No Idle Cores

DYNAMIC LOAD BALANCING IN ATLAS & OTHER NEWS

WILLIAM STRECKER-KELLOGG <WILLSK@BNL.GOV>

RHIC/ATLAS Computing Facility Overview

- Main HTCondor Pools
  - STAR, PHENIX, ATLAS
    - Each just over 15kCPU
- Running stable 8.2.7
- RHIC Pools
  - Individual+Special Users
    - Workload management done by experiments
- ATLAS: Focus of this talk
  - Local batch systems driven by external workload manager (PANDA)
  - Jobs are pilots
    - Scheduling→provisioning

ATLAS Configuration

- Use Hierarchical Group Quotas + Partitionable Slots
  - My HTCondor Week talk last year was all about this
- A short recap:
  - PANDA Queues map to groups in a hierarchical tree
    - Leaf-nodes have jobs
    - Surplus-sharing is selectively allowed
- Group allocation controlled via web-interface to DB
  - Config file written when DB changes

- All farm has one STARTD config

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

Partitionable Slots

- Each batch node is configured to be partitioned into arbitrary slices of CPUs
  - Condor terminology:
    - Partitionable slots are automatically sliced into dynamic slots
- Multicore jobs are thus accommodated with no administrative effort
  - Only minimal (~1-2%) defragmentation necessary
    - Empirically based on our farm—factors include cores/node, job sizes & proportions, and runtimes. Something like

draining=(job-length*job-size^2)/(machine-size*%mcore*occupancy)

Defragmentation Policy

Defragmentation Daemon

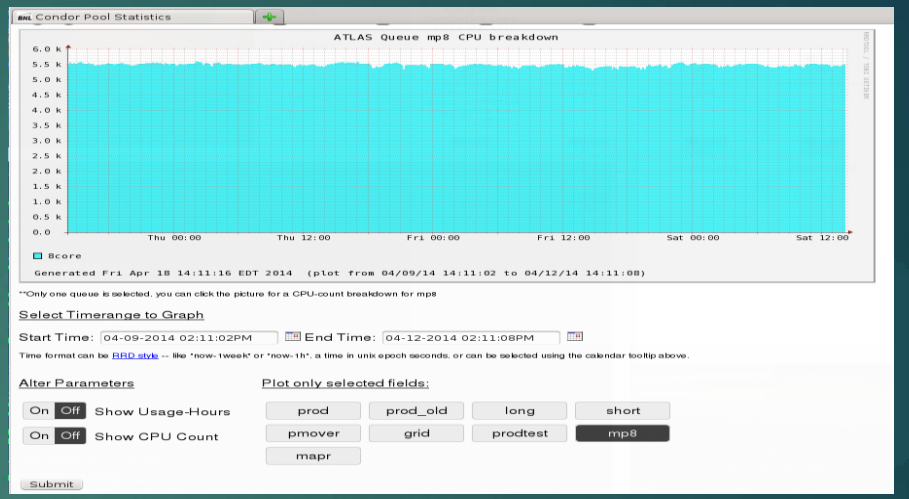
- Start Defragmentation
  - `(PartitionableSlot && !Offline && TotalCpus > 12)`
- End Defragmentation
  - `(Cpus >= 10)`
- Rate: max 4/hr

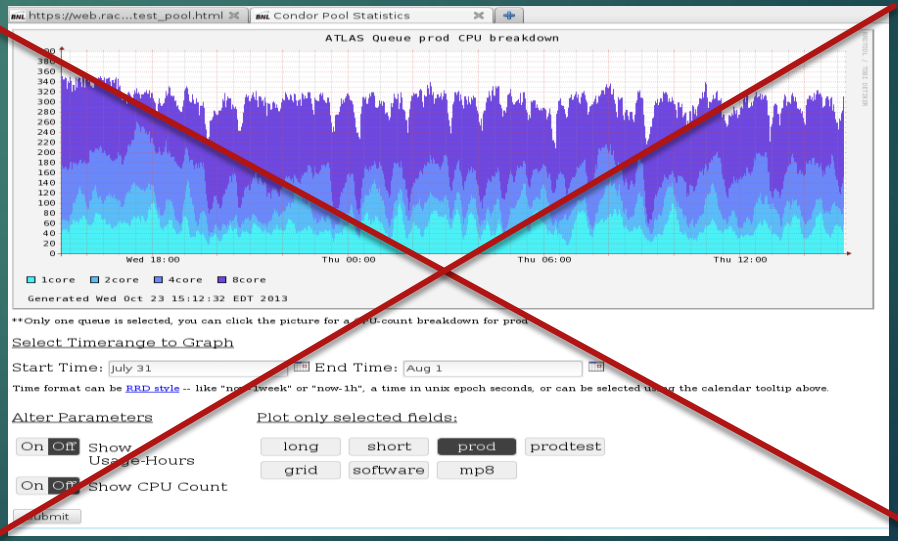
Key change: Negotiator Policy

- Setting NEGOTIATOR_POST_JOB_RANK
- Default policy is breadth-first filling of equivalent machines
  - `(Kflops - SlotId)`
- Depth-first filling preserves continuous blocks longer
  - `(-Cpus)`

PANDA Queues

- PANDA Queues
  - One species of job per-queue
  - Map to groups in our tree
- Currently two non-single-core queues
  - 8-core ATHENA-MP
  - 2-core (Actually high-memory)
    - No support yet for SlotWeight!=cpus
    - Have 2Gb/core, so 4Gb jobs get 2 cores





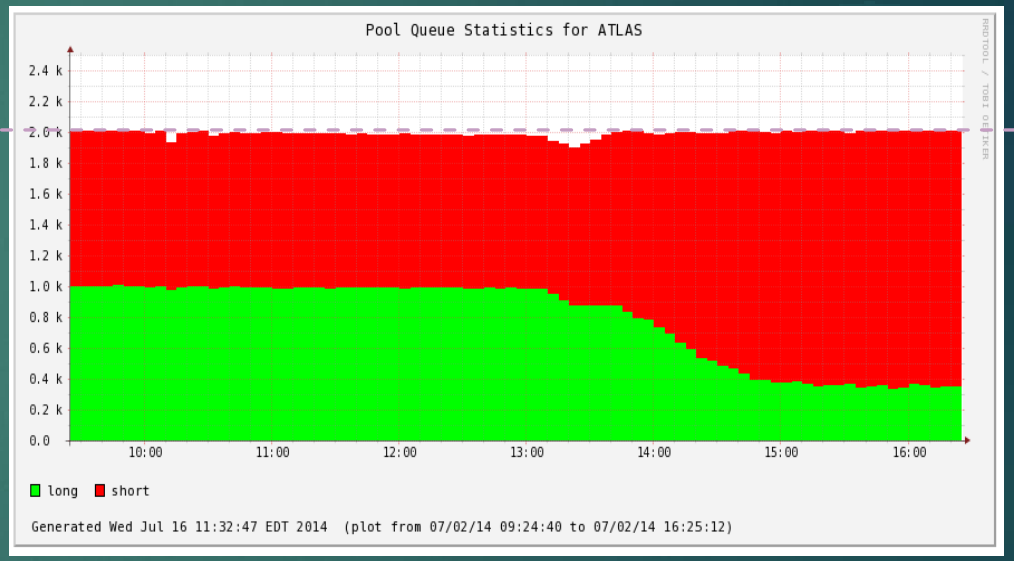
ATLAS Tree Structure

Surplus Sharing

- Surplus sharing is controlled by boolean *accept_surplus* flag on each queue
  - Quotas are normalized in units of SlotWeight (CPUs)
- Groups with flag set to True can take unused slots from their siblings
  - Parent groups with flag allow surplus to "flow down" the tree from their siblings to their children
  - Parent groups without *accept_surplus* flag constrain surplus-sharing to among their children

Surplus Sharing

- Scenario: **analysis** has quota of *2000* and no *accept_surplus*; **short** and **long** have a quota of *1000* each and *accept_surplus* on
  - short=1600, long=400…possible
  - short=1500, long=700…impossible (violates analysis quota)



Where's the problem? (it's starvation)

- Everything works perfectly with all single-core, just set accpet_surplus everywhere!
- However… Multicore jobs will not be able to compete for surplus resources fairly
  - Negotiation is greedy, if 7 slots are free, they won't match an 8-core job but will match 7 single-core jobs in the same cycle
    - If any multicore queues compete for surplus with single core queues, the multicore will always lose
- A solution outside Condor is needed
  - Ultimate goal is to maximize farm utilization—No Idle Cores!

Dynamic Allocation

- A program to look at the current state of the demand in various queues and set the surplus-flags appropriately
  - Based on comparing "weight" of queues
    - Weight defined as size of jobs in queue (# cores)
  - Able to cope with any combination of demands
  - Prevents starvation by allowing surplus into "heaviest" queues first
    - Avoids both single-core and multicore queues competing for the same resources
  - Same algorithm is extensible up the tree to allow sharing between entire subtrees
  - Much credit to Mark Jensen (summer student in '14)

Balancing Algorithm

- Groups have the following properties pertinent to the algorithm
  - Surplus flag
  - Weight
  - Threshold
  - Demand

- If Demand > Threshold a queue is considered for sharing

```
MariaDB [group_quotas]> select group_name, quota, weight, accept_surplus as surplus, sur
plus_threshold as threshold from atlas_group_quotas;
+----------------------------+-------+--------+---------+-----------+
| group_name                 | quota | weight | surplus | threshold |
+----------------------------+-------+--------+---------+-----------+
| group_atlas                | 14823 |      2 |       0 |         0 |
| group_atlas.analysis       |  4008 |      1 |       0 |         0 |
| group_atlas.analysis.long  |  2000 |      1 |       0 |       250 |
| group_atlas.analysis.mcore |     8 |      0 |       0 |        20 |
| group_atlas.analysis.short |  2000 |      1 |       0 |       250 |
| group_atlas.prod           | 10810 |      2 |       0 |         0 |
| group_atlas.prod.mp        |  6110 |      8 |       1 |        40 |
| group_atlas.prod.production |  3700 |      1 |       0 |       160 |
| group_atlas.prod.test      |  1000 |      2 |       0 |        80 |
| group_atlas.software       |     5 |      0 |       0 |         1 |
| group_grid                 |   500 |      1 |       0 |        50 |
+----------------------------+-------+--------+---------+-----------+
11 rows in set (0.00 sec)

MariaDB [group_quotas]>
```

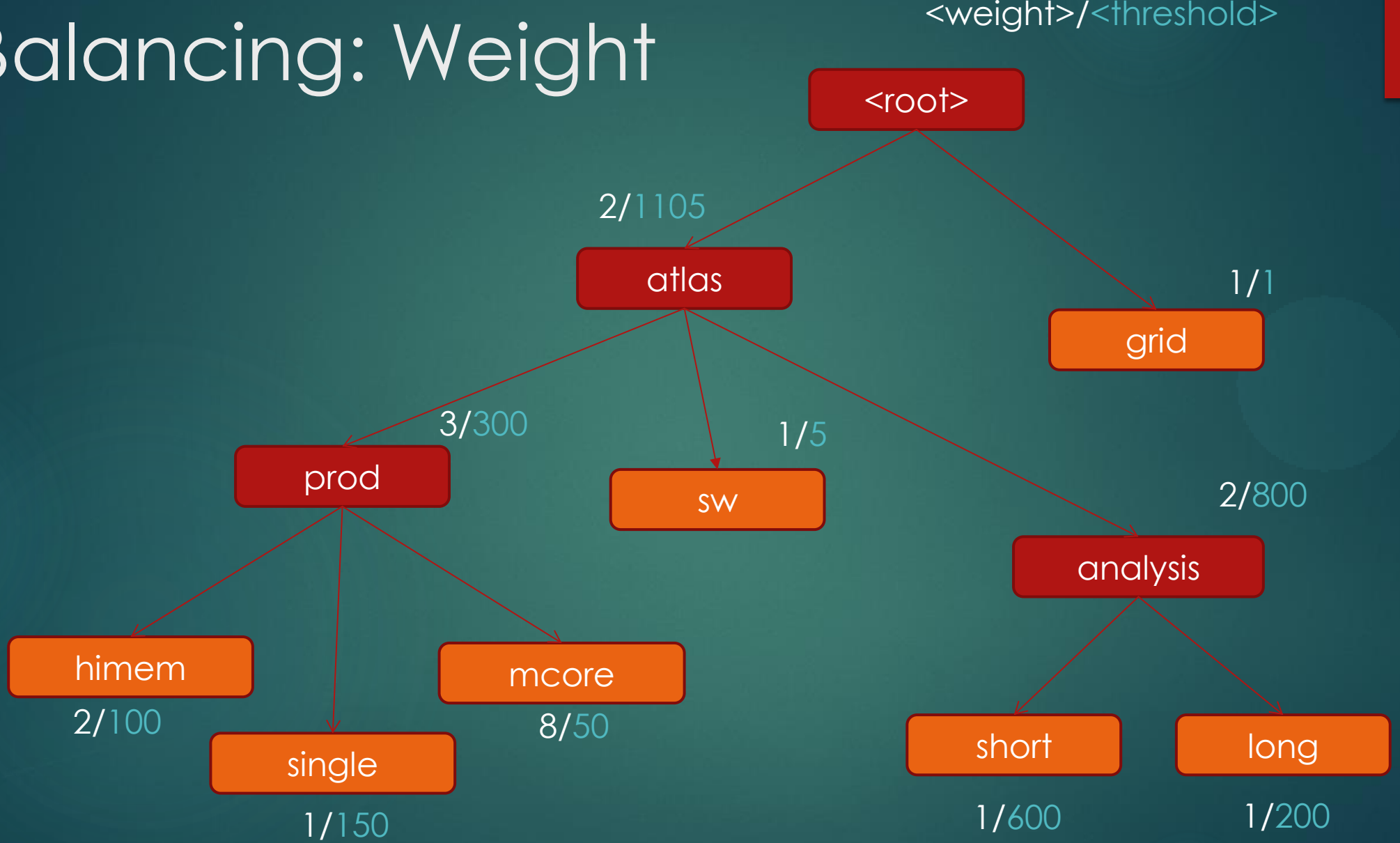
Balancing: Demand

- PANDA Queues are monitored for “activated” jobs
  - Polling every 2 minutes
- Last hour is analyzed
  - Midpoint sampling
  - Moving average
  - Spikes smoothed out
  - Queue considered “loaded” if calculated demand > threshold

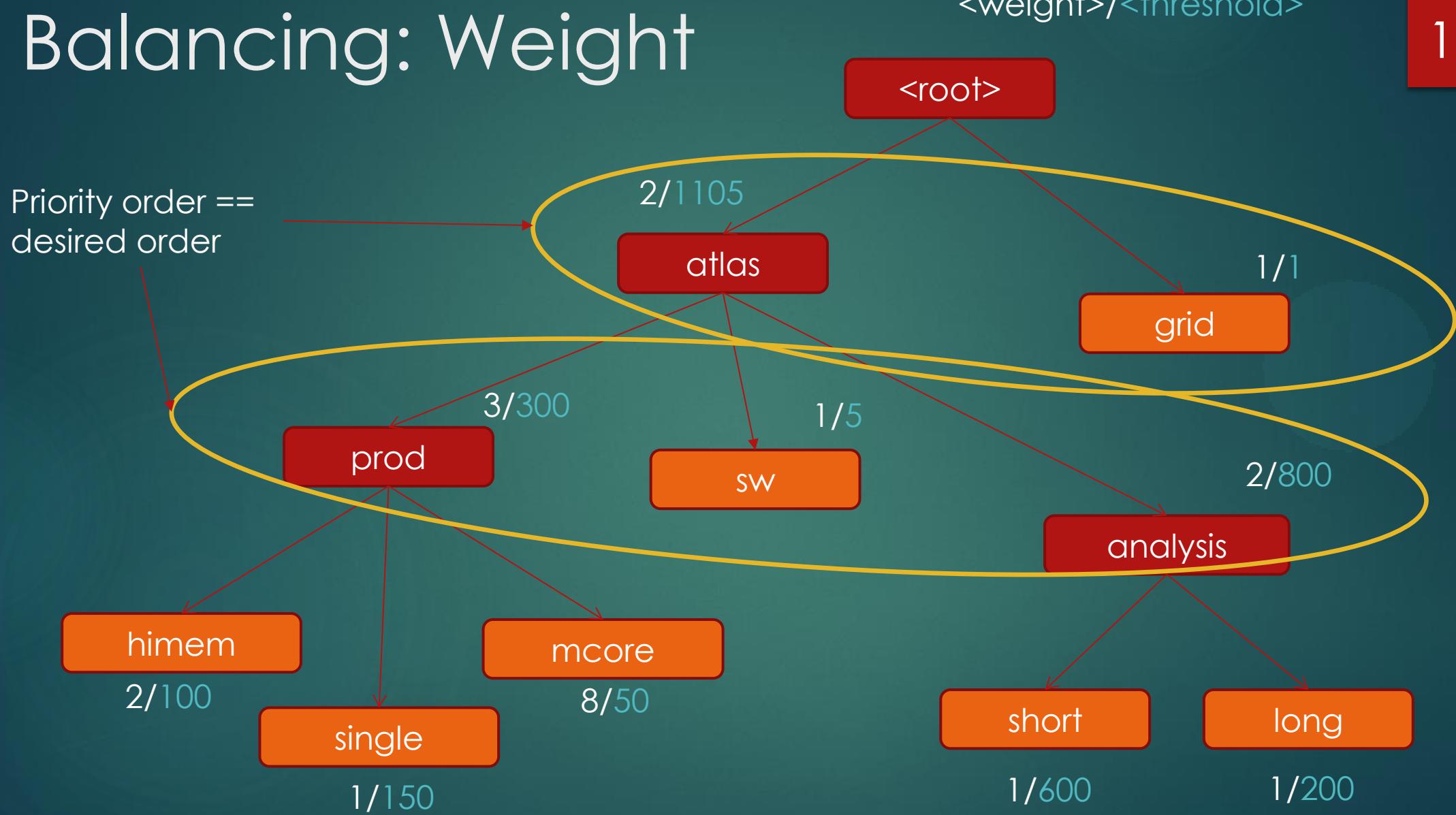
Extending Weight & Demand

- Leaf groups' weights are the cores they need (8, 2, 1)
- How to extend beyond leaf-groups?
  1. Define in custom priority order
  2. Define as sum() or avg() of child groups's weights
     - Problem with 1. is you can't guarantee starvation-free
     - For now, manually set weights to match what would be the case for 2.
- For demand and threshold: easy—sum of child-groups values

Balancing: Weight



Balancing: Weight



Algorithm

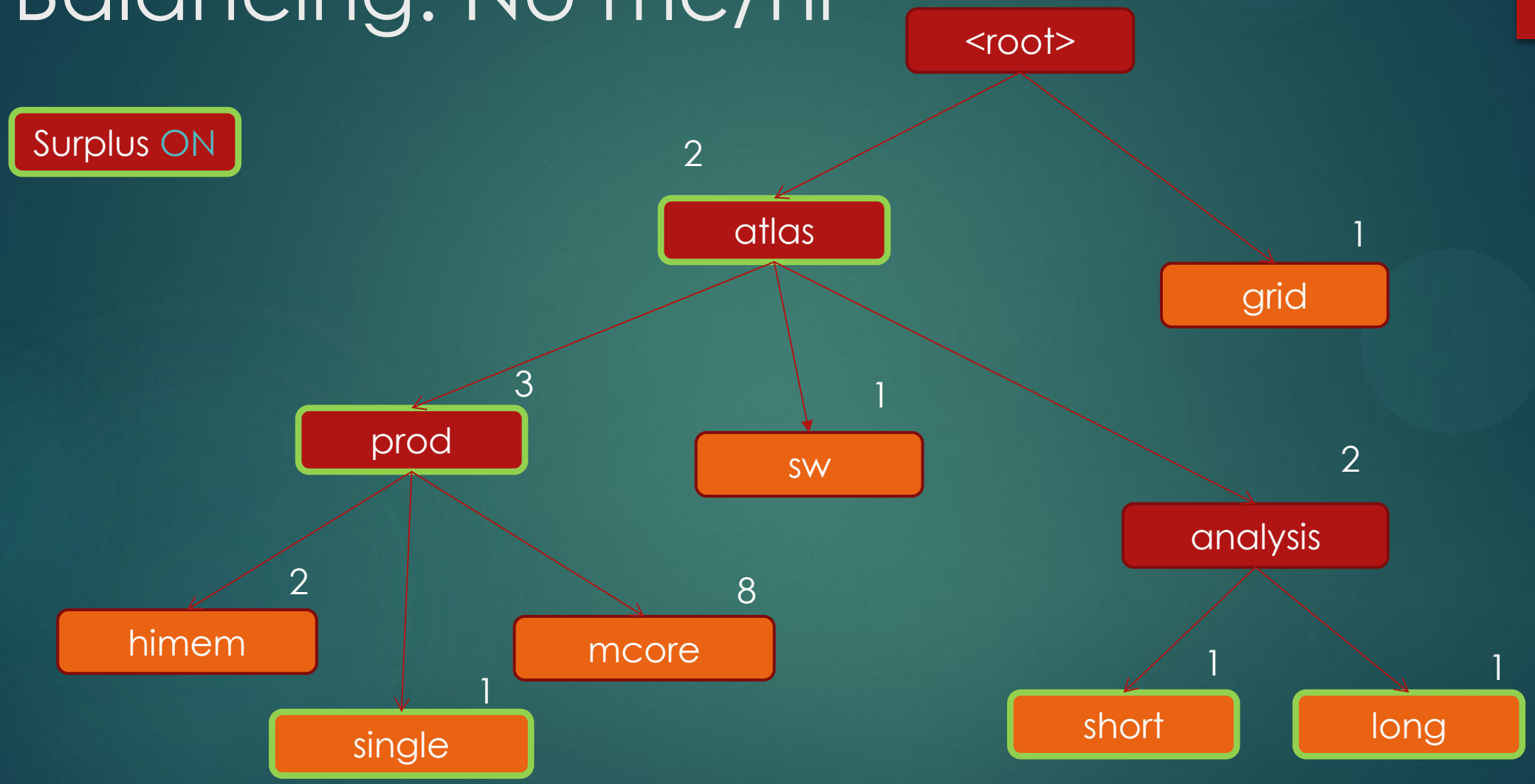
- The following algorithm is implemented
  1. For each sibling-group in DFS order:
     1. For each member in descending weight order
        1. Set to TRUE unless it does not have demand and lower-weight groups do
        2. Break if set to TRUE

- In other words…

In each group of siblings, set *accept_surplus* to TRUE for all the highest-weighted groups that have demand

Balancing: All Full

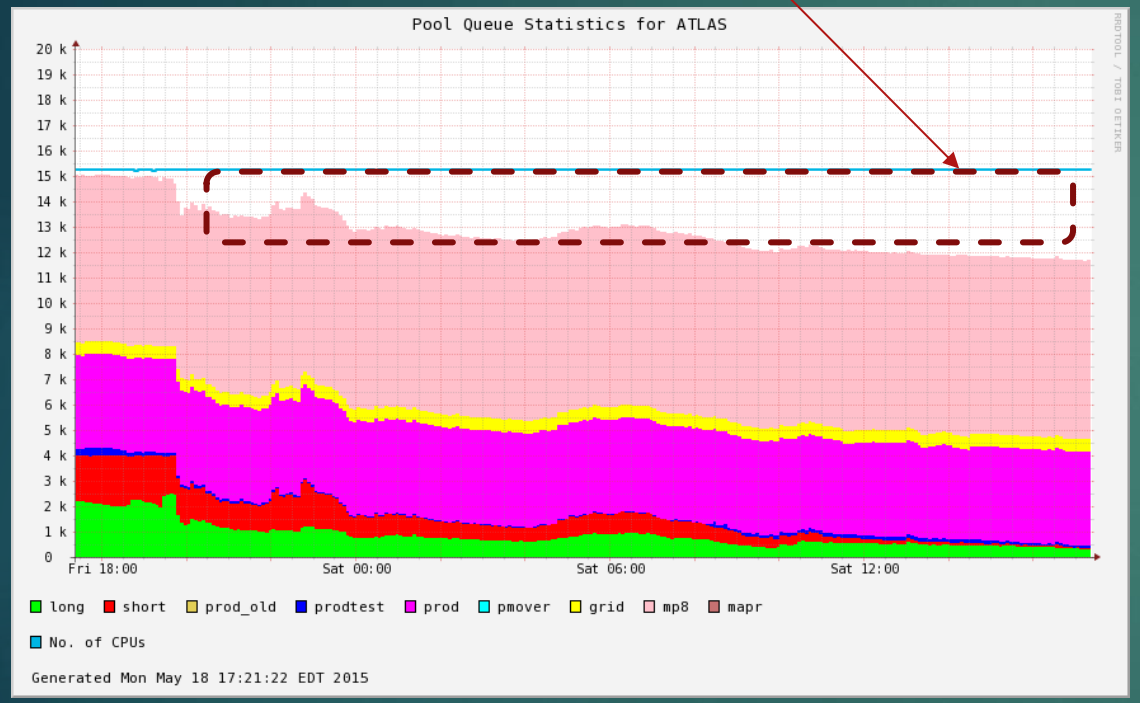
Balancing: No mc/hi

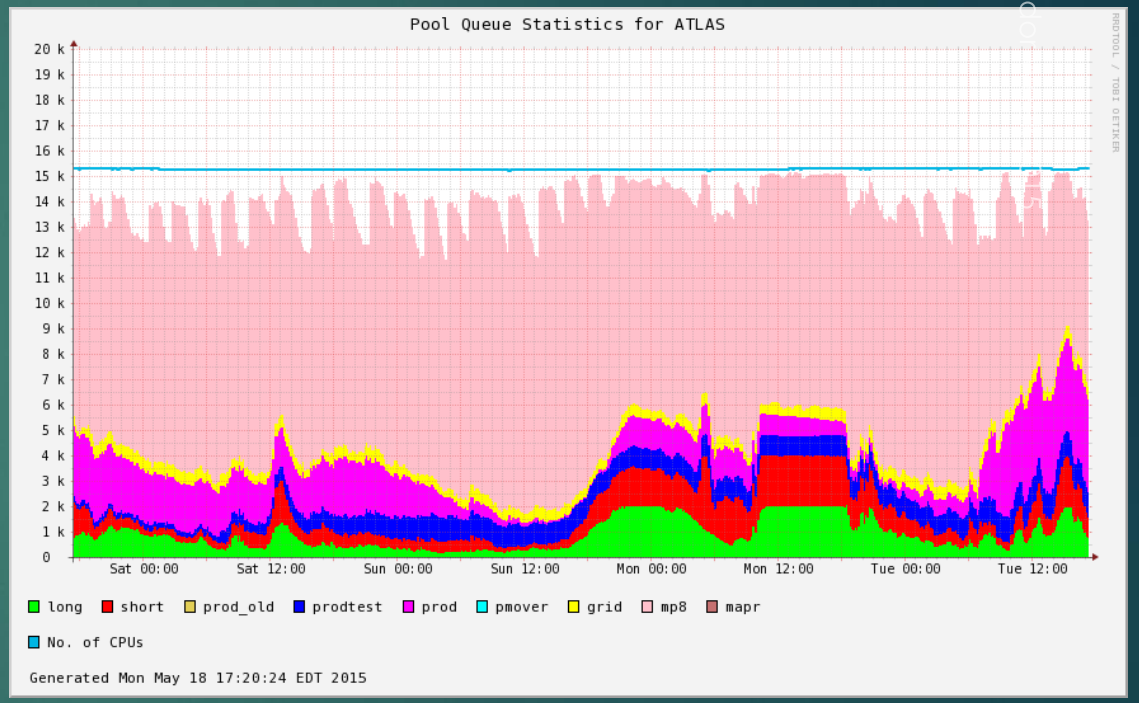


Balancing: No prod

Results

Wasted Slots





Results & Context

- Multicore is ready to take slack from other production queues
- Spotty analysis-demand the past few months has allowed many millions of CPU-hours to go unwasted
- If all ATLAS has a lull in demand, OSG jobs can fill the farm
  - Caveat: Preemption!
- Fast Negotiation
  - Averages for last 3 days:

Matches	14.99
Duration	7.05s

- Who is this useful for?
  - Algorithm works for any tree
    - Extensible beyond ATLAS where work is structured outside of batch system
  - A multi-tenant service provide with a hierarchy of priorities
    - Really a problem of efficient provisioning, not scheduling
- Constraints
  - Workflow defined outside of HTCondor
  - Segregation of multicore in separate queues for scheduling

Desired Features & Future Work

Preemption

- Wish to maintain reasonably minimum-runtime to grid jobs
- When ATLAS demand comes back, need OSG jobs to be evicted
- Require preempting the *dynamic* slots that are created under the *partitionable* one
  - Work is progressing along these lines, although final state is not clear

SlotWeight != CPUs

- Would like to "value" RAM less than CPUs for jobs
  - High-memory kludge is inelegant
  - Not extensible to different shaped jobs (high-RAM/low-CPU, vice versa)
- Tricky because total slot-weight of the farm needs to be constant to give meaning to quota allocation

The End

QUESTIONS? COMMENTS?

THANKS TO MARK JENSEN, AND THE HTCONDOR TEAM!