Agenda

- Understand role of negotiator
- Learn how priorities work
- Learn how quotas work
- Encourage thought about possible policies!
Overview of condor
3 sides

Submit

Central Manager

Execute
Startd Mission Statement

› Near sighted

› 3 inputs only:
  • Machine
  • Running Job
  • Candidate Running Job

› Knows nothing about the rest of the system!
Schedd mission

Run *jobs* on

*slots* the negotiator

has assigned to *submitters*.

Inputs:

All the jobs in that schedd

All the slots given to it by the negotiator
Schedd mission

Schedd Can:
- Re-use a slot for > 1 job (in succession)
- Pick which job for a user goes first

Schedd cannot:
- Reassign slots from one submitter to other
Submitter vs User

- Submitters: what are they?

- User: an OS construct

- Submitter: Negotiator construct
Negotiation Mission

Assign the slots of the whole pool to users based on some policy that’s ‘fair’
Negotiator Inputs

- All the slots in the pool
- All the submitters in the pool
- All the submitters’ priorities and quotas
- One request per submitter at a time
How the Negotiator Works

Periodically tries to:

*Rebalance* percentage of slots assigned to users
  
  Via preemption, if enabled
  
  Via assigning empty slots if not

_Negotiator is always a little out of date_
Concurrency Limits

- Simplest Negotiator (+ schedd) policy
- Useful for pool wide, across user limits,
Useful Concurrency Limits:

> 100 running NFS jobs crash my server

License server only allows X concurrent uses

Only want 10 database jobs running at once
Concurrence Limits: How to Configure

add to negotiator config file
(condor_reconfig needed):

```
NFS_LIMIT = 100
DB_LIMIT = 42
LICENSE_LIMIT = 5
```
Concurrency Limits: How to use

Add to job ad

Executable = somejob
Universe = vanilla
...
ConcurrencyLimits = NFS
queue
Concurreny Limits:
How to use

OR

Executable = somejob
Universe = vanilla
...
ConcurrencyLimits = NFS:4
queue
Concurrency Limits: How to use

Add to job ad

Executable = somejob
Universe = vanilla
...
ConcurrencyLimits = NFS, DB
queue
Part of the picture

- Concurrency limits very “strong”
- Can throw off other balancing algorithms
- No “fair share” of limits
“Fair Share of Users”
Main Loop of Negotiation Cycle*

1. Get all slots in the pool
2. Get all jobs submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed
The Negotiator as Shell Script

1. Get all slots in the pool
2. Get all jobs submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed
1: Get all slots in pool
1: Get all slots in pool

$ condor_status
1: Get all slots* in pool

NEGOTIATOR_SLOT_CONSTRAINT = some classad expr

NEGOTIATOR SLOT CONSTRAINT

Defaults to true, what subset of pool to use
For sharding, etc.
1: Get all slots in pool

```
$ condor_status -af Name State RemoteOwner

slot1@... Claimed Alice
slot2@... Claimed Alice
slot3@... Claimed Alice
slot4@... Unclaimed undefined
slot5@... Claimed Bob
slot6@... Claimed Bob
slot7@... Claimed Charlie
slot8@... Claimed Charlie
```
1: Get all slots in pool

$ condor_status -af Name RemoteOwner

![Pie chart showing slots分配: Alice, Bob, Charlie, Unclaimed]
2: Get all submitters in pool

$ condor_status -submitters
# 2: Get all submitters in pool

```bash
$ condor_status -submitters
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Machine</th>
<th>RunningJobs</th>
<th>IdleJobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>submit1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>submit1</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>Charlie</td>
<td>submit1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Danny</td>
<td>submit1</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>
### 2: Get all submitters in pool

```bash
$ condor_status -submitters
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Machine</th>
<th>RunningJobs</th>
<th>IdleJobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>submit1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>submit1</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>Charlie</td>
<td>submit1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Danny</td>
<td>submit1</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>
3: Compute per-user “share”

- Tricky
- Based on historical usage
3a: Get historical usage

$ condor_userprio -all
3a: Get historical usage

```bash
$ condor_userprio -all
```

<table>
<thead>
<tr>
<th>UserName</th>
<th>Effective Priority</th>
<th>Real Priority</th>
<th>Priority Factor</th>
<th>Res in use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3100</td>
<td>3.1</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>4200</td>
<td>4.2</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Charlie</td>
<td>1500</td>
<td>1.5</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Danny</td>
<td>8200</td>
<td>8.2</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
3a: Get historical usage

**Effective Prio** = Real Prio \* Prio Factor

<table>
<thead>
<tr>
<th>UserName</th>
<th>Effective Priority</th>
<th>Real Priority</th>
<th>Priority Factor</th>
<th>Res in use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3100</td>
<td>3.1</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>4200</td>
<td>4.2</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Charlie</td>
<td>1500</td>
<td>1.5</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Danny</td>
<td>8200</td>
<td>8.2</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
So What is Real Priority?

Real Priority is smoothed historical usage
Smoothed by `PRIORITY_HALFLIFE`
`PRIORITY_HALFLIFE` defaults 86400s (24h)
Another PRIORITY_HALFLIFE

PRIORITY_HALFLIFE = 1
## 3a: Get historical usage

```bash
$ condor_userprio -all
```

<table>
<thead>
<tr>
<th>UserName</th>
<th>Effective Priority</th>
<th>Real Priority</th>
<th>Priority Factor</th>
<th>Res in use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3100</td>
<td>3.1</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>4200</td>
<td>4.2</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Charlie</td>
<td>1500</td>
<td>1.5</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Danny</td>
<td>8200</td>
<td>8.2</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
Effective Priority is the *ratio* of the pool that the negotiator tries to allot to *users*. Lower is better, 0.5 is the best real priority.
<table>
<thead>
<tr>
<th>UserName</th>
<th>Effective Priority</th>
<th>Real Priority</th>
<th>Priority Factor</th>
<th>Res in use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1000</td>
<td>1.0</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>2000</td>
<td>2.0</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>Charlie</td>
<td>2000</td>
<td>2.0</td>
<td>1000</td>
<td>2</td>
</tr>
</tbody>
</table>

Alice deserves 2x Bob & Charlie

Alice: 4
Bob: 2
Charlie: 2 (Assuming 8 total slots)
Priority factor lets admin say
If equal usage, User A gets 1/nth User B

$ condor_userprio -setfactor alice 5000
3 different PrioFactors
Whew! Back to negotiation

1. Get all slots in the pool
2. Get all jobs submitters in pool
3. Compute # of slots submitters should get
4. In priority order, hand out slots to submitters
5. Repeat as needed
### Target allocation from before

<table>
<thead>
<tr>
<th>User</th>
<th>Effective Priority</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>2</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
</tr>
</tbody>
</table>

Assume 8 total slots (claimed or not)
Look at current usage

<table>
<thead>
<tr>
<th>User</th>
<th>Effective Priority</th>
<th>Goal</th>
<th>Current Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>User</td>
<td>Effective Priority</td>
<td>Goal</td>
<td>Current Usage</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------</td>
<td>------</td>
<td>---------------</td>
</tr>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>User</td>
<td>Effective Priority</td>
<td>Goal</td>
<td>Current Usage</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------</td>
<td>------</td>
<td>---------------</td>
</tr>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Limits determined, matchmaking starts

In Effective User Priority order,
Find a schedd for that user, get the request

<table>
<thead>
<tr>
<th>User</th>
<th>Effective Priority</th>
<th>Difference (“Limit”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
</tr>
</tbody>
</table>
"Requests", not "jobs"

```
$ condor_q -autocluster Alice
Id   Count  Cpus Memory Requirements
20701 10    1   2000 OpSys == "Linux"
20702 20    2   1000 OpSys == "Windows"
```
Match all machines to requests

<table>
<thead>
<tr>
<th>Id</th>
<th>Count</th>
<th>Cpus</th>
<th>Memory Requirements</th>
<th>OpSys == “Linux”</th>
</tr>
</thead>
<tbody>
<tr>
<td>20701</td>
<td>10</td>
<td>1</td>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

| slot1@... Linux | X86_64 | Idle | 2048 |
| slot2@... Linux | X86_64 | Idle | 2048 |
| slot1@... Linux | X86_64 | Idle | 1024 |
| slot2@... Linux | X86_64 | Claimed | 2048 |
| slot1@... WINDOWS | X86_64 | Claimed | 1024 |
Sort All matches

By 3 keys, in order

NEGOTIATOR_PRE_JOB_RANK
RANK
NEGOTIATOR_POST_JOB_RANK
Why Three?

NEGOTIATOR_PRE_JOB_RANK

Strongest, goes first over job RANK

RANK

Allows User some say

NEGOTIATOR_POST_JOB_RANK

Fallback default
Finally, give matches away!

<table>
<thead>
<tr>
<th>slot1@... Linux</th>
<th>X86_64</th>
<th>Unclaimed</th>
<th>2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot2@... Linux</td>
<td>X86_64</td>
<td>Unclaimed</td>
<td>2048</td>
</tr>
<tr>
<td>slot1@... Linux</td>
<td>X86_64</td>
<td>Claimed</td>
<td>2048</td>
</tr>
</tbody>
</table>

Up to the limit specified earlier
If below limit, ask for next job request
Done with Alice, on to Bob

<table>
<thead>
<tr>
<th>User</th>
<th>Effective Priority</th>
<th>Difference (“Limit”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td>2</td>
</tr>
</tbody>
</table>
But, it isn’t that simple…

› Assumed every job matches every slot
  And infinite supply of jobs!

› … But what if they don’t match?

There will be leftovers – then what?
Lather, rinse, repeat

This whole cycle repeats with leftover slots

Again in same order…
Big policy question

- Preemption: Yes or no?
- Tradeoff: fairness vs. throughput
- (default: no preemption)
Preemption: disabled by default

PREEMPTION_REQUIREMENTS = false

Evaluated with slot & request ad. If true, Claimed slot is considered matched, and Subject to matching
Example PREEMPTION_REQs

PREEMPTION_REQUIREMENTS=\n  RemoteUserPrio > SubmittorPrio * 1.2
PREEMPTION_RANK

- Sorts matched preempting claims

PREEMPTION_RANK = -TotalJobRunTime
MaxJobRetirementTime

- Can be used to guarantee minimum time
- E.g. if claimed, give an hour runtime, no matter what:
  - MaxJobRetirementTime = 3600
- Can also be an expression
Whew!

Now, on to Groups.
AccountingGroup as alias
Accounting_Group_User = Ishmael
“Call me Ishmael”
With no dots, and no other configuration
Means alias: Maps “user” to “submitter”
Complete trust in user job ad (or xform)
  • Viz-a-vis SUBMIT_REQUIREMENTS
<table>
<thead>
<tr>
<th>User</th>
<th>Effective Priority</th>
<th>Accounting Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1,000.00</td>
<td>“Alice”</td>
</tr>
<tr>
<td>Bob</td>
<td>2,000.00</td>
<td>“Alice”</td>
</tr>
<tr>
<td>Charlie</td>
<td>2,000.00</td>
<td></td>
</tr>
</tbody>
</table>

Merged to one user

No fair share between old Alice and old Bob!
Accounting Groups With Quota

Only way to get “quotas” for users or groups
quota, n.

Pronunciation: Brit. /ˈkwɔːta/, U.S. /ˈkwʌʊðə/
Forms:
α. 16–quota, 16–quoto (chiefly U.S. regional), 17 cottā, 17 gottā.

Dictionaries: < post-classical Latin quota...

1.

a. Originally: the part or share which an individual is obliged to contribute to a total amount (in early use chiefly with reference to contributions of men, money, or supplies from a particular town, district, or country; cf. CONTINGENT n. 5). Later more widely: an amount contributed to a larger quantity.

1618–1968

b. Econ. A maximum quantity of a particular product which, under official controls, can be produced, exported, imported, or caught. Also: a target setting a minimum production for a particular factory, employee, etc.
Minimum

2. a. A share of a larger number or quantity; a portion, an allocation.

b. Polit. In a system of proportional representation: the minimum number of votes required to elect a candidate.

3. Chiefly U.S.
   a. A maximum number of immigrants allowed to enter a country within a set period. Also: a maximum number of students (as of a particular racial or ethnic group) allowed to enrol for a course at a college, etc., in a particular year.

   The Emergency Quota Act was passed by the U.S. Congress in 1921.

b. A minimum number or proportion (of racial or ethnic minorities, or women) sought in order to ensure a desired balance in a workforce, student body, etc.
Accounting Groups with quotas

Must be predefined in config file

GROUP_NAMES = group_a, group_b
GROUP_QUOTA_GROUP_A = 10
GROUP_QUOTA_GROUP_B = 20

Slot weight is the unit – default cpus
Or, with Dynamic quotas

- Can also be a percentage

GROUP_NAMES = group_a, group_b
GROUP_QUOTA_GROUP_A = 0.3
GROUP_QUOTA_GROUP_B = 0.4

If sum != 100, scaled
And jobs opt in (again)

Accounting_Group = group_a

But you retain identity within your group.
AcctGroups w/quota

Reruns the whole cycle as before

• But with pool size constrained to quota
• And fair share, between users in group
Order of groups?

- By default, in starvation order
- Creates overprovisioning trick for strict fifo:
  - `GROUP_QUOTA_HIPRIO = 100000000`
- Means this group always most starving
  - `GROUP_SORT_EXPR` overrides
“Not” strict quotas

One way is:

GROUP_AUTO_REGROUP = true

After all groups go, one last round with no groups, every user outside of their group.
“Surplus”

Assumes a hierarchy of groups:

GROUP_NAMES = group_root, group_root.a, group_root.b, group_root.c
GROUP_QUOTA_GROUP_root = 60
GROUP_QUOTA_GROUP_root.a = 10
GROUP_QUOTA_GROUP_root.b = 20
GROUP_QUOTA_GROUP_root.b = 30
GROUP_ACCEPT_SURPLUS = true
How “Surplus” works

Before **matchmaking**

Assume all jobs match all slots,

- See if there will be leftover slots
- If so, “loan” leftover slots to nearest group that accepts surplus
accept_surplus = true

3 slots of demand at A

7 quota slots moved to B & C

Proportional to B & C quota
accept_surplus = true
Gotchas with quotas

- Quotas don’t know about matching
- Assuming everything matches everything
- Surprises with partitionable slots
- Managing groups not easy
In summary

- Negotiator is very powerful, often ignored
- Lots of opportunity to tune system
- Many ways to peak under the hood
Thank you

- Questions?
- Talk to us
- htcondor-users
- manual