Matchmaker Policies: Users and Groups

HTCondor Week, Madison 2016

Zach Miller (zmiller@cs.wisc.edu)
Jaime Frey (jfrey@cs.wisc.edu)

Center for High Throughput Computing
Department of Computer Sciences
University of Wisconsin-Madison

HTCondor scheduling policy

- So you have some resources…
  … how does HTCondor decide which job to run?
- The admin needs to define a policy that controls the relative priorities
- What defines a "good" or "fair" policy?

First Things First

- HTCondor does not share the same model of, for example, PBS, where jobs are placed into a first-in-first-out queue
- It instead is based around a concept called "Fair Share"
  - Assumes users are competing for resources
  - Aims for long-term fairness

Spinning Pie

- Available compute resources are “The Pie”
- Users, with their relative priorities, are each trying to get their “Pie Slice”
- But it’s more complicated: Both users and machines can specify preferences.
- Basic questions need to be answered, such as “do you ever want to preempt a running job for a new job if it’s a better match”? (For some definition of “better”)

Spinning Pie

- First, the Matchmaker takes some jobs from each user and finds resources for them.
- After all users have got their initial “Pie Slice”, if there are still more jobs and resources, we continue “spinning the pie” and handing out resources until everything is matched.

Relative Priorities

- If two users have the same relative priority, then over time the pool will be divided equally among them.
- Over time?
- Yes! By default, HTCondor tracks usage and has a formula for determining priority based on both current demand and prior usage
- However, prior usage “decays” over time

Pseudo-Example

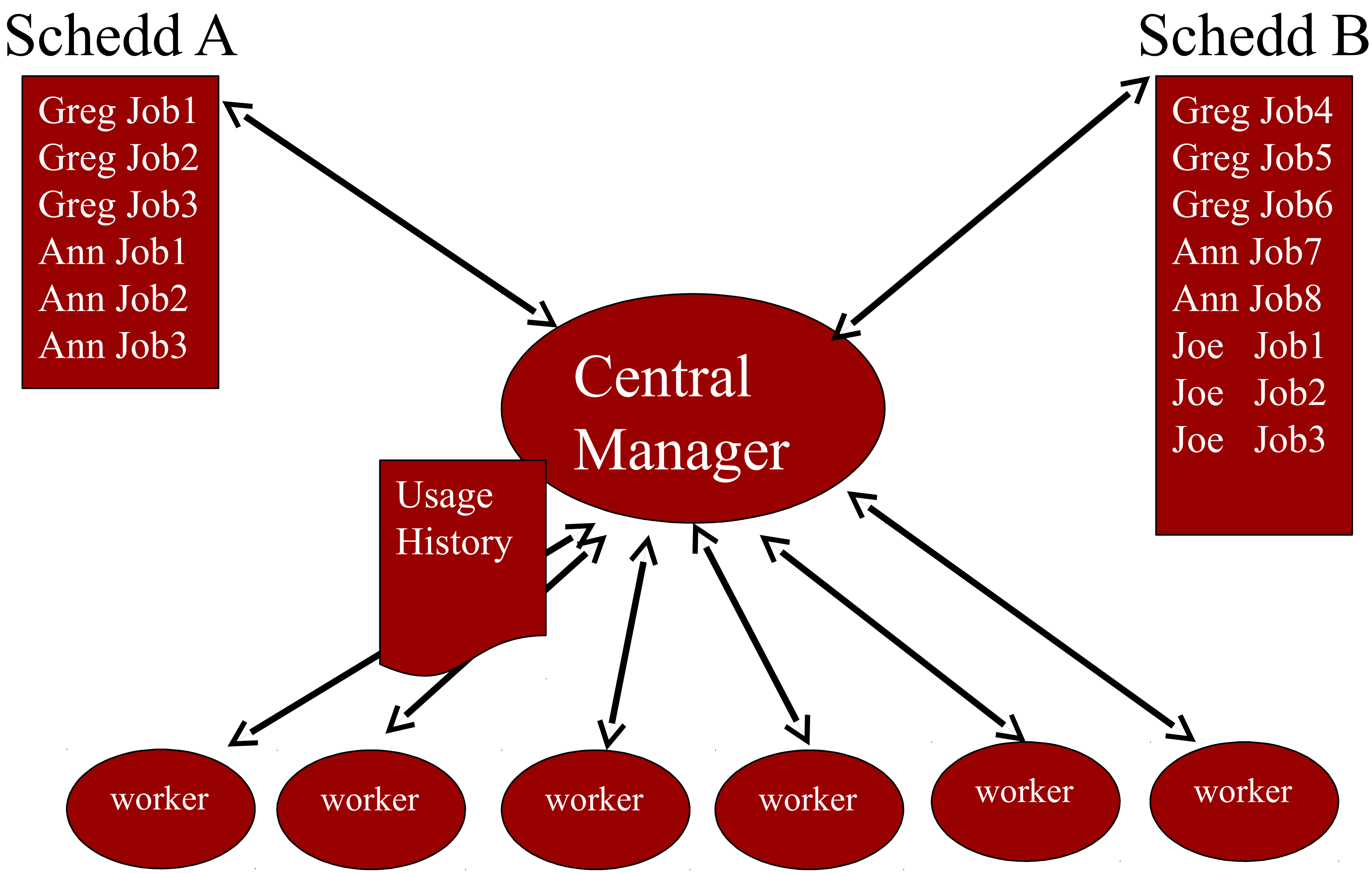
- Example: (A pool of 100 cores)
- User ‘A’ submits 100,000 jobs and 100 of them begin running, using the entire pool.
- After 8 hours, user ‘B’ submits 100,000 jobs

- What happens?

Pseudo-Example

- Example: (A pool of 100 cores)
- User 'A' submits 100,000 jobs and 100 of them begin running, using the entire pool.
- After 8 hours, user 'B' submits 100,000 jobs
- The scheduler will now allocate MORE than 50 cores to user 'B' because user 'A' has accumulated a lot of recent usage
- Over time, each will end up with 50 cores.

Overview of Condor Architecture



Negotiator metric: User Priority

- Negotiator computes, stores the user prio

- View with **`condor_userprio`** **tool**
- Inversely related to machines allocated (lower number is better priority)
  - A user with priority of 10 will be able to claim twice as many machines as a user with priority 20

What's a user?

- Bob in schedd1 same as Bob in schedd2?
- If have same UID_DOMAIN, they are.

- We'll talk later about other user definitions.

- Map files can define the local user name

User Priority

- (Effective) User Priority is determined by multiplying two components

- Real Priority * Priority Factor

Real Priority

- Based on actual usage
- Starts at 0.5
- Approaches actual number of machines used over time
  - Configuration setting **PRIORITY_HALFLIFE**
  - If PRIORITY_HALFLIFE = +Inf, no history
  - Default one day (in seconds)
- Asymptotically grows/shrinks to current usage

Priority Factor

- Assigned by administrator
  - Set/viewed with **`condor_userprio`**
  - Persistently stored in CM
- Defaults to 100 (**`DEFAULT_PRIO_FACTOR`**)
- Allows admins to give prio to sets of users, while still having fair share within a group
- "Nice user"s have Prio Factors of 1,000,000

condor_userprio

› Command usage:

```
condor_userprio
                                         Effective   Priority
User Name                                 Priority    Factor   In Use (wghted-hrs) Last Usage
---------------------------------------------- --------- --------- ------ ------------ ----------
lmichael@submit-3.chtc.wisc.edu                     5.00     10.00      0        16.37    0+23:46
blin@osghost.chtc.wisc.edu                          7.71     10.00      0      5412.38    0+01:05
osgtest@osghost.chtc.wisc.edu                      90.57     10.00     47     45505.99      <now>
cxiong36@submit-3.chtc.wisc.edu                   500.00   1000.00      0         0.29    0+00:09
ojalvo@hep.wisc.edu                               500.00   1000.00      0    398148.56    0+05:37
wjiang4@submit-3.chtc.wisc.edu                    500.00   1000.00      0         0.22    0+21:25
cxiong36@submit.chtc.wisc.edu                     500.00   1000.00      0        63.38    0+21:42
```

Different Type of Priority

- So far everything we saw was BETWEEN different users
- Individual users can also control the priorities and preferences WITHIN their own jobs

Schedd Policy: Job Priority

- Set in submit file with

  `JobPriority = 7`

- … or dynamically with `condor_prio` cmd
- Users can set priority of their own jobs
- Integers, larger numbers are better priority
- Only impacts order between jobs for a single user on a single schedd
- A tool for users to sort their own jobs

Schedd Policy: Job Rank

- Set in submit file with

```
RANK = Memory
```

- Not as powerful as you may think:
  - Remember steady state condition – there may not be that many resources to sort at any given time when pool is fully utilized.

Accounting Groups (2 kinds)

- Manage priorities across groups of users and jobs
- Can guarantee maximum numbers of computers for groups (quotas)
- Supports hierarchies
- Anyone can join any group (well…)

Accounting Groups as Alias

- In submit file
  - Accounting_Group = "group1"

- Treats all users as the same for priority
- Accounting groups not pre-defined
- No verification – HTCondor trusts the job
- condor_userprio replaces user with group

Prio factors with groups

```
condor_userprio -setfactor 10
group1@wisc.edu
condor_userprio -setfactor 20
group2@wisc.edu
```

Note that you must get UID_DOMAIN correct

Gives group1 members twice as many resources as group2

Accounting Groups w/ Quota

- Must be predefined in cm's config file:

```
GROUP_NAMES = a, b, c
GROUP_QUOTA_a = 10
GROUP_QUOTA_b = 20
```

- And in submit file:

```
Accounting_Group = a
Accounting_User = gthain
```

Group Quotas

- "a" limited to 10
- "b" to 20

- Must be predefined in cm's config file:

```
GROUP_NAMES = a, b, c
GROUP_QUOTA_a = 10
GROUP_QUOTA_b = 20
```

- And in submit file:

```
Accounting_Group = a
Accounting_User = gthain
```

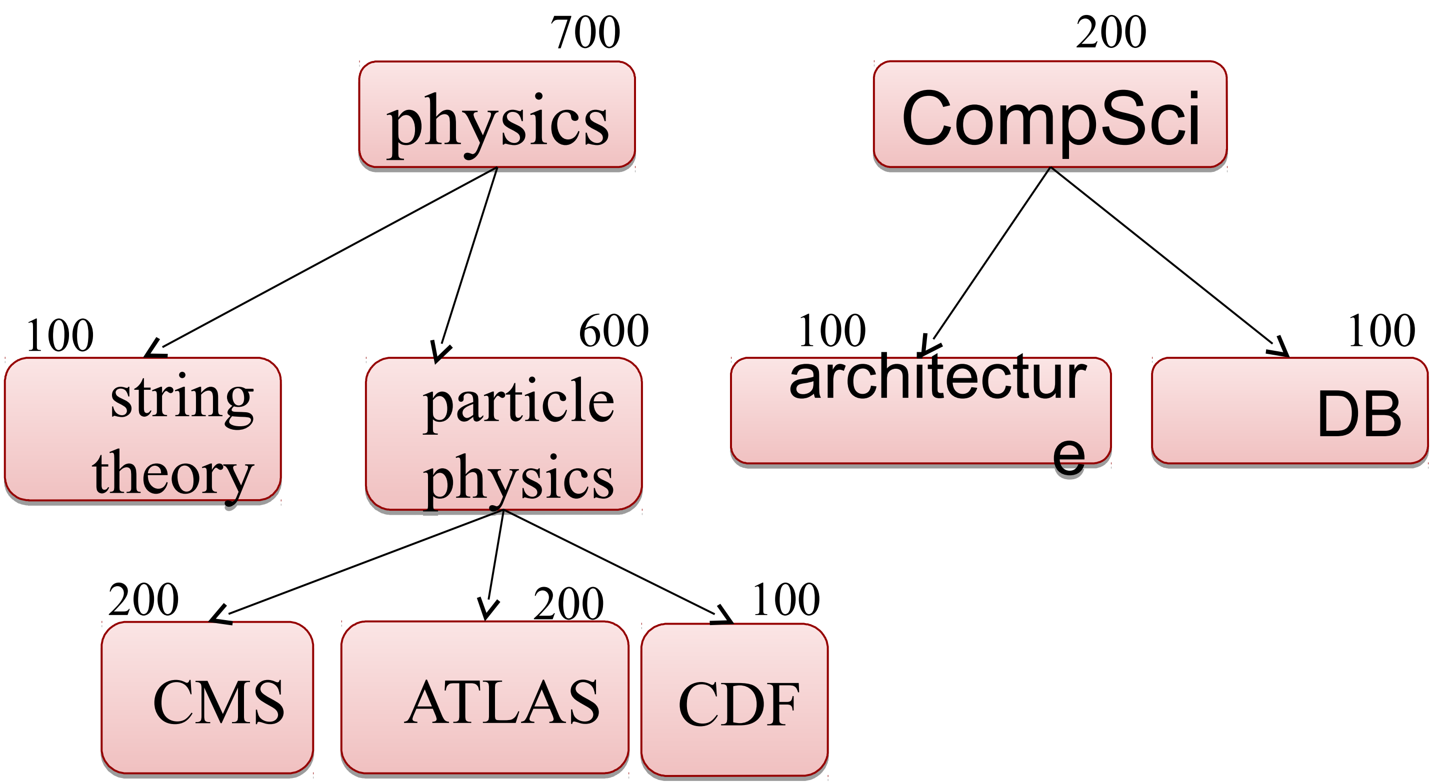
- Even if idle machines
- What is the unit?
  - Slot weight.
- With fair share for users within group

Hierarchical Group Quotas

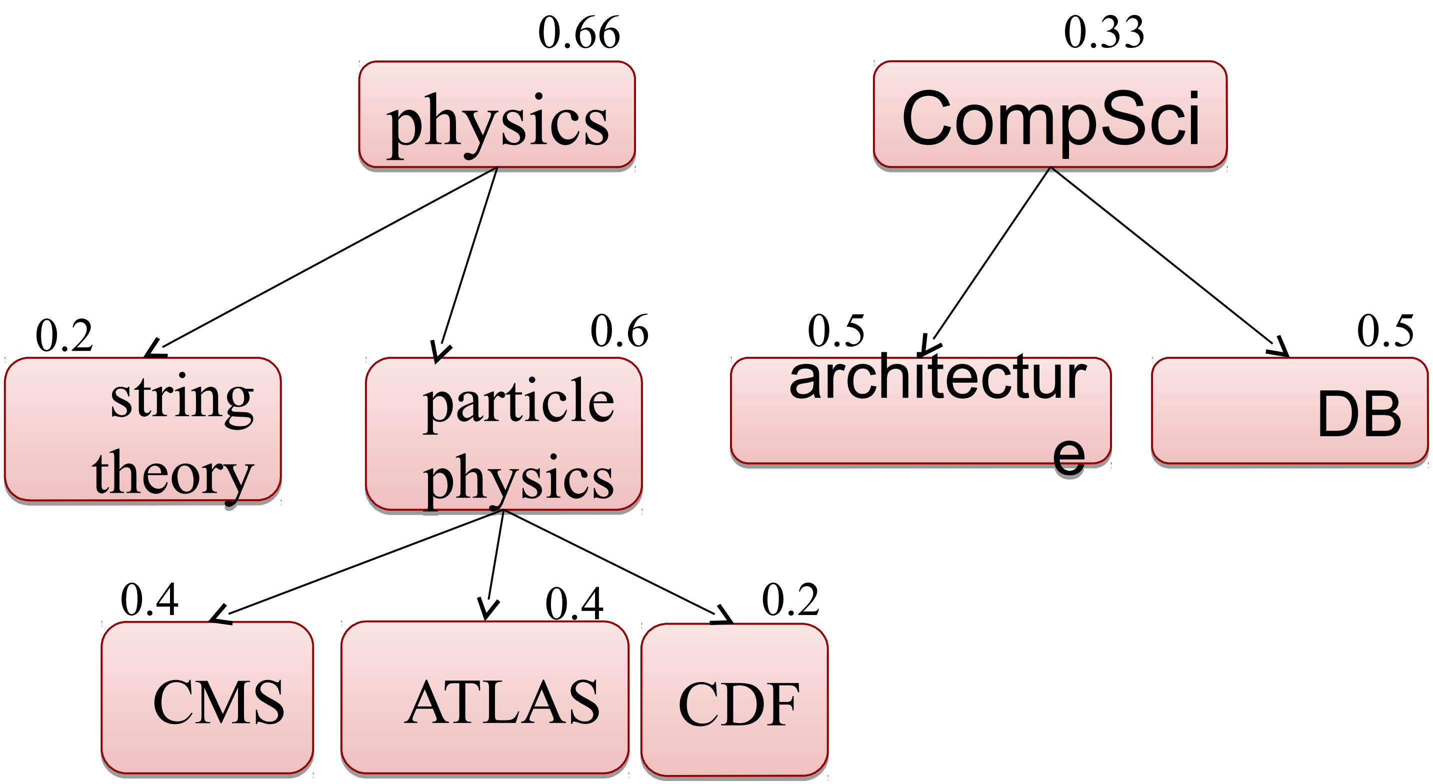
- Static versus Dynamic: Number of nodes versus proportion of the nodes

- Dynamic scales to size of pool.
- Static only “scales” if you oversubscribe your pool – HTCondor shrinks the allocations proportionally so they fit
  - This can be disabled in the configuration

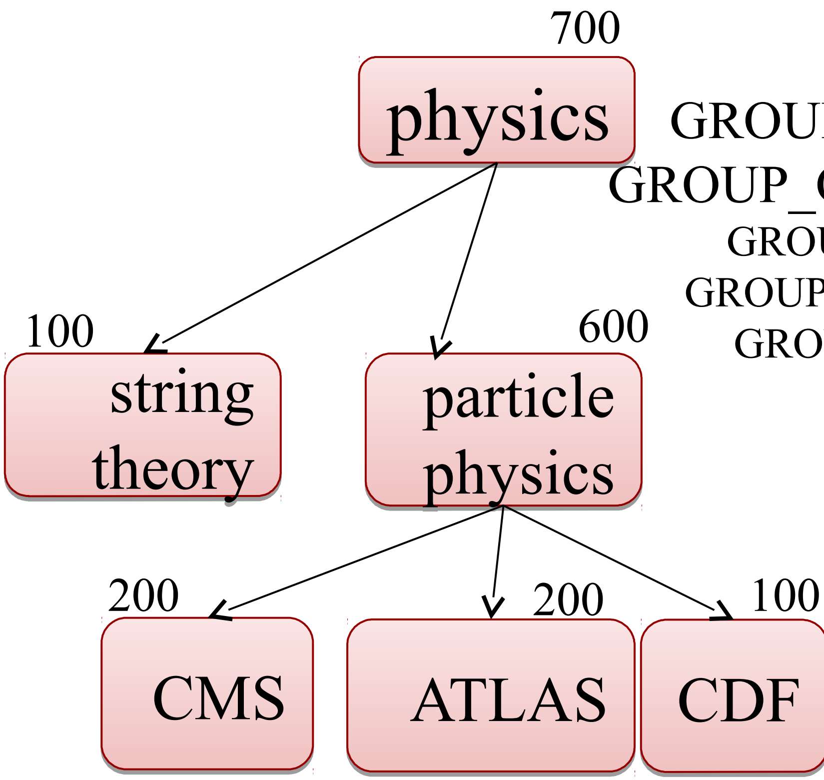
Hierarchical Group Quotas



Hierarchical Group Quotas



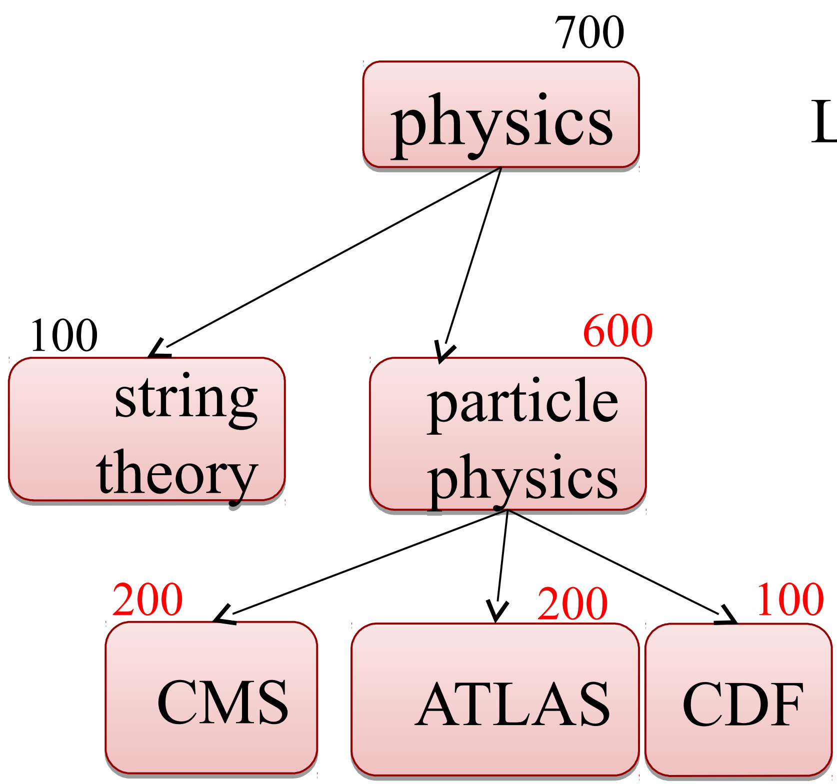
Hierarchical Group Quotas



GROUP_QUOTA_physics = 700
GROUP_QUOTA_physics.string_theory = 100
GROUP_QUOTA_physics.particle_physics = 600
GROUP_QUOTA_physics.particle_physics.CMS = 200
GROUP_QUOTA_physics.particle_physics.ATLAS = 200
GROUP_QUOTA_physics.particle_physics.CDF = 100

group.sub-group.sub-sub-group…

Hierarchical Group Quotas

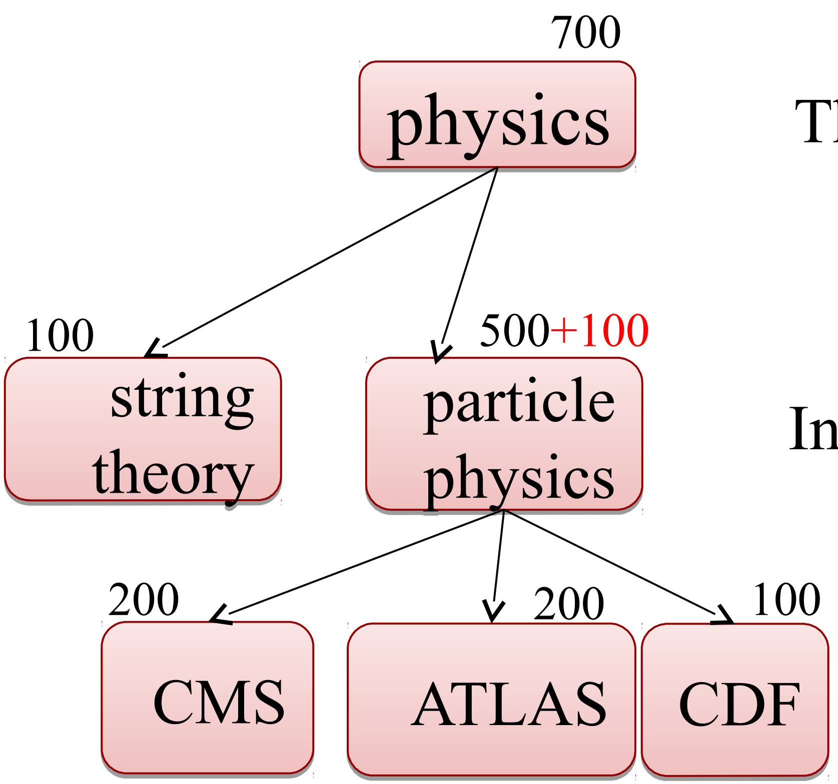


Look closely at the numbers in red

600 – (200 + 200 + 100) = 100

There are extra resources there… now what?

Hierarchical Group Quotas



There are 100 extra resources there

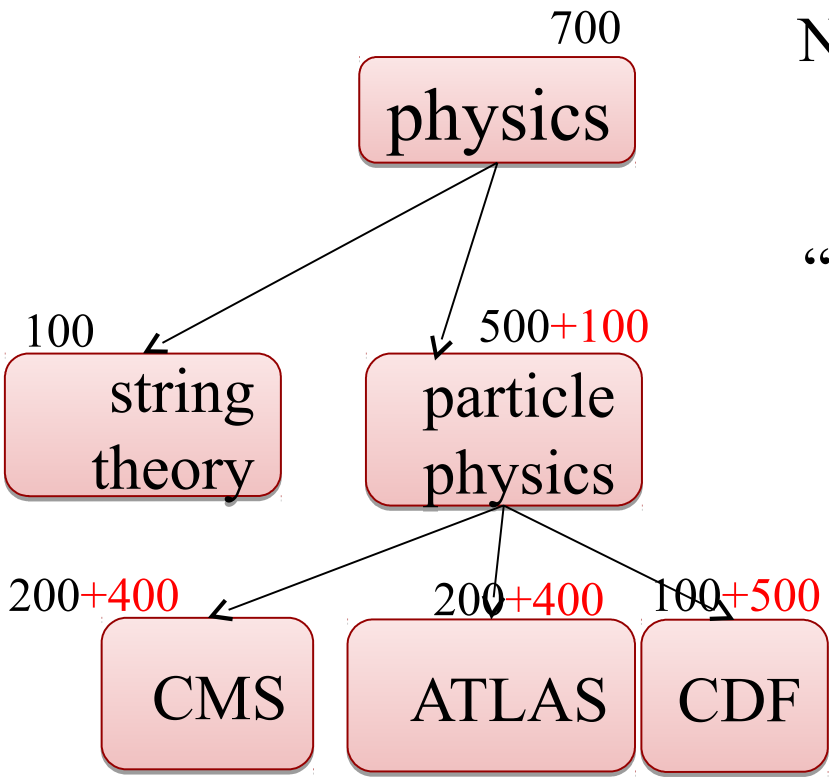
Who gets to use them?

In this case, only "particle physics" (not the children… quotas are still strictly enforced there)

GROUP_ACCEPT_SURPLUS

- Determines who can share extra resources
- Allows groups to go over quota if there are idle machines
- Creates the true hierarchy
- Defined per group, or subgroup, or sub-sub…

Hierarchical Group Quotas

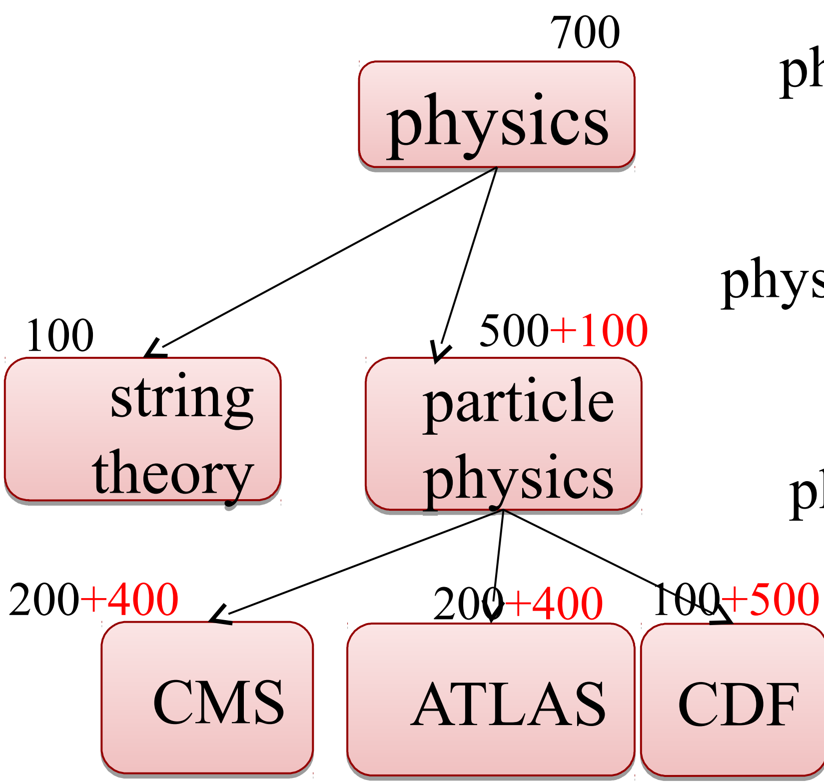


Numbers in RED are shared across the three children

"Particle physics" is still capped at 600 even if "string theory" is completely idle

CMS/ATLAS/CDF can now go over their quotas if the other groups have no jobs

Hierarchical Group Quotas



GROUP_ACCEPT_SURPLUS_physics.particle_physics.CMS = TRUE

GROUP_ACCEPT_SURPLUS_physics.particle_physics.ATLAS = TRUE

GROUP_ACCEPT_SURPLUS_physics.particle_physics.CDF = TRUE

GROUP_ACCEPT_SURPLUS_physics.particle_physics = TRUE

GROUP_AUTOREGROUP

- Also allows groups to go over quota if idle machines.
- “Last chance” round, with every submitter for themselves.

Enough with groups…

- We'll switch gears a little bit to talk about other pool-wide mechanisms that affect matchmaking…
- Welcome Jaime!

Rebalancing the Pool

- Match between schedd and startd can be reused to run many jobs
- May need to create opportunities to rebalance how machines are allocated
  - New user
  - Jobs with special requirements (GPUs, high memory)

How to Rematch

- Have startds return frequently to negotiator for rematching
  - `CLAIM_WORKLIFE`
  - Draining
  - More load on system, may not be necessary
- Have negotiator proactively rematch a machine
  - Preempt running job to replace with better job
  - `MaxJobRetirementTime` can minimize killing of jobs

A note about Preemption

- Fundamental tension between
  - Throughput vs. Fairness
- Preemption is required to have fairness

- Need to think hard about runtimes, fairness and preemption
- Negotiator implements preemption
- (Workers implement eviction: different)

Two Types of Preemption

- Startd Rank
  - Startd prefers new job
    - New job has larger startd Rank value
- User Priority
  - New job's user has higher priority (deserves increased share of the pool)
    - New job has lower user prio value
- No preemption by default
  - Must opt-in

Negotiation Cycle

- Gets all the slot ads
- Updates user prio info for all users
- Based on user prio, computes submitter limit for each user
- For each user, finds the schedd
  - For each job (up to submitter limit)
    - Finds all matching machines for job
    - Sorts the machines
    - Gives the job the best sorted machine

Sorting Slots: Sort Levels

- Single sort on a five-value key
  - `NEGOTIATOR_PRE_JOB_RANK`
  - Job `Rank`
  - `NEGOTIATOR_POST_JOB_RANK`
  - No preemption > Startd Rank preemption > User priority preemption
  - `PREEMPTION_RANK`

Negotiator Expression Conventions

- Evaluated as if in the machine ad
- `MY.Foo` : Foo in machine ad
- `TARGET.Foo` : Foo in job ad
- `Foo` : check machine ad, then job ad for Foo
- Use `MY` or `TARGET` if attribute could appear in either ad

Accounting Attributes

- Negotiator adds attributes about pool usage of job owners
- Info about job being matched
  - `SubmitterUserPrio`
  - `SubmitterUserResourcesInUse`
- Info about running job that would be preempted
  - `RemoteUserPrio`
  - `RemoteUserResourcesInUse`

Group Accounting Attributes

- More attributes when using groups
  - `SubmitterNegotiatingGroup`
  - `SubmitterAutoregroup`
  - `SubmitterGroup`
  - `SubmitterGroupResourcesInUse`
  - `SubmitterGroupQuota`
  - `RemoteGroup`
  - `RemoteGroupResourcesInUse`
  - `RemoteGroupQuota`

NEGOTIATOR_PRE_JOB_RANK

- `(10000000 * My.Rank) + (1000000 * (RemoteOwner=?=UNDEFINED)) - (100000 * Cpus) - Memory`
- Default
- Prefer machines that like this job more
- Prefer idle machines
- Prefer machines with fewer CPUs, less memory

NEGOTIATOR_POST_JOB_RANK

- `KFlops - SlotID`
- Prefer faster machines
- Breadth-first filling of statically-partitioned machines

If Matched machine claimed, extra checks required

- **`PREEMPTION_REQUIREMENTS`** and **`PREEMPTION_RANK`**
- Evaluated when **`condor_negotiator`** considers replacing a lower priority job with a higher priority job
- Completely unrelated to the **`PREEMPT`** expression (which should be called evict)

PREEMPTION_REQUIREMENTS

- If False, will not preempt for user priority
- Only replace jobs running for at least one hour and 20% lower priority

```
StateTimer = \
 (CurrentTime – EnteredCurrentState)
HOUR = (60*60)
PREEMPTION_REQUIREMENTS = \
 $(StateTimer) > (1 * $(HOUR)) \
 && RemoteUserPrio > SubmitterUserPrio * 1.2
```

NOTE: classad debug() function v. handy

Preemption with HQG

- Can restrict preemption to restoring quotas

```
PREEMPTION_REQUIREMENTS =
  ( SubmitterGroupResourcesInUse <
    SubmitterGroupQuota ) &&
  ( RemoteGroupResourcesInUse >
    RemoteGroupQuota )
```

PREEMPTION_RANK

- Of all claimed machines where PREEMPTION_REQUIREMENTS is true, picks which one machine to reclaim
- Strongly prefer preempting jobs with a large (bad) priority and less runtime

```
PREEMPTION_RANK = \
  (RemoteUserPrio * 1000000) \
 - TotalJobRuntime
```

No-Preemption Optimization

- `NEGOTIATOR_CONSIDER_PREEMPTION = False`
- Negotiator completely ignores claimed startds when matching
- Makes matching faster
- Startds can still evict jobs, then be rematched

Concurrency Limits

› Manage pool-wide resources
  - E.g. software licenses, DB connections

› In central manager config

› `FOO_LIMIT = 10`

› `BAR_LIMIT = 15`

› In submit file

› `concurrency_limits = foo,bar:2`

Summary

- Many ways to schedule