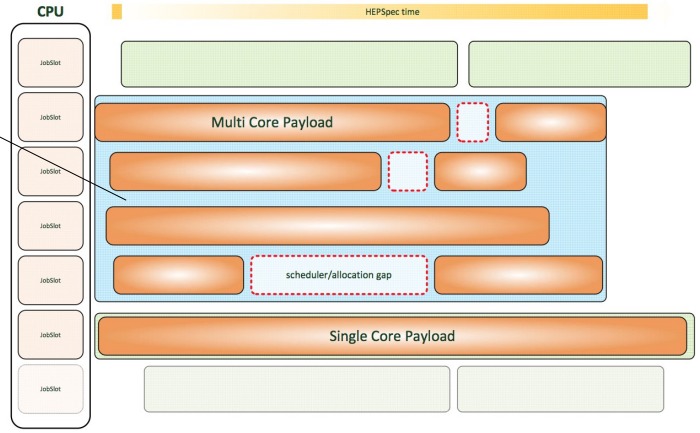
Without walltime high entropy of jobs so far was enough to fix the problem. Multicore require more organisation and backfilling with a guesstimate of walltime is needed to reduce gaps.



Inside a scheduler



Atlas



CMS

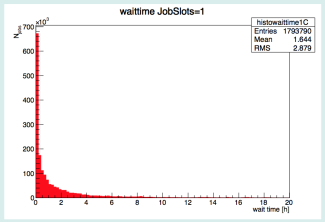
Reasons why there is no walltime

- Inherent to the jobs themselves, as the instantaneous luminosity and pile-up determine the complexity of events and thus the job running time.
  - This is different for analysis, MC production and data reconstruction/reprocessing.
  - There are mitigating tools in both experiments
- 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 difference between the fastest and slowest machines at a given site is not larger than 15-20%.
- The main middleware never really worked.
  - At most sites it doesn't pass the arguments to the batch system

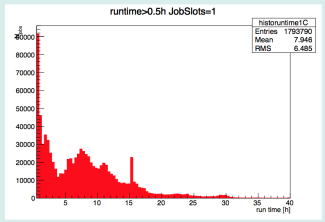
So far....

KIT: SingleCore Statistics 2014.Jan

wait time

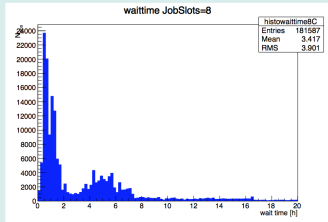


wall time (queue length=120h)(zoom 40h)

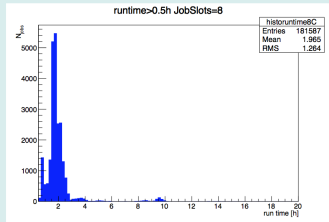


KIT: MCore Statistics 2014.Jan

wait time



wall time (queue length=120h)(zoom 20h)



- Longer waiting times while draining combined with short jobs
- Short jobs (empty pilots included) are disruptive because they don't exploit the slots freed.

Cancelled all draining worker nodes

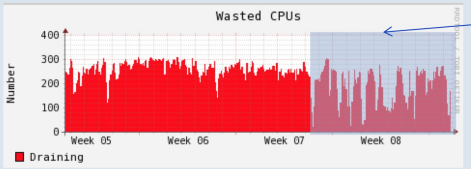
Conclusion:

Need to drain slots continuously to maintain number of running multi-core jobs

Need to drain constantly

Added monitoring of wasted CPUs due to draining

- Past month



Attempting to reduce wasted resources (next slide)

- We an clearly see the wastage - it's not hidden within a multi-core pilot running a mixture of single & multi-core jobs

- Wavelike submission most disruptive. Waste of CPU affected by submission patterns.

T. Hartmann, A.Lahiff plots

Partitioning

- The solution most sites have gone for is dynamic partitioning their clusters and limiting the number of draining cores at the time.
  - FZK have batch system native solution
  - Nikhef and RAL creatively (adding their own scripts).
- Partitioning allow single and multicore to cohexist without trampling on each other and still allowing fair shares to work.
  - Priorities maybe become secondaries but then so they do with backfilling

In Jeff words

- Separate pool : avoid the ‘ops job’ (or other higher prio job) takes 1 of my 8 slots and destroys ‘mc slot’
- Floating pool boundary w/ policies for filling and draining the tank:
  - Avoid too many empty slots during filling
  - Avoid empty slots if supply of mc jobs consistently (10+ minutes) dries up
- Protect against short stops

Is it the right way?

- It is a way forward.
  - It reduces the Four Apocalypse Horsemen power
    - Draining, short jobs, waves and no walltime
  - It accomodates both Atlas and CMS models
    - without dedicating resources to multicore
  - It is native to a number of batch systems
    - Maui sites may benefit from Nikhef scripts

Still

- Jobs longer than 2h, constant job submission and an estimate of the walltime should be an experiment priority because they would reduce the problem further, resolving also those few tenths or hundreds constantly draining cpus per site
  - which may look little when talking about 1-2% but they are still a waste considering shrinking funding everywhere.

Next phase

- The immediate objective is to test both models in a shared environment.
  - Find out if they can work together and how the global performance depends on the size of the site, the actual mixture of single core / multicore jobs (or pilots), if the site is dedicated mainly to HEP or not, the actual batch system capabilities and its particular tuning, etc.
    - CMS is starting to test more widely
    - More sites should give a go with the recipes exposed so far
- Should still work on wallclock time
  - ATLAS more dynamic queues and getting blah scripts for sites