HTCondor at the RACF

SEAMLESSLY INTEGRATING MULTICORE JOBS AND OTHER DEVELOPMENTS

William Strecker-Kellogg
RHIC/ATLAS Computing Facility
Brookhaven National Laboratory
April 2014
RACF Overview

Main HTCondor pools
- PHENIX—12.2kCPU
- STAR—12.0kCPU
- ATLAS—13.1kCPU

STAR/PHENIX are RHIC detectors
- Loose federation of individual users

ATLAS—tightly controlled, subordinate to PANDA workflow management, strict structure

Smaller Experiments
- LBNE
- Dayabay
- LSST
Supporting Multicore

Priorities and Requirements

- Keep management workload at a minimum
  - No manual re-partitioning
  - Needs to be dynamic and/or partitionable
  - Support automatic demand-spillover
  - Need to retain group-quota system with accept-surplus feature
- Maximize throughput—minimize latency
  - Same goals as above
- Attempt to support future developments in same framework
  - More than just multicore
    - High-Memory already used in this context (with caveat)
- Principle of proportional pain
  - Okay to make multicore wait longer—but no starvation is allowed
Supporting Multicore

STEP 1: PARTITIONABLE SLOTS EVERYWHERE

Required change to monitoring
  - Job-count no longer correct metric for measuring occupancy

Minor script change with SlotID
  - Slot<n> → Slot<m>_<n>

Works with no side effects

STEP 2: POLICY CHANGES

Preemption is no longer possible
  - OK for now since not needed

Slot-Weight can only be CPUs
  - Needs to change in the future

Defragmentation is necessary
  - Detail next slide

Dedicated queues for now
Defragmentation Policy

DEFRAGMENTATION DAEMON

Start Defragmentation
  ◦ (PartitionableSlot && !Offline && TotalCpus > 12)

End Defragmentation
  ◦ (Cpus >= 10)

Rate: max 4/hr

KEY CHANGE: NEGOTIATOR POLICY

Default policy is breadth-first filling of equivalent machines
  ◦ (Kflops – SlotId)

Depth-first filling preserves continuous blocks longer
  ◦ (-Cpus)
Multicore in Dedicated Queues

Works well now, issues were resolved
- Allocation is handled by group quota and surplus-share

Dedicated queue per species of job
- Currently two—high-memory (6Gb) and 8-core
- Changing requirements require manual action
Multicore in common Queue

Works well, no starvation
- Lack of control over strict allocation of m. vs. n core jobs within queue

Not currently done in ATLAS
- Issue of allocation control is just part of reason why
- Structural and not likely to change soon
Fixing bugs in HTCondor

Last summer a period of intensive development/testing in collaboration with HTCondor team

- Built a VM testbed, rapid build & test of patches from HTCondor team
- Built new monitoring interface
- After many iterations had working config with Partitionable slots and Hierarchical Group Quotas with accept_surplus

Now it does!
Bugfix Testbed Details

Rapid build, test, deploy cycle from git patches
- Email patch
- Rebuild condor
- Run test-feeder

Job Feeder
- Defines groups in config-file with different random length-ranges and requirements
- Variable workload—keep N jobs idle in each queue

```
# Queue num_idle avg_runtime splay racf weight_string
`group_atlas.analysis.long` 10 1200 300 long 1,1,8,2,4
`group_atlas.analysis.short` 18 300 100 short 1,1,1,2
`group_atlas.prod.production` 49 900 300 prod 1,1,1,2,4,8
`group_atlas.prod.test` 2 200 160 proptest 1
`group_atlas.prod.mp` 7 300 100 mp8 16
`group_atlas.software` 2 120 20 sw 1,2
`group_grid` 6 300 120 grid 1,1,2
```
Current Multicore Status

Fully utilize PSlots
  ◦ All traditional nodes (x86_64) can have same config

  SLOT_TYPE_1 = 100%
  NUM_SLOTS = 1
  NUM_SLOTS_TYPE_1 = 1
  SLOT_TYPE_1_PARTITIONABLE = True
  SlotWeight=Cpus

Fix works perfectly when accept_surplus is on for any combination of groups

Major Limitation: SlotWeight=Cpus
  ◦ High-memory jobs can be accommodated by asking for more CPUs.
  ◦ Need ability to partition better and interact with global resource limits
  ◦ SlotWeight should be a configurable function of all consumable resources

Other Limitation: No Preemption
  ◦ Required to support opportunistic usage
Issue With High Memory Jobs

1...ok, 2...ok, 3...not ok!

General problem:
- Inefficiencies in heterogeneous jobs scheduling to granular resources
- Worse as you add dimensions: imagine GPSs, Disks, CoProcessors, etc...
Goals

Need all the same behavior as we have now regarding groups and accept_surplus

Want to be able to slice by any resource

Sane and configurable defaults/quantization of requests

Defragmentation inefficiencies should be kept to a minimum—we are mostly there already!

Overall we are something like ¾ of the way to our ideal configuration.
Problem of Weights and Costs

What does SlotWeight mean with heterogeneous resources?

- Job of administrator to determine how much to “charge” for each requested resources
  - E.g. (cpus + 1.5(ram exceeding cpus * ram/core))
- Are these weights normalized to what CPU counting would give?
  - If not then what does the sum of SlotWeights represent?

Quotas related to sum of SlotWeights, needs to be constant pool-wide and independent of current job allocation—if specifying static number!

- Cost functions need to be linear?
- Only dynamic quota instead (e.g. 80%X + 20%Y)...

Implications

A picture is worth 1000 words...

The more barriers between nodes that can be broken down the better

- MPI-like batch software with NUMA-aware scheduling making other machines like further away NUMA nodes?
ATLAS Load Balancing

PANDA contains knowledge of upcoming work

Wouldn’t it be nice to adapt the group allocation accordingly

- A few knobs can be tuned—surplus and quota
- Dynamic adjustment based on current pending-work
- Gather heuristics on past behavior to help

Timeframe: Fall 2014—project will be picked up by a summer student this year
Data stored on PHENIX nodes (dCache)
- New this year is RAW data is placed hot off the DAQ
- Reprocessing no longer requires second read from tape
  - Less tape wear, faster—no stage latency
- Analysis continues to read input from dCache

No intelligent placement of jobs
- HDFS-like job placement would be great—but without sacrificing throughput
- Approach:
  - Need to know where files are first!
  - Need to suggest placement without waiting for the perfect slot
- Started as proof-of-concept for efficacy of non-flat network
  - Testing Infiniband fabrics with tree-based topology
PHENIX Job Placement

File placement harvested from nodes and placed in database
◦ Database contains map of file->machine
◦ Also contains machines->rack and rack->rack-group mapping

Machines run a STARTD_CRON to query & advertise their location

Job-RANK statement used to steer jobs towards machines where their files are
◦ E.g: \(3 \times (\text{Machine}="a" \mid \mid \text{Machine}="c") + 2 \times (\text{Rack}="21-6") + (\text{RackGroup} = \"10\")\)
◦ Slight increase in negotiation time, upgraded hardware to compensate
◦ Several thousand matches/hr with possibly unique RANK statements

Working on modified dCache client to directly read file if on local node
PHENIX Job Placement Results

We achieved expected results with machine-local jobs
- >80% on an empty farm
- ~10% on a full farm

All localization in rack-group
- >90% empty farm
- ~15% full farm

Argues that great benefit could be gained from utilizing multi-tier networking
- Without it, only machine-local jobs benefit
PHENIX Job Placement Results

1. Machine-Local
2. Rack-Local (exclusive of Machine-Local)
3. All Localized (Sum 1. + 2.)
4. All Jobs
PHENIX Job Placement Results

1. Machine-Local
2. Rack-Local (exclusive of Machine-Local)
3. All Localized (Sum 1. + 2.)
PHENIX Job Placement Results

1. Machine-Local
2. Rack-Local (exclusive of Machine-Local)
3. Non-Local

Histogram of portion of jobs in each state taken in 1 hour intervals

*Plots generated by Alexandr Zaytsev*
THANK YOU

Questions? Comments?
Stock Photo?
Configuration Changes

TAking advantage of config-dir

Since 7.8 Condor supports a config.d/ directory to read configuration from

More easily allows programmatic/automated management of configuration

Refactored configuration files at RACF to take advantage

Old Way

Main Config:
LOCAL_CONFIG_FILES = /dir/a, /dir/b

Order:
1. /etc/condor/condor_config (or $CONDOR_CONFIG)
2. /dir/a
3. /dir/b

New Way

Main Config:
LOCAL_CONFIG_DIR = /etc/condor/config.d
LOCAL_CONFIG_FILES = /dir/a, /dir/b

Order:
1. /etc/condor/condor_config (or $CONDOR_CONFIG)
2. /etc/condor/config.d/* (in alphanumerical order)
3. /dir/a
4. /dir/b