Multicore

Alessandra Forti, Antonio Perez-Calero Yzquierdo
Thomas Hartmann, Manfred Alef, Andrew Lahiff,
Jeff Templon, Stefano Dal Pra
On behalf of the WLCG Multicore Task Force
CHEP, Okinawa
14 April 2015
Layout

- WLCG Task Force
- Scheduling Problem
- CMS&ATLAS models
- Initial observations
- Keeping the slots alive
- Range of options
- Dynamic partitioning
- Dynamic scheduling
- Site status
- Conclusions
WLCG Task Force

- Running multicore a long standing problem in WLCG
  - 2 experiments (different philosophies)
  - 170 sites different sizes
    - 5 batch systems + versioning
    - 3 CE flavours
    - Other supported VOs
- The objective of the WLCG Task Force is to
  - Find a set of easy to implement recommendations to schedule multicore without waisting resources
    - Batch system capability, experiments approach
  - Get the sites to run multicore
Scheduling problem

- **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!

- **Key Problems:**
  - Create mcore slots
  - Conserve mcore slots
  - Reduce draining vs ability to run mcore effectively
Experiments submission

- CMS move the scheduling within the pilot
  - Predictability
  - Shared sites still have single core to handle

- ATLAS: mcore and score in parallel with 1 payload per pilot and let the scheduler do the job.
  - Entropy
  - Predictability still helps
    - Backfilling not an option yet
Early observations

- Multicore require continuous draining of slots
  - Reduce the number of draining slots at the time
- Longer waiting times for multicore jobs
  - Sometimes not running for days
- Short jobs (<6h)
  - Disruptive because they don't exploit the slots freed.
- Long jobs (>24h)
  - Disruptive at shared sites
- Bursty submission most disruptive. Waste of CPU affected by submission patterns.
  - Disruptive whatever the solution
Keep the mcore slots alive

• Mixture of entropy and predictability
  • Experiments:
    • Continuous and stable supply of multicore jobs
    • Agreed common slot size at each site (default 8)
    • Avoid bursty submission patterns, which force the system to continue and re-adjust the level of draining
    • Avoid too short jobs or too long at non-dedicated sites
  • Sites
    • Allocate multicore jobs to multicore slots
      • Instead of single core jobs disrupting the drain process.
    • Rank/prioritise multicore over single core
    • Limit the number of slots that can be drained at the time
Range of options

• Treated in the TF
  1. Dynamic partitioning (Torque/LSF)
  2. Dynamic scheduling preferential mcore treatment and adaptable N of drained slots (HTcondor)
  3. Dynamic scheduling capacity to limit N of drained slots (SGE)

• Some other sites
  1. Static partitioning
     • Some dedicated sites with inflexible BS still use this.
  2. Dynamic scheduling preferential mcore treatment
     • No way to limit the number of draining slots
  3. Dynamic scheduling with no adjustments
     • All the problems described and no benefits at all!!
Dynamic Partitioning
(Torque/LSF)

- Separate pools: avoid other higher priority jobs taking 1 of the 8 slots and destroy the ‘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 dries up
- Protect against short stops
Dynamic scheduling (HTcondor)

- GROUP_SORT_EXPR to evaluate mcore before score
- Enabled defrag daemon
  - Pick WN in order of how many 8-slots can be freed
  - WNs can run both score and mcore at the same time
- Cron to adjust number of drained slots to workload
  - Adjust condor config DEFRAG_MAX_CONCURRENT_DRAINING

2.4%
0.4%
Dynamic Scheduling (SGE)

- Goal: minimize waste of resources by limiting draining
  - Create a PE (Parallel Environment)
  - Max number of jobs that can be considered for draining
    - max_reservation_set
      - ~10=0.5-1% degradation
      - ~20=1-1.5% degradation
  - -R y option to enable reservations
  - Relies on experiments to rank/prioritize their workload
    - No extra queue
    - No partitions
    - WNs can run score and mcore
Passing Parameters

• Backfilling is the traditional BS way to minimize waste
  • Requires jobs to pass the walltime at submission
  • Work ongoing on passing parameters to the BS in the TF
    • Concerning only ATLAS for now
    • Not only walltime but also memory
      • cgroups required to handle memory properly
      • Not all BS integrated with cgroups
        • Torque, SoGE, UGE <8.3.1, LSF<9.1.1
      • https://twiki.cern.ch/twiki/bin/view/LCG/BSPassingParameters
Sites status

• 85% of ATLAS sites have MCORE enabled
  • Still going through optimization
  • Reached 40% of resources, 50% slots in March.

• CMS priority for 2015 is multicore prompt data reconstruction which requires T0 plus 50% of T1 CPU resources.
  • All CMS T1s support multicore and target of 50% of T1 CPUs has been achieved.
  • T2s still on voluntary basis
Conclusions

- Quite few people put a lot of work and some creativity in solving this long standing problem both on the sites part and the experiments.

- There is still ongoing work
  - Looking at related high memory jobs scheduling
  - Passing parameters to the batch system

- Infrastructure to make it work is there
  - Infact it is currently already working
    - Needs fine tuning
Backup slides
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
- **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.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Torque/MAUI</th>
<th>SLURM</th>
<th>HTCondor</th>
<th>USGE/OSGE</th>
<th>Son of GE</th>
<th>LSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient Backfilling</td>
<td>tunable</td>
<td>tunable</td>
<td>not out-of-the-box, but similar behaviour can probably be configured</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Backfilling
Reasons why there is no walltime (yet)

• 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 most used CE type it require a standardization of the scripts to pass parameters to the batch system.
  • The TF has taken this on board