Scheduling multicore workload on shared multipurpose clusters

A. Perez-Calero Yzquierdo, J.Flix Molina, C. Acosta Silva (PIC)
A. Forti (Manchester)
R. Starink, Jeff Templon (Nikhef)

2015.04.14

• OUTLINE:
  • Intro scheduling theory
  • Background of multicore scheduling
  • How mcfloat works
  • Performance of mcfloat
  • How to tune & coupling with theory
Scheduling Theory 101

- A/S/C
  - A arrival time distribution
  - S size of jobs
  - C number of servers (i.e., worker nodes)
- Some are solved e.g., M/D/k
  - Poisson arrival time dist (M = Markov), deterministic size of jobs (D), k WNs
- G/G/k is not solved
- Best you can do is statistics
Statistics reminder

What wins in statistics, are configurations (ways to partition a system) in which there are many ways (permutations) to achieve the desired result.

Think about rolling 100 dice … answer will nearly always be between 300 and 400, even though a roll of “100” is just as likely as any particular roll of 350. Just many, many more ways to get 350 than to get 100.

Try “distribution 100 dice” on Wolfram Alpha
Try other numbers than 100 …
Multicore in Practice: Boundary Conditions

- Important customers want multicore
- No easy solution @ Nikhef:
  - Usually >7 groups active (also important)
  - Almost never empty (99% used)
  - Funding is for shared facility: cannot dedicate slots statically
- Typical jobs on system don’t allow scheduler to progress on multicore problem (ie no backfilling)
Doing it without scheduler: draining

8-core nodes

Running jobs

Fastest: 8 hr drain

Average: 21 hr drain

Slowest: 34 hr drain

Job completion time (hours into the future)

Real data, production cluster – 8 groups active, Snapshot 31 march 2015 15:42:02 CEST

Can’t do better than statistics anyway

Can’t do better than statistics anyway
Multicore slot conservation with mcfloat

Multicore slot conservation can be achieved with **dynamic partitioning** of site resources: implemented by **mcfloat tool** for Torque/Maui sites

**Principles:**

- 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

---

![Diagram](image)
A week at Nikhef

Note “unused” here is unused fraction of multicore pool. Also note: pool includes both draining and drained nodes.

Multicore jobs

Other jobs still running on draining nodes

Period in which all nodes 100% drained
Mcfloat performance at PIC

A week of multicore jobs running at PIC

- Continuous ramp up of multicore resources being offered by the site as a response to user pressure
- Robust with respect to individual users bursty job submission patterns

unused slots at 2% over period
Dynamic partition works

- Mcfloat works
- Rest: How to optimize
  - Operation of dynamic multicore pool
  - Acquisition of running multicore slots
Examine job distributions: what do they tell us?

Job wall time distribution (main pool)

Job remaining time distribution (main pool)

Time distributions on shared cluster – 8 groups active
Snapshot 31 march 2015 15:42:02 CEST
Prob distribution: e.g. 23% chance that job picked at random exits within 4 hours

Mult x 8: dist for 8-core nodes. For random 8-core node, within four hours, 1.8 slots will vacate, within 8 hours 3.7 slots, etc.

For 12-core nodes

For 32-core nodes
Average time-to-start from real data

Recall reminder from statistics beginning of talk!!!

White: Wasted core hours

<table>
<thead>
<tr>
<th>Job size</th>
<th>t (hr) 8c nod</th>
<th>time 12c nod</th>
<th>time 32c nod</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 cores</td>
<td>21</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>4 cores</td>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

4-core jobs win twice: Start faster and waste fewer slots.
The switch at Nikhef

8 core jobs

4 core jobs
Conclusions

- Dynamic partitioning with simple algorithm works well in practice
- Validated at PIC and Manchester
  - Since used at other sites as reported at HEPiX
- For fast growth & little waste in your pool, employ combinatorics:
  - Run jobs with as few cores as possible
  - More cores per node is better
Background info & exercises (if you like)

• Think about doodle poll difficulty: 8 people vs 4 of 8

• Visit the [wikipedia page](http://www.wikipedia.org) on queueing theory

• Visit the Wolfram alpha [dice roll page](http://www.wolframalpha.com) … try 1, 2, 5, 100 dice and see what happens to the probability distribution