HTCondor at the RAL Tier-1

Andrew Lahiff,
Alastair Dewhurst, John Kelly, Ian Collier, James Adams
STFC Rutherford Appleton Laboratory

HTCondor Week 2014
• Overview of HTCondor at RAL
• Monitoring
• Multi-core jobs
• Dynamically-provisioned worker nodes
• RAL is a Tier-1 for all 4 LHC experiments
  – Provide computing & disk resources, and tape for custodial storage of data
  – In terms of Tier-1 computing requirements, RAL provides
    • 2% ALICE
    • 13% ATLAS
    • 8% CMS
    • 32% LHCb
  – Also support ~12 non-LHC experiments, including non-HEP
• Computing resources
  – 784 worker nodes, over 14K cores
  – Generally have 40-60K jobs submitted per day
Migration to HTCondor

- Torque/Maui had been used for many years
  - Many issues
  - Severity & number of problems increased as size of farm increased
- Migration
  - 2012 Aug: Started evaluating alternatives to Torque/Maui (LSF, Grid Engine, Torque 4, HTCondor, SLURM)
  - 2013 Jun: Began testing HTCondor with ATLAS & CMS
  - 2013 Aug: Choice of HTCondor approved by management
  - 2013 Sep: HTCondor declared production service
    - Moved 50% of pledged CPU resources to HTCondor
  - 2013 Nov: Migrated remaining resources to HTCondor
Experience so far

- **Experience**
  - Very stable operation
    - Staff don’t need to spend all their time fire-fighting problems
  - Job start rate much higher than Torque/Maui, even when throttled
    - Farm utilization much better
  - Very good support
• **Version**
  – Currently 8.0.6
  – Trying to stay up to date with the latest stable release

• **Features**
  – Partitionable slots
  – Hierarchical accounting groups
  – HA central managers
  – PID namespaces
  – Python API
  – condor_gangliad

• **In progress**
  – CPU affinity being phased in
  – cgroups has been tested, probably will be phased-in next
• All job submission to RAL is via the Grid
  – No local users

• Currently have 5 CEs, schedd on each:
  – 2 CREAM CEs
  – 3 ARC CEs

• CREAM doesn’t currently support HTCondor
  – We developed the missing functionality ourselves
  – Will feed this back so that it can be included in an official release

• ARC better
  – But didn’t originally handle partitionable slots, passing CPU/memory requirements to HTCondor, …
  – We wrote lots of patches, all included in upcoming 4.1.0 release

• Will make it easier for more European sites to move to HTCondor
• Increasing usage of HTCondor in WLCG sites in the UK
  – 2013-04-01: None
  – 2014-04-01: RAL Tier-1, RAL Tier-2, Bristol, Oxford (in progress)

• The future
  – 7 sites currently running Torque/Maui
  – Considering moving to HTCondor or will move if others do
Monitoring
Jobs monitoring

• Useful to store details about completed jobs in a database
• What we currently do
  – Nightly cron reads HTCondor history files, inserts data into MySQL
• Problems
  – Currently only use subset of content of job ClassAds
    • Could try to put in everything
    • What happens then if jobs have new attributes? Modify DB table?
  – Experience with similar database for Torque
    • As database grew in size, queries took longer & longer
    • Database tuning important
• Is there a better alternative?
CASTOR team at RAL have been testing Elasticsearch
  - Why not try using it with HTCondor?

Elasticsearch ELK stack
  - Logstash: parses log files
  - Elasticsearch: search & analyze data in real-time
  - Kibana: data visualization

Hardware setup
  - Test cluster of 13 servers (old diskservers & worker nodes)
    - But 3 servers could handle 16 GB of CASTOR logs per day

Adding HTCondor
  - Wrote config file for Logstash to enable history files to be parsed
  - Add Logstash to machines running schedds
**Jobs monitoring**

- Can see full job ClassAds
• Custom plots
  – E.g. completed jobs by schedd
• Custom dashboards
• **Benefits**
  – Easy to setup
    • Took less than a day to setup the initial cluster
  – Seems to be able to handle the load from HTCondor
    • For us (so far): < 1 GB, < 100K documents per day
  – Arbitrary queries
  – Queries are faster than using condor_history
  – Horizontal construction
    • Need more capacity? Just add more nodes
Multi-core jobs
Multi-core jobs

- Situation so far
  - ATLAS have been running multi-core jobs at RAL since November
  - CMS submitted a few test jobs, will submit more eventually
  - Interest so far only for multi-core jobs, not whole-node jobs
    - Only 8-core jobs

- Our aims
  - Fully dynamic
    - No manual partitioning of resources
  - Number of running multi-core jobs determined by group quotas
Multi-core jobs

• Defrag daemon
  – Essential to allow multi-core jobs to run
  – Want to drain 8 cores only. Changed:
    
    DEFRAG_WHOLE_MACHINE_EXPR = Cpus == TotalCpus && Offline=!True
    to
    DEFRAG_WHOLE_MACHINE_EXPR = (Cpus >= 8) && Offline=!True
  – Which machines more desirable to drain? Changed
    
    DEFRAG_RANK = -ExpectedMachineGracefulDrainingBadput
    to
    DEFRAG_RANK = ifThenElse(Cpus >= 8, -10, (TotalCpus - Cpus)/(8.0 - Cpus))
  – Why make this change?
    • With default DEFRAG_RANK, only older full 8-core WNs were being selected for draining
    • Now: (Number of slots that can be freed up)/(Number of needed cores)
      – For us this does a better job of finding the “best” worker nodes to drain
Multi-core jobs

• Effect of changing DEFRAG_RANK

- No change in the number of concurrent draining machines
- Rate in increase in number of running multi-core jobs much higher
• **Group quotas**
  – Added accounting groups for ATLAS and CMS multi-core jobs
  – Force accounting groups to be specified for jobs using `SUBMIT_EXPRS`
  • Easy to include groups for multi-core jobs
    ```
    AccountingGroup =
    ...
    ifThenElse(regexp("patl",Owner) && RequestCpus > 1, "group_ATLAS.prodats_multicore", \
    ifThenElse(regexp("patl",Owner), "group_ATLAS.prodats", \
    ...
    SUBMIT_EXPRS = \$(SUBMIT_EXPRS) AccountingGroup
    ```

• **Negotiator**
  – Modified `GROUP_SORT_EXPR` so that the order is:
    • High priority groups (Site Usability Monitor tests)
    • Multi-core groups
    • Remaining groups
  – Helps to ensure multi-core slots not lost too quickly
Multi-core jobs

• Defrag daemon issues
  – No knowledge of demand for multi-core jobs
    • Always drains the same number of nodes, irrespective of demand
    • Can result in large amount of wasted resources
  – Wrote simple cron script which adjusts defrag daemon config based on demand
    – Currently very simple, considers 3 cases:
      • Many idle multi-core jobs, few running multi-core jobs
        – Need aggressive draining
      • Many idle multi-core jobs, many running multi-core jobs
        – Less aggressive draining
      • Otherwise
        – Very little draining
  – May need to make changes when other VOs start submitting multi-core jobs in bulk
• Recent ATLAS activity

Running & idle multi-core jobs

Gaps in submission by ATLAS results in loss of multi-core slots.

Number of “whole” machines & draining machines

Data from condor_gangliad

Significantly reduced CPU wastage due to the cron
Multi-core jobs

- Other issues
  - Defrag daemon designed for whole-node, not multi-core
    - Won’t drain nodes already running multi-core jobs
    - Ideally may want to run multiple multi-core jobs per worker node
  - Would be good to be able to run “short” jobs while waiting for slots to become available for multi-core jobs
    - On other batch systems, backfill can do this

Time taken for 8 jobs to drain
- lots of opportunity to run short jobs
• **Next step: enabling “backfill”**
  – ARC CE adds custom attribute to jobs: JobTimeLimit
    • Can have knowledge of job run times
  – Defrag daemon drains worker nodes
    • Problem: machine can’t run any jobs at this time, including short jobs
  – **Alternative idea:**
    • Python script (run as a cron) which plays the same role as defrag daemon
      – But doesn’t actually drain machines
    • Have a custom attribute on all startds, e.g. NodeDrain
      – Change this instead
    • START expression set so that:
      – If NodeDrain false: allow any jobs to start
      – If NodeDrain true: allow only short jobs under certain conditions, e.g. for a limited time after “draining” started
    – Provided (some) VOs submit short jobs, should be able to reduce wasted resources due to draining
Dynamically-provisioned worker nodes
Private clouds at RAL

• Prototype cloud
  – StratusLab (based on OpenNebula)
  – iSCSi & LVM based persistent disk storage (18 TB)
  – 800 cores
  – No EC2 interface

• Production cloud
  – (Very) early stage of deployment
  – OpenNebula
  – 900 cores, 3.5 TB RAM, ~1 PB raw storage for Ceph

• Aims
  – Integrate with batch system, eventually without partitioned resources
  – First step: allow the batch system to expand into the cloud
    • Avoid running additional third-party and/or complex services
    • Use existing functionality in HTCondor as much as possible
    • Should be as simple as possible
• Use HTCondor’s existing power management features
  – Send appropriate offline ClassAd(s) to the collector
    • Hostname used is a random string
    • Represents a type of VM, rather than specific machines
  – condor_rooster
    • Provisions resources
    • Configured to run appropriate command to instantiate a VM
  – When there are idle jobs
    • Negotiator can match jobs to the offline ClassAds
    • condor_rooster daemon notices this match
      – Instantiates a VM
      – Image has HTCondor pre-installed & configured, can join the pool
  – HTCondor on the VM controls the VM’s lifetime
    • START expression
      – New jobs allowed to start only for a limited time after VM instantiated
    • HIBERNATE expression
      – VM is shutdown after machine has been idle for too long
Testing in production

- Initial test with production HTCondor pool
- Ran around 11,000 real jobs, including jobs from all LHC VOs
- Started with 4-core 12GB VMs, then changed to 3-core 9GB VMs
Summary

- Due to scalability problems with Torque/Maui, migrated to HTCondor last year
- We are happy with the choice we made based on our requirements
  - Confident that the functionality & scalability of HTCondor will meet our needs for the foreseeable future
- Multi-core jobs working well
  - Looking forward to more VOs submitting multi-core jobs
- Dynamically-provisioned worker nodes
  - Expect to have in production later this year
Thank you!