

RACF Condor and ATLAS Multicore Jobs

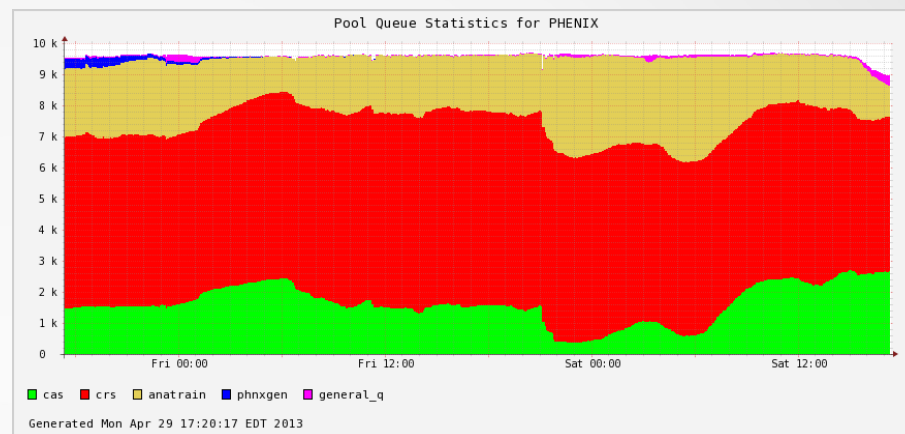
*Condor at the RACF and
matching heterogeneous jobs with heterogeneous
resources in our ATLAS workflow*

Talk Outline

- RHIC/ATLAS Computing Facility (RACF)
 - Our Condor Setup
 - Recent Changes
- Hierarchical Group Quotas in ATLAS
 - Issues, problems, and solutions
- Supporting Multicore Jobs
 - Where we are and where we'd like to be
- Speculation about future developments

RACF Batch System Overview

- Condor pools at the RACF
 - PHENIX—12.7kCPU
 - STAR—11.9kCPU
 - ATLAS—14.0kCPU
- Characteristics
 - RHIC—federation of individual users, some central control, data on nodes
 - ATLAS—tightly controlled, master batch system (PANDA), central data, strict group layout



- Smattering of smaller experiments are users of batch system
 - LBNE
 - Dayabay
 - LSST
 - BRAHMS / PHOBOS

How We Use Condor at the RACF

- Job Submission
 - ATLAS—jobs submitted locally via 6 main submit nodes, each handling 3-4k jobs, coming from PANDA through autopyfactory
 - PHENIX + STAR
 - Interactive machines (20 and 10 respectively) allow direct user submission
 - Special nodes for submitting to special queues (CRS / Anatrain)
- 4 Central Managers
 - Need >4Gb RAM
 - 1Gb each for collector + negotiator
 - Collector sometimes forks so need extra 1-2 Gb
- Condor Packages
 - Build our own from any git snapshot, configure to only include libraries / features we use
 - Configuration is puppet-managed

Configuration Changes

- Since 7.8.x condor allows configuration macros in a config.d/ directory
- Leveraged to make puppet-management much easier
 - Single files much easier to handle
- Reduced by thousands of lines the configs we maintain
 - Mostly same main config over-and-over

Old Way

Main Config:

```
LOCAL_COFIG_FILES = /dir/filea, /dir/fileb
```

Order:

1. /etc/condor_config (or \$CONDOR_COFIG)
2. /dir/filea
3. /dir/fileb

New Way

Main Config:

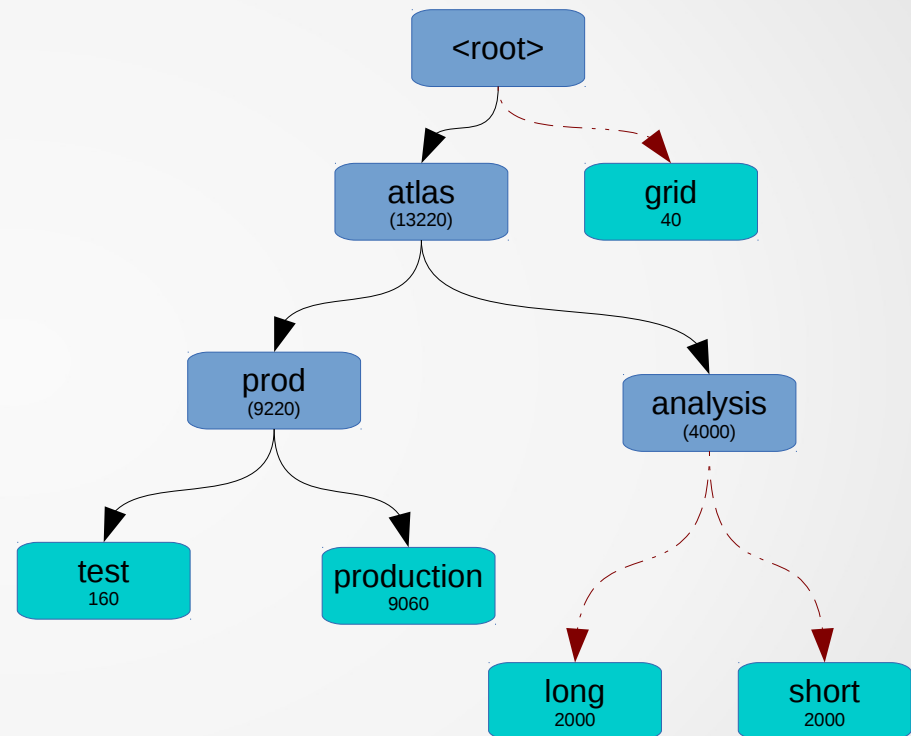
```
LOCAL_CONFIG_DIR = /etc/condor/config.d/  
LOCAL_COFIG_FILES = /dir/filea, /dir/fileb
```

Order:

1. /etc/condor_config (or \$CONDOR_COFIG)
2. /etc/condor/config.d/* (in alphanumeric order)
3. /dir/filea
4. /dir/fileb

ATLAS Structure

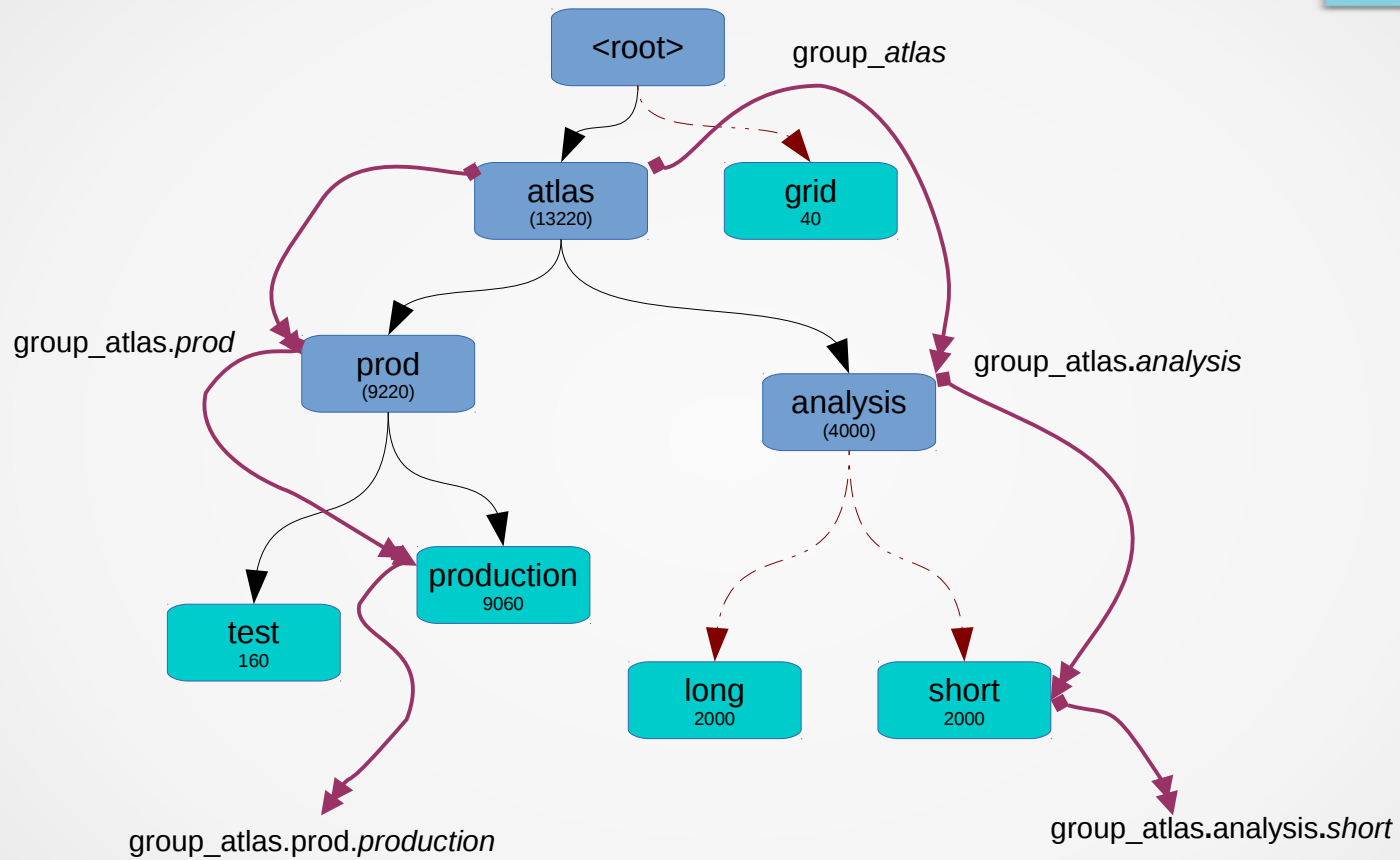
- Flat, uniform farm in both hardware and software (for now)
- PANDA Queues map to AccountingGroup(s)
- Hierarchical structure
- Only leaf nodes have jobs submitted to them
- Spillover between arbitrary (related) groups
 - *short* and *long* can share but are constrained to 4k by parent (*analysis*)
 - *grid* can accept all surplus not used by ATLAS



Key

- Turquoise
leaf group with jobs
- Blue
middle group, quota is sum of children, no jobs
- Red arrow
group has accept_surplus on

ATLAS Structure Example



Multicore Support 1st Attempt

- Static slots
 - Initially a test queue with a group under production
 - 24-core machines with 2x8_{CPU} and 8x1_{CPU} slots
 - Others with just 3x8_{CPU} slots
 - Jobs set to require (CPUS == 8) in job-description-file
- Discovered problem with groups—wanted quota usage to be #CPUs (default slot-weight)—but jobs wouldn't match correctly (see condor ticket [#2958](#))
 - Fix was provided, but still failed when *any* group has **accept_surplus** enabled
- We need HGQ with `accept_surplus` *and* multicore jobs in ATLAS
 - Kludge fix: set slot-weight to 1, but this throws off accounting/fairness

Node Consistency

- Reasons to avoid hard partitioning:
 - Balance between queues changes frequently
 - Made >30 adjustments this year so far
 - Configs need to change to adapt to differing workloads
 - Limitations of Condor in altering fundamental machine characteristics
 - Can't change slot count or slot resource-makeup without restart
 - Restart = full drain = inefficient
 - Even harder for cloud nodes
 - Maintain balance with machines appearing and disappearing
- Theme: keep nodes the same!
 - Even with tools like puppet, partitioning the farm by config is inefficient

Multicore Problems in ATLAS

- Three Problems
 - Previously mentioned multicore support
 - Long standing structural issue with group-quotas and surplus-sharing.
 - Children with `accept_surplus` would violate parent group's quota under certain circumstances
 - (longer term) Jobs can only split along local resources like CPU, Disk, RAM.
 - Will need to define arbitrary “consumable” resources

Q: How to integrate multicore jobs into existing groups?

Q: How to integrate high-memory jobs into existing groups?

A: Partitionable Slots (pslots)!

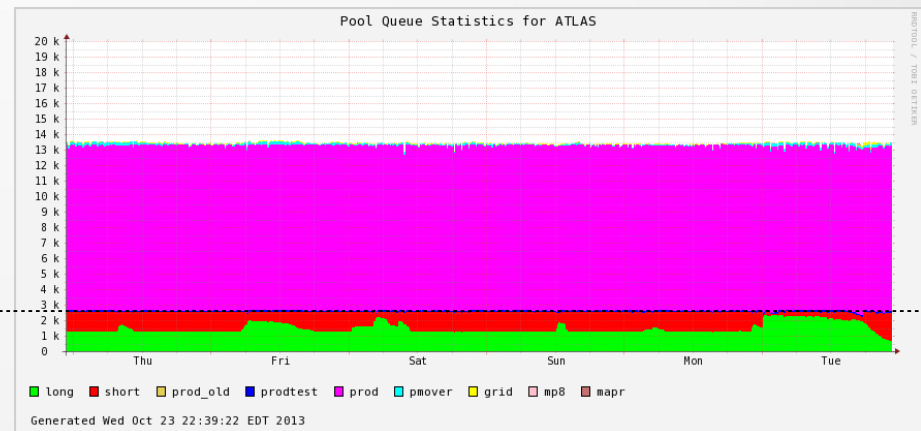
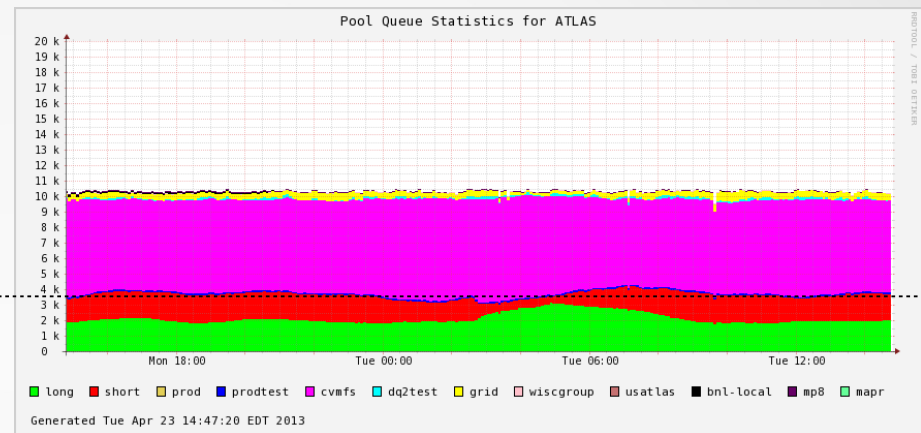
Not Working With Group Quotas



Fixing bugs in Condor

- Over summer a period of intensive development / testing in collaboration with the Condor team to fix these issues
 - Built a VM testbed, rapid build & test of patches from condor team
 - Built new monitoring interface
 - After many cycles, had working config with partitionable slots and HGQ with **accept_surplus!**

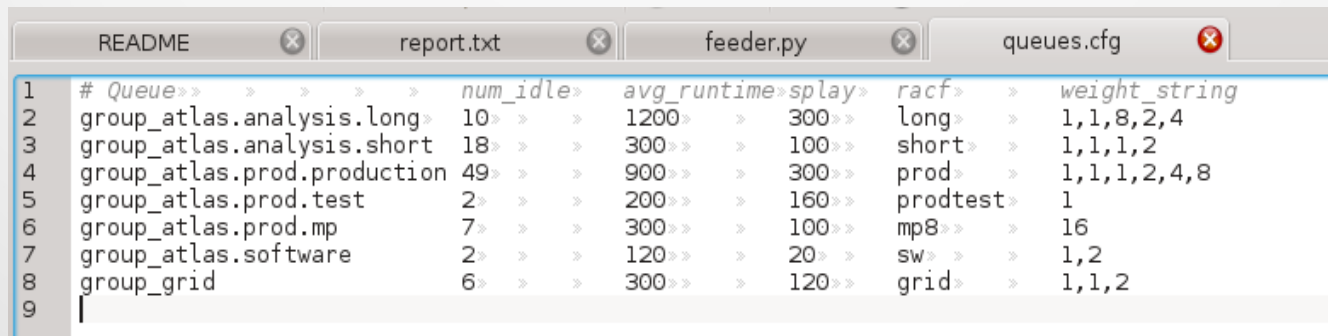
Supposed to stay here...



...now, it does 11

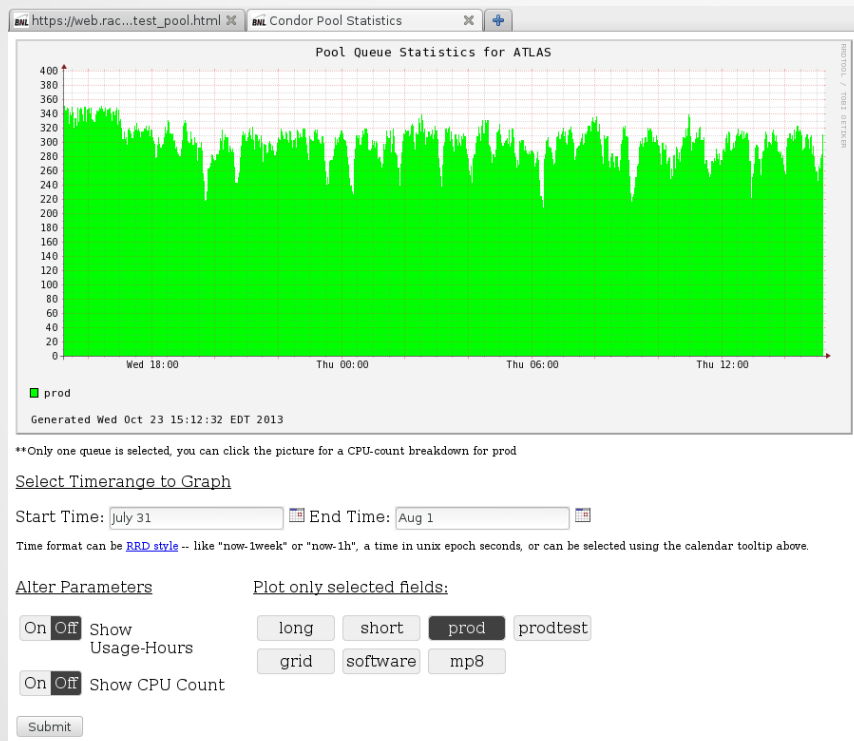
Bugfix Testbed Details

- Rapid build, test, deploy cycle from git patches:
 - Email patches
 - Build condor
 - Run test job feeder
 - Change parameters & examine behavior
- Job Feeder
 - Define groups in config file, with differing random job-length ranges and cpu-requirements
 - Variable workload: keep N idle jobs of each group in queue at all times
 - Live-tune to simulate real-life workload scenarios



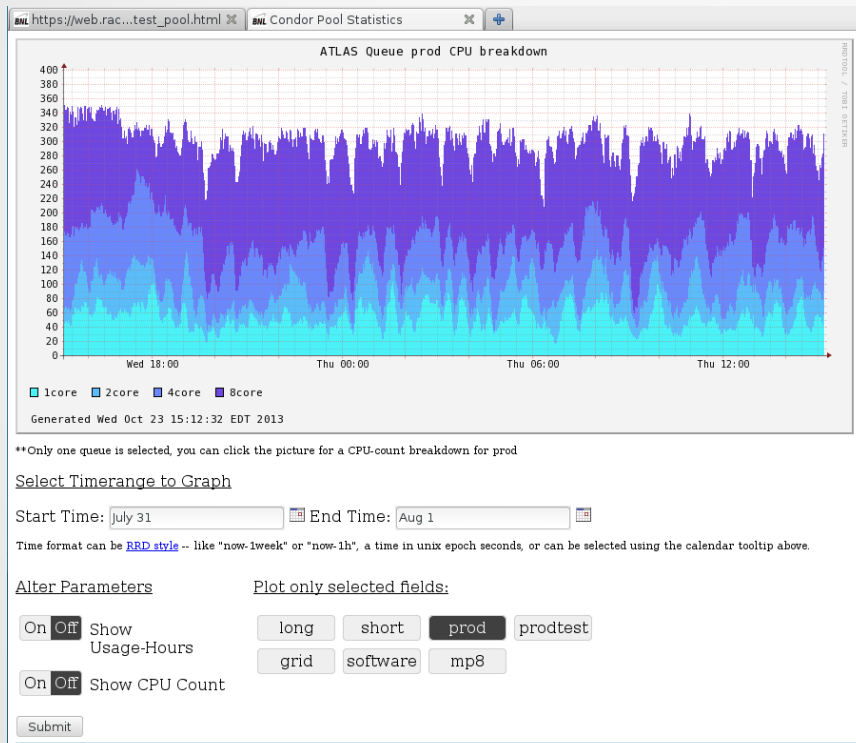
```
1 # Queue>> > > > > num_idle> avg_runtime> splay> racf> > weight_string
2 group_atlas.analysis.long> 10> > > 1200> > 300>> long> > 1,1,8,2,4
3 group_atlas.analysis.short 18> > > 300>> > 100>> short> > 1,1,1,2
4 group_atlas.prod.production 49> > > 900>> > 300>> prod> > 1,1,1,2,4,8
5 group_atlas.prod.test 2> > > 200>> > 160>> prodtest> 1
6 group_atlas.prod.mp 7> > > 300>> > 100>> mp8>> > 16
7 group_atlas.software 2> > > 120>> > 20> > sw> > 1,2
8 group_grid 6> > > 300>> > 120>> grid> > 1,1,2
9 |
```

Multicore jobs visualized



- Weighted random job generator
 - Give weights for how much of the queue should be what species of job
- Visualized here, not much control over this allocation is currently possible

Multicore jobs visualized



- Weighted random job generator
 - Give weights for how much of the queue should be what species of job
- Visualized here, not much control over this allocation is currently possible

Multicore Support After Fixes

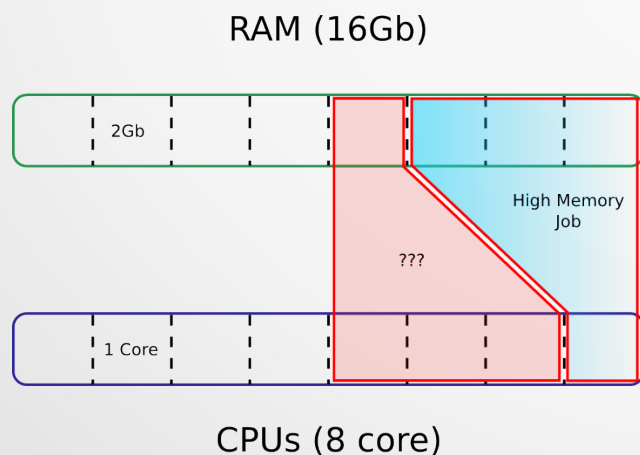
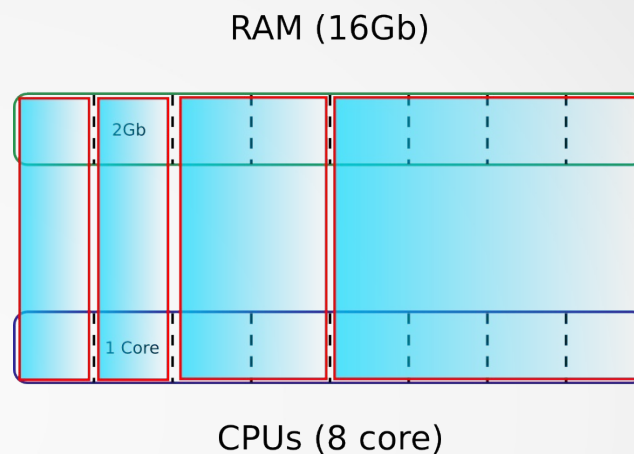
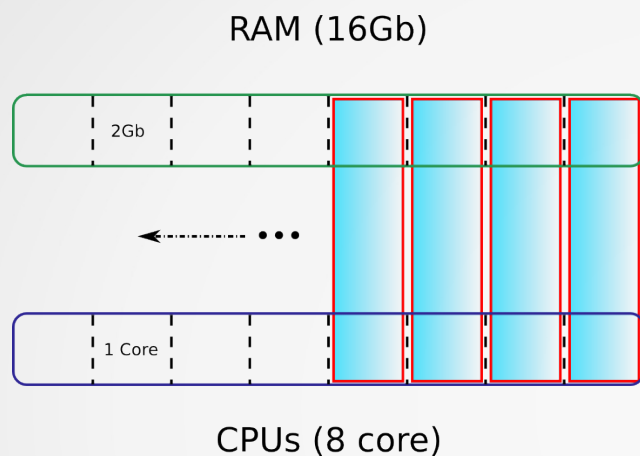
- Fully utilize partitionable slots
 - All traditional nodes (standard x86) can have same configuration:

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

For now, we can live with this since all our hardware is “traditional”

- Fix works perfectly when **accept_surplus** is enabled for any combination of groups
- Only works with SlotWeight=Cpus
 - High-memory jobs can be accommodated by asking for more CPUs
 - Need ability to partition better, what about low-memory high-cpu jobs?
 - Weight should be a (linear) function of all consumable resources

High Memory Jobs Pose Potential Problem



- Clockwise from top-left:
 - Ok, Ok, NOT OK!
- General problem:
 - Inefficiencies in heterogeneous jobs scheduling to granular resources
 - This is just with two dimensions, imagine when GPUs, Disk Space, Xeon PHI CoProcessors, etc... come into play

Partitionable Slot Requirements

- Want to be able to slice by RAM, CPU, and possibly Disk
 - In the future slicing by any local-resource (GPU...)
- Want sane (configurable) defaults for existing job-configs
 - Request: 1 core, TotalRam/TotalCPU memory, etc...
- Want no complete starvation of larger jobs that can be accommodated somewhere
 - Implies some form of defragmentation/drainage
 - Ideally defragmentation would be group-aware

Defragmentation In Detail

- Scheduler-aware defragmentation would help
 - 1 Spread “pain” of defragmentation across users/groups
 - 2 Ensure fair-share respected for users/groups across schedulers
- Implementation ideas
 - 1 Look-ahead at queue to determine defrag targets
 - Looking at demand from idle jobs in queue, or allowing users to provide targets
 - 2 Keep historical data to improve heuristics
 - “This user's jobs in this cluster typically run for X hours”, etc...

Data Driven Scheduling

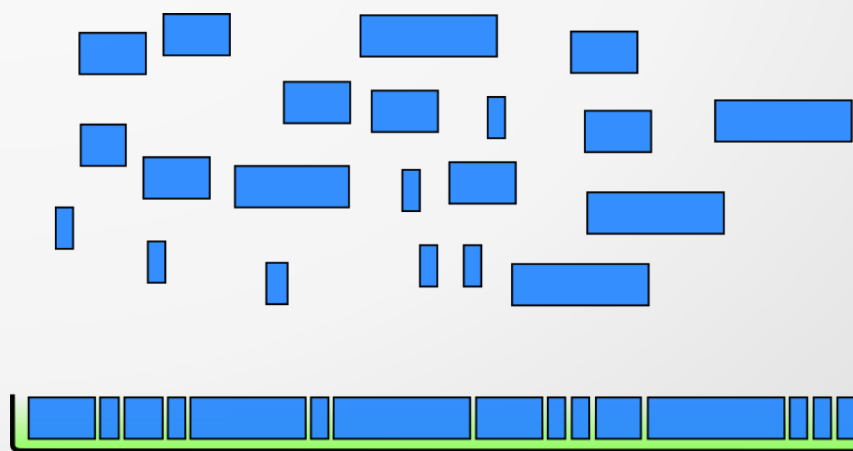
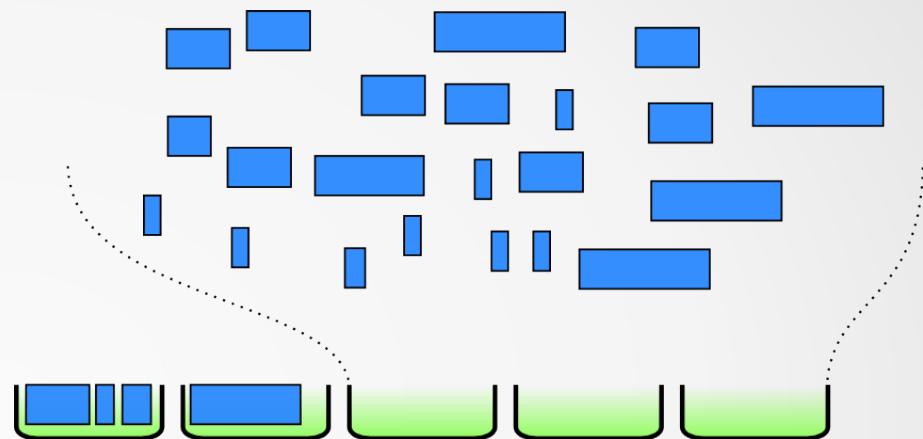
- Given a queue of idle work, no a priori knowledge of the throughput requirement
 - Resource partitioning for a given workload
 - E.g. Do 8-core jobs finish in $1/8^{\text{th}}$ the time of a single core job?
 - VM provisioning for a given work queue
- Historical data collection can help—up to a point
 - Rely on users to provide information on job throughput?
 - Do they even know beforehand?
 - In a system like ATLAS, there are way too many layers between the user and the executing job for this to work

The Problem of Weights and Costs

- What does slot-weight mean with heterogeneous resources?
 - Job of administrator to determine how much to “charge” for resource usage, e.g. $\text{cpus} + 1.5 * (\text{ram exceeding cpus} * \text{ram/core})$
 - Are these weights normalized to what simple CPU counting would give?
 - If not, then what does the sum of the slot-weights represent
 - How to represent resources in same box that are completely orthogonal to ordinary jobs (GPUS, Xeon PHI, etc...)
- Group quotas related to sum of slot-weights, needs to be constant pool-wide regardless of allocation (cost functions must be linear?)
 - Weight related to the maximum capacity
 - CPUs are an irreducible resource
 - Jobs must request at minimum 1 core each
 - Should other resources be quantized?

Future Speculation

- Don't think of matching a job to a slot
 - Jobs are a set of resource-requirements
 - A compute farm is a large pool of resources chopped up in arbitrary places
 - A “match” is a multidimensional slice of a resource that can satisfy the job's requirements
- Larger “blobs” of compute are better—quantization of resources leads to inefficiency
 - Which looks easier to fit jobs into, the top or bottom picture?
- Breaking down barriers between nodes—increased use of MPI-like software with NUMA-aware scheduling making other machines just like farther away NUMA nodes?



Thank You!

Questions? Comments?