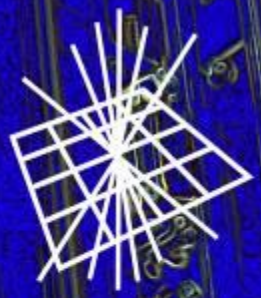


GridPP

UK Computing for Particle Physics

Multicore at RAL

HEP SYSMAN, Jan 2014

What we used to do

- Whole-node queue configured on CREAM CE
  - Submits to whole node queue in Torque, which has:

    ```
    resources_default.nodes = 1:ppn=8
    ```

    1 node; 8 virtual processors per node
  - Maui config

    ```
    JOBFLAGS=DEDICATEDNODE
    ```
- Used dedicated whole-node (8-core) worker nodes

```
# Whole node queue
SRCFG[wholenode]  STARTTIME=0:00:00 ENDTIME=24:00:00
SRCFG[wholenode]  PERIOD=INFINITY
SRCFG[wholenode]  CLASSLIST=gridWN
SRCFG[wholenode]  HOSTLIST=lcg0990,lcg0991,lcg0992,lcg0993,lcg0994
```

- More information:

  http://www.gridpp.rl.ac.uk/blog/tag/whole-node-jobs/

What we used to do

- Problems
  - CREAM CE: nothing stops single core jobs from being submitted to & running on the whole node queue
    - Mainly problem with glite-WMS jobs
    - Therefore limited whole node queue only to VOs which asked for it
    - Still issues with LHCb SUM tests ending up on the whole node queue
      - Jobs may be queued for a long time – not good for SUM tests
    - Is there any way around this?
  - Partitioned resources
    - Need to manually move resources between single-core & multi-core jobs
  - Maui wasn't good at scheduling whole-node jobs
    - Even with dedicated nodes!
    - Solution: wrote our own scheduler (the "Kelly Scheduler"), running in parallel with Maui, which scheduled whole-node jobs

Current status

The situation after moving from Torque/Maui to HTCondor:

- CREAM CEs
  - Initially had 8-core queues
  - Removed them because ALICE/LHCb were submitting normal single core SUM test jobs to them
- ARC CEs
  - Ours configured to have only a single queue
  - Any VO can run multicore jobs if they want to
    - Just have to request > 1 CPU. E.g. add to XRSL:

      `(count=8)`
    - Any number of CPUs can be requested (2, 4, 8, 31, …)
      - But a job requesting 402 CPUs, for example, is unlikely to run ☺

Current status

- HTCondor configuration
  - All WNs configured to have partitionable slots
  - Each WN has a *single partitionable slot* with a fixed set of resources (cores, memory, disk, swap, …)
  - These resources can then be divided up as necessary for jobs
    - Dynamic slots created from the partitionable slot
    - When dynamic slots exit, merge back into the patitionable slot
    - When any single resource is used up (e.g. memory), no more dynamic slots can be created
  - Enables us to run jobs with different memory requirements, as well as jobs requiring different numbers of cores
  - Configuration:

```
NUM_SLOTS = 1
SLOT_TYPE_1 = cpus=100%,mem=100%,auto
NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1_PARTITIONABLE = TRUE
```

Current status

- Example WN

```
# condor_status lcg1560.gridpp.rl.ac.uk -autoformat Name State Activity SlotType Memory Cpus
slot1@lcg1560.gridpp.rl.ac.uk     Unclaimed  Idle   Partitionable  101500  0  ← partitionable slot
slot1_10@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         2500    1
slot1_11@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         4000    1
slot1_12@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_13@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         4000    1
slot1_14@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_15@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_16@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_17@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         4000    1
slot1_18@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         2500    1
slot1_19@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_1@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_20@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         4000    1
slot1_21@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_22@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         4000    1
slot1_23@lcg1560.gridpp.rl.ac.uk  Claimed    Busy   Dynamic         3000    1
slot1_2@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_3@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_4@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_5@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_6@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         4000    1
slot1_7@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         3000    1
slot1_8@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         4000    1
slot1_9@lcg1560.gridpp.rl.ac.uk   Claimed    Busy   Dynamic         2000    8  ← multicore job
```

Current status

- If lots of single core jobs running, how does a multicore job start?
- condor_defrag daemon
  - Finds worker nodes to drain
  - Configuration parameters, including:
    - How often to run
    - Max number of “whole” machines *(we’re using 300 currently)*
    - Max concurrent draining machines *(we’re using 60 currently)*
    - Expression that specifies which machines to drain
    - Expression that specifies when to cancel draining
    - Expression that specifies which machines are already “whole” machines
    - Expression that specifies which machines are more desirable to drain
  - Currently fairly simple
    - E.g. doesn’t know anything about demand (idle multicore jobs)

Current status

- Current setup
  - Added accounting groups (fairshares) for ATLAS multicore jobs

```
group_ATLAS.atlas
group_ATLAS.atlas_pilot
group_ATLAS.atlas_pilot_multicore
group_ATLAS.prodatls
group_ATLAS.prodatls_multicore
```

  - Definition of "whole" machines (currently require 8 cores only drained)

```
DEFRAG_WHOLE_MACHINE_EXPR = ((Cpus == TotalCpus) || (Cpus >= 8))
```

default

Entire machines drained, but draining is cancelled when a minimum of 8 cores become available

  - Changed expression that specifies which machines are more desirable to drain
    - Default:

```
DEFRAG_RANK = -ExpectedMachineGracefulDrainingBadput
```

The job runtime in cpu-seconds that would be lost if graceful draining were initiated at the time the machine's ad was published. Assumes jobs will run to their walltime limit.

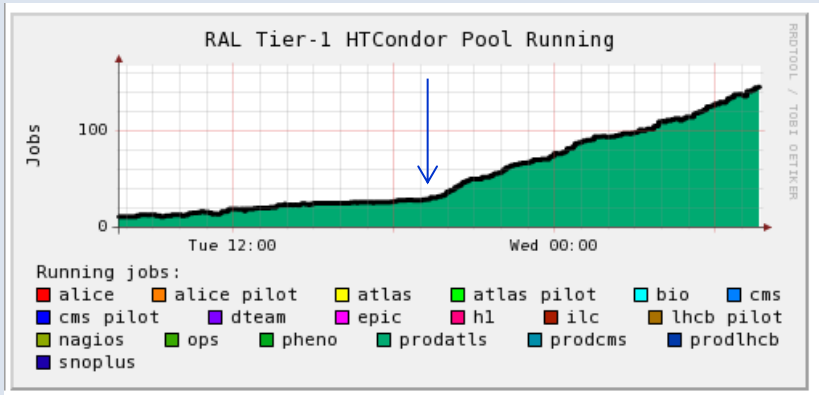
    - New (last week):

```
DEFRAG_RANK = ifThenElse(Cpus >= 8, -10, (TotalCpus - Cpus)/(8.0 - Cpus))
```

    - Made a big improvement
      - Previously only older 8-core machines were selected for draining
      - Now machines with the most numbers of cores are selected for draining

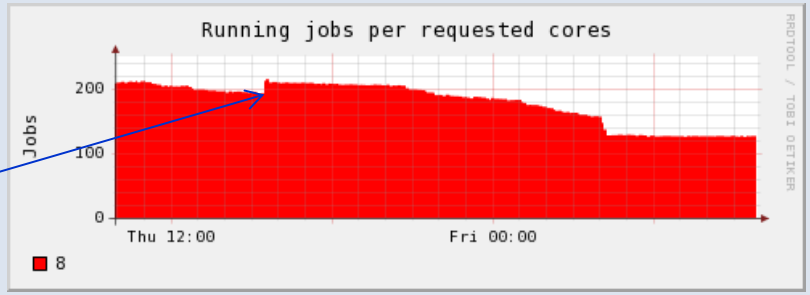
Current status

- Effect of change to DEFRAG_RANK



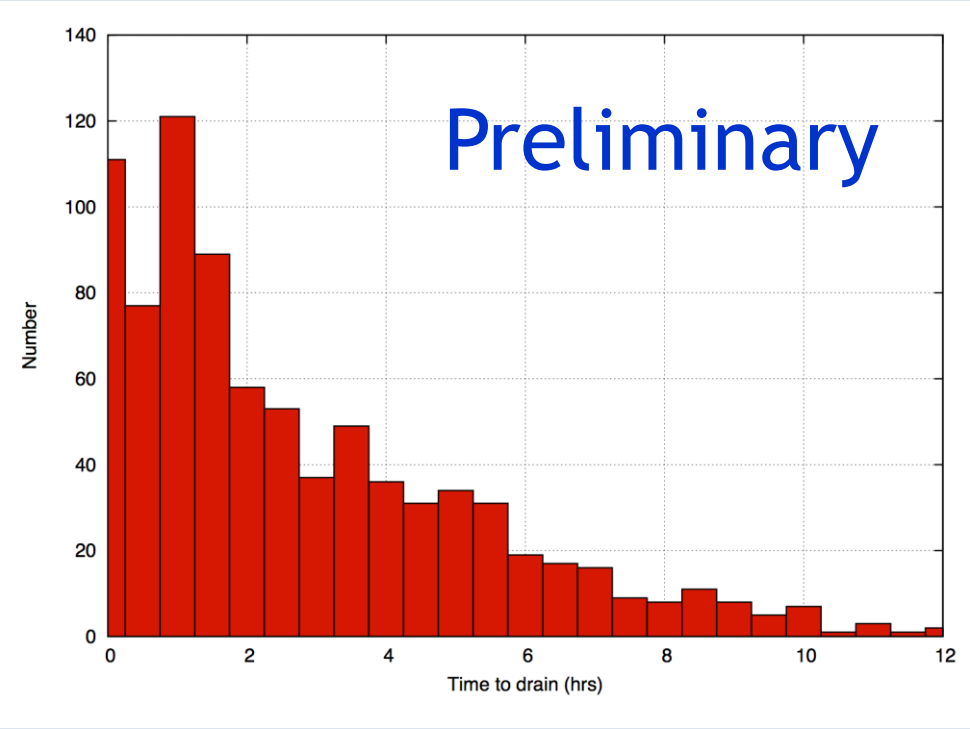
- Effect of switching off condor_defrag
  - Will number of running multicore jobs quickly reduce to zero?
  - Number of running multicore jobs decreased slowly, then became constant

Cancelled all draining worker nodes



Current status

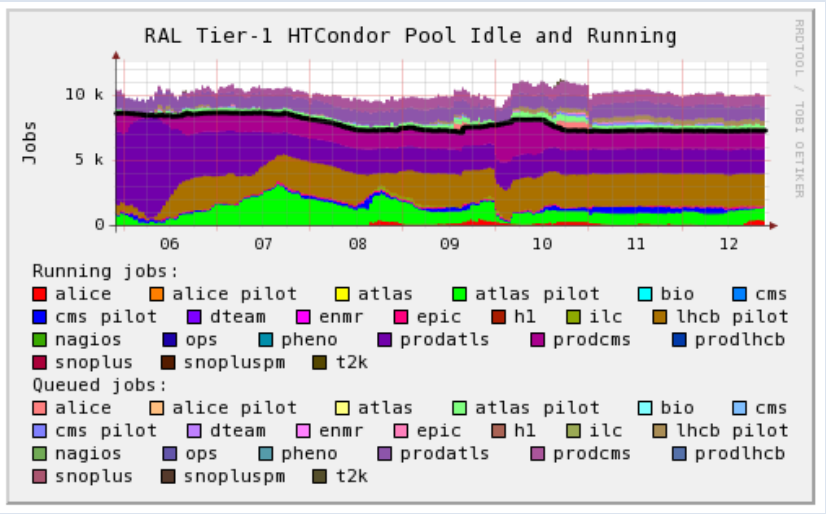
- Time to drain worker nodes
  - Data extracted from log files on WNs (/var/log/condor/StartLog)



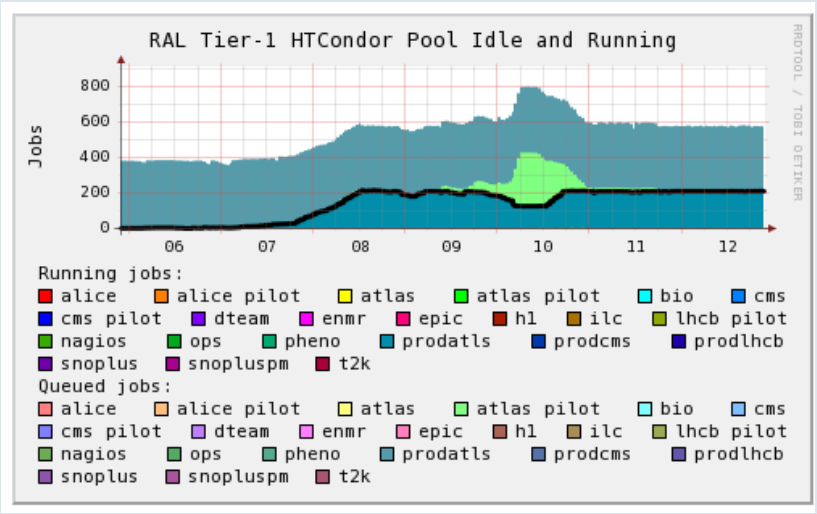
Current status

Past week

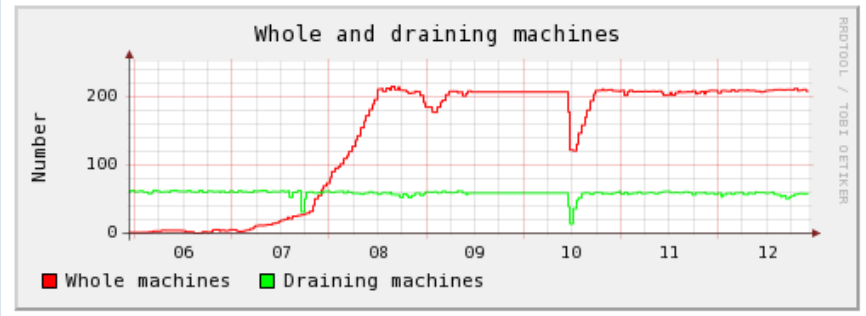
Jobs idle/running



Multicore jobs idle/running



Number of running ATLAS multicore jobs has been relatively constant due to fairshares (except when condor_defrag was switched off)



Future plans

- Concerns
  - Resources wasted while slots are draining
    - Currently always a fixed number of WNs are draining (60)
  - Can this be improved?
    - Simple option perhaps: cron which changes defrag parameters
  - Do we need to try to make multicore slots “sticky”?
    - i.e. when a multicore job runs, prefer to run another multicore job rather than 8 single core jobs
- Multiple multicore jobs per WN
  - Currently will have at most 1 multicore job per WN (unless not many single core jobs)
  - Modifications to our condor_defrag configuration necessary to allow this