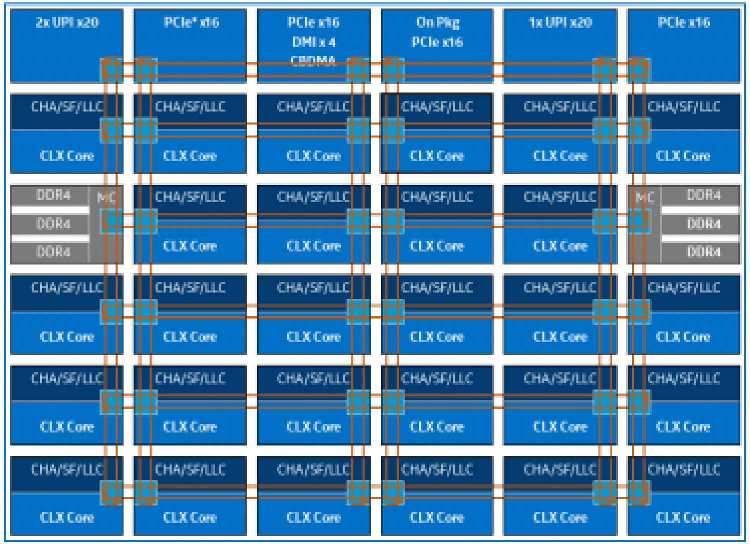
Managing multicore machines: pslots, draining and more

Center for High Throughput Computing

HELP! There's a Cluster in my Server!

Center for High Throughput Computing

2nd gen Intel® Xeon® Scalable Processors



[28 core die]

Features	2nd gen Intel® Xeon® Scalable Processors
Cores and Threads Per Processor	[8200] Up to 28 cores and 56 threads **[9200] Up to 56 cores and 112 threads**
Data Center Cache Hierarchy	1MB dedi[illegible] Cache per Core Up to 38.5 MB ([illegible]usive) Shared L3 Cache
Cross-Die Interconnect	Up to [illegible] UPI @ 10.4 GT/s Up to [illegible]cket glueless connectivity
PCIe Lanes	Up [illegible] Lanes PCIe 3.0 (2.5, 5, 8 GT/s)
Memory	Up [illegible] channels **at 2933 MT/s** per processor [illegible] 2 DPC RDIMMs, LRDIMMs, 3DS LRDIMMs, **[illegible]upporting up to 16Gb DDR4 devices** **Intel® Optane™ DC Persistent Memory Module support for up to 4.5TB system memory per processor**
Vector Compute	Intel® AVX-512 with up to 16 DP, 32 SP, **and 128 INT8 MACs w/ Intel® DL Boost** per cycle per core
Mitigations for Side-Channel Methods	**Variants 2, 3, 3a, 4, and L1TF**
Turbo Frequency	**Boost across Stack**

My Desktop Computer

8 Cores (Hyper-threaded)
16 Gigabytes RAM

HTCondor out of the box

1 core per slot, memory divided evenly

Create static slots from detected resources

```
$ condor_status
Name      OpSys       Arch   State     Activity LoadAv Mem   ActvtyTime
slot1@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:00
slot2@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot3@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot4@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot5@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot6@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot7@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20
slot8@c LINUX       X86_64 Unclaimed Idle      0.000 1997  0+00:00:20

             Total   Owner Claimed Unclaimed Matched Preempting Backfill  Drain
X86_64/LINUX      8      0       0         8       0          0        0      0
Total             8      0       0         8       0          0        0      0
```

You can Lie about Resources!

```
# HTCondor Config file for startd

# Tell HTCondor I've got 6 cores
NUM_CPUS = 6

# Memory for all slots (in Megabytes)
MEMORY = 4096

# Subtract 1GB from whatever memory is detected
RESERVED_MEMORY = 1024

# Tell HTCondor that execute disk size (in KB) is this
DISK = 10240
```

Conventional Wisdom about cores

“Modern machines have lots of cores, so I should make all my jobs each use as many cores as possible, so they finish as fast as possible”

HT Wisdom: “Probably not”

Rules of HT Optimization

The fewer resources a job needs,

the more places it can run

But sometimes, you just need more…

Slot are where jobs run

```
$ con
Name
slot1
slot2
slot3
slot4
slot5
slot6
slot7
slot8

X86_6
Total
```

```
# Submit file
Executable = calculate
Arguments = 1 2 42

Request_Cpus = 2
Request_Memory = 2048
Request_Disk    = 1G

Log = log
queue
```

Don't Forget These – may be required!

What happens if…

```
# Submit file
Executable = calculate
Arguments = 1 2 42

Request_Cpus = 2
Request_Memory = 2048
Request_Disk    = 1G

Log = log
queue
```

There are no 2 core slots?

Idle Forever...

```
$ condor_q
-- Schedd: gthain@c: <10.5.1.1:33601?...

OWNER   BATCH_NAME      SUBMITTED    DONE    RUN     IDLE   TOTAL JOB_IDS
gthain ID: 577        9/16 17:57        _       _       1       1 577.0

Total for query: 1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0
suspended      Total for all users: 1 jobs; 0 completed, 0 removed, 1 idle, 0
running, 0 held, 0 suspended
```

What if I lie to HTCondor?

```
# Submit file
Executable = calculate
Arguments = 1 2 42

Request_Cpus = 1
Request_Memory = 1024
Request_Disk    = 1G

Log = log
queue
```

And it really needs 2 cores and 4 Gb of memory?

Running, but in Jail!

```
$ condor_q
-- Schedd: gthain            3601?...

OWNER   BATCH_NAM       UBMIT         RUN     ID        TAL JOB_IDS
gthain ID: 577          16 17:5                 1                 1 577.0

Total for query:       s; 0 complete          ved,            0 running, 0 held, 0
suspended       To        r all users: 1                     , 0 removed, 1 idle, 0
running, 0 held,          ded
```

Solution: Partitionable Slots

2 kinds of slots:

partitionable-slots:

always unclaimed

hold unused resources

Dynamic slots

create/destroyed to fit one job

Enabling partionable slots

```
# HTCondor Config file for startd

NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1_PARTITIONABLE = true
SLOT_TYPE_1 = cpus=100%
```

Enabling partionable slots

```
$ condor_status
Name                  OpSys       Arch   State     Activity LoadAv Mem    ActvtyTime

slot1@c               LINUX       X86_64 Unclaimed Idle      0.000 15976  0+08:46:59

                  Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

  X86_64/LINUX      1     0       0         1       0          0        0      0

         Total      1     0       0         1       0          0        0      0
```

What's in the p-slot?

```
$ condor_status -af Name SlotType Cpus Memory

slot1@c Partitionable 8 15976
```

Now this job can run…

```
# Submit file
Executable = calculate
Arguments = 1 2 42

Request_Cpus = 2
Request_Memory = 2048
Request_Disk    = 1G

Log = log
queue
```

This didn't fit with static slots

And this one...

```
# Submit file
Executable = calculate
Arguments = 1 2 42

Request_Cpus = 3
Request_Memory = 1024
Request_Disk    = 1G

Log = log
queue
```

What’s in the p-slot?

```
$ condor_status
Name       OpSys       Arch   State     Activity LoadAv Mem
slot1@c    LINUX       X86_64 Unclaimed Idle      0.000 12904
slot1_1@c LINUX        X86_64 Claimed   Busy      0.000  2048
slot1_2@c LINUX        X86_64 Claimed   Busy      0.000  1024
slot1_3@c LIN          X86_64 Claimed   Busy       .000  1024

Total Owner Claimed Unc    ed Matched   empting Backfill  Drain
LINUX     4      0      3                      0        0      0
Total     4      0      3                      0
```

Note non-uniform Memory sizes

Note the underscore in d-slots

What's in the p-slot?

```
$ condor_status -af Name SlotType Cpus Memory
slot1@c    Partitionable 0 11880
slot1_1@c Dynamic 2 2048
slot1_2@c Dynamic 3 1024
slot1_3@c Dynamic 3 1024
```

When a d-slot completes?

```
$ condor_status -af Name SlotType Cpus Memory
slot1@c   Partitionable 3 12904
slot1_1@c Dynamic 2 2048
~~slot1_2@c Dynamic 3 1024~~
slot1_3@c Dynamic 3 1024
```

Are we good?

No – Starvation

If I submit 8 one core jobs

Completely used machine

```
$ condor_status
Name        OpSys       Arch    State     Activit      v Mem

slot1@c     LINUX       X86_64 Unclaimed              0.000 7784
slot1_1@c LINUX       X86_64 Claim                   0.000 1024
slot1_2@c LINUX       X86_64 Cl           sy          0.000 1024
slot1_3@c LINUX       X86                  Busy        0.000 1024
slot1_4@c LINUX       X            med     Busy        0.000 1024
slot1_5@c LINUX                 laimed     Busy        0.000 1024
slot1_6@c LIN             64 Claimed     Busy        0.000 1024
slot1_7@c L              X86_64 C                            24
slot1_8@c L              X86_64 C                            24

          Total  Owner Claimed                      g Backfill
LINUX       9      0                                      0

Total       9      0       8                               0
```

When can a 2-core job run?

Note total slots!

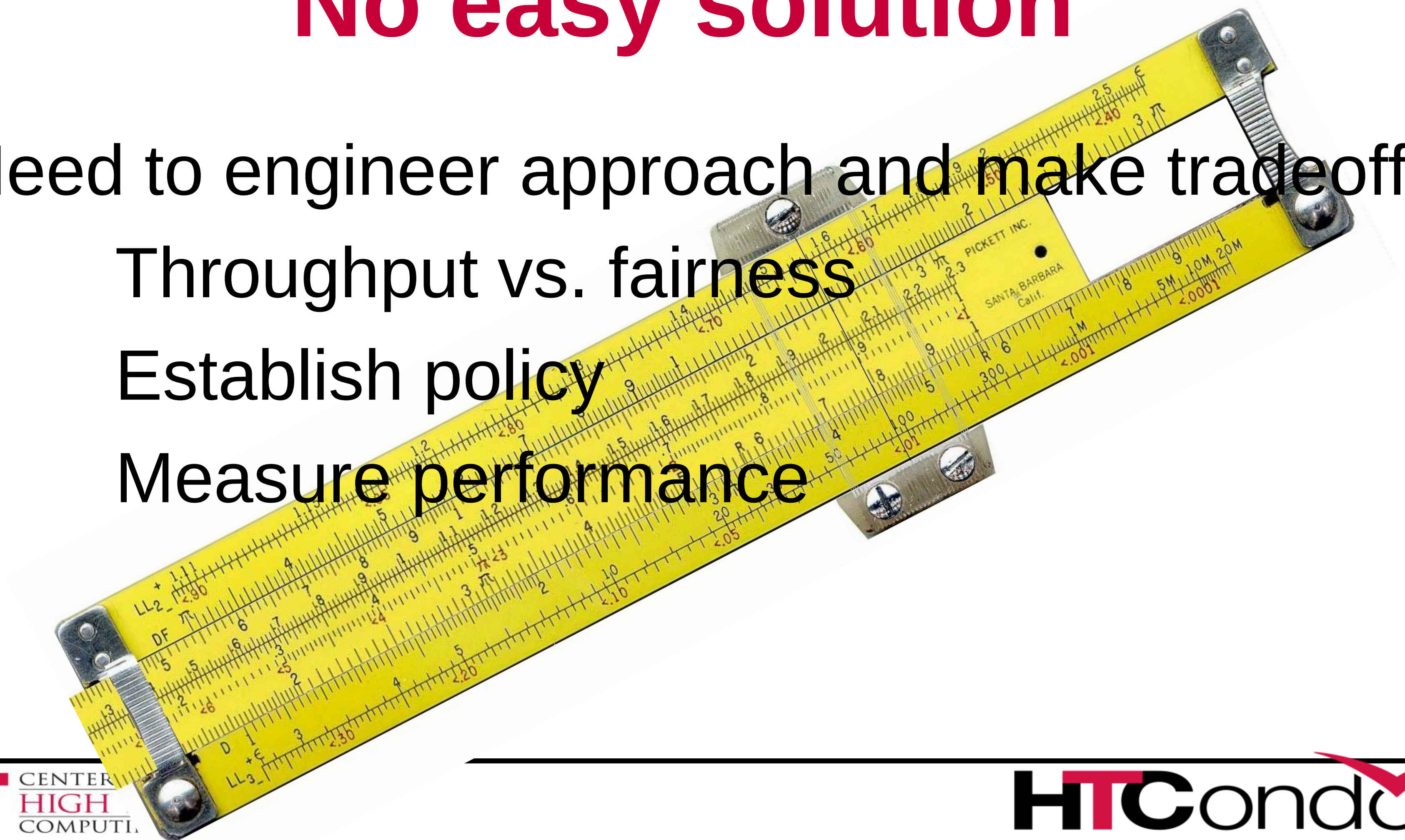
Starvation

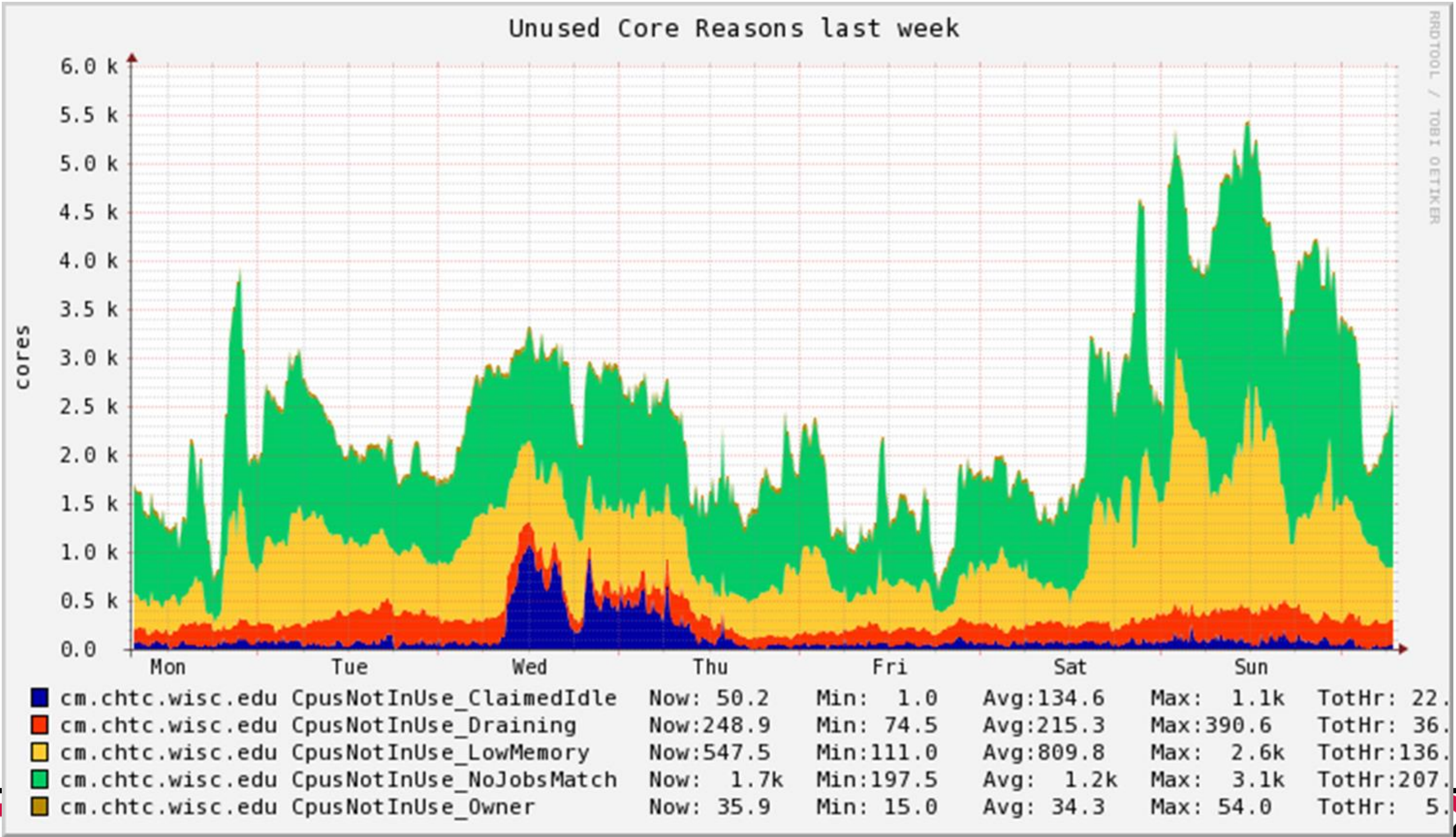
- If there is a supply of one core jobs…
- A two-core job will never match!

No easy solution

Need to engineer approach and make tradeoffs

- Throughput vs. fairness
- Establish policy
- Measure performance



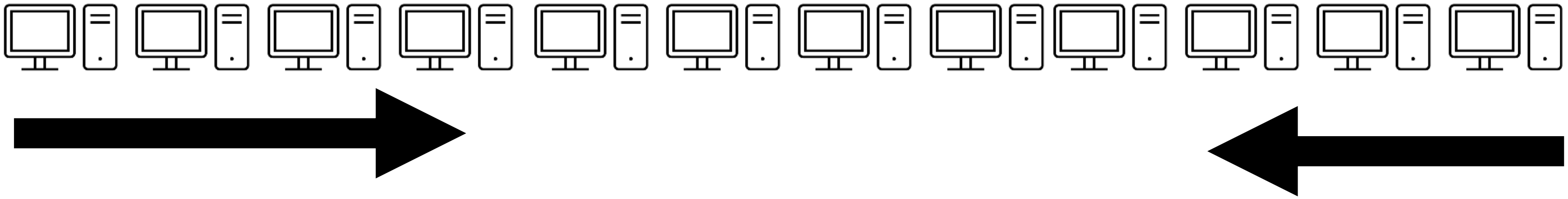


Unused Core Reasons last week
RRDTOOL / TOBI OETIKER
cores
6.0 k
5.5 k
5.0 k
4.5 k
4.0 k
3.5 k
3.0 k
2.5 k
2.0 k
1.5 k
1.0 k
0.5 k
0.0
Mon Tue Wed Thu Fri Sat Sun
cm.chtc.wisc.edu CpusNotInUse_ClaimedIdle Now: 50.2 Min: 1.0 Avg:134.6 Max: 1.1k TotHr: 22.
cm.chtc.wisc.edu CpusNotInUse_Draining Now:248.9 Min: 74.5 Avg:215.3 Max:390.6 TotHr: 36.
cm.chtc.wisc.edu CpusNotInUse_LowMemory Now:547.5 Min:111.0 Avg:809.8 Max: 2.6k TotHr:136.
cm.chtc.wisc.edu CpusNotInUse_NoJobsMatch Now: 1.7k Min:197.5 Avg: 1.2k Max: 3.1k TotHr:207.
cm.chtc.wisc.edu CpusNotInUse_Owner Now: 35.9 Min: 15.0 Avg: 34.3 Max: 54.0 TotHr: 5.

COMPUTING

First approach: Steering

Works best when 2 sizes (cpus)



Fill 1 core jobs
“left to right”

Fill 8 core jobs
“right to left”

Pool-wide policy -> negotiator

Assume every startd advertises Longitude:

```
Longitude = 2.3522
START_ATTRS = Longitude
```

And in the negotiator, have config like…

```
NEGOTIATOR_PRE_JOB_RANK = \
  RequestCpus == 1 ? Longitude : - Longitude
```

Brief Advertisement

- Whole talk on negotiator
- Google for
  - site:youtube.com center for high throughput computing channel

2nd approach: Draining

```
$ condor_status
Name         OpSys        Arch    State      Activity LoadAv Mem
slot1@c      LINUX        X86_64 Unclaimed Idle        0.000 12904
slot1_1@c LINUX        X86_64 Claimed    Busy        0.000  2048
slot1_2@c LINUX        X86_64 Claimed    Busy        0.000  1024
slot1_3@c LINUX        X86_64 Claimed    Busy        0.000  1024

Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain
LINUX      4      0        3         1        0          0        0
Total      4      0        3         1        0          0        0
```

condor_drain command

```
$ condor_drain c
```

```
$ condor_status
Name        OpSys                        LoadAv Mem
slot1@c     LINUX                        0.000 12904
slot1_1@c LINUX                          0.000  2048
slot1_2@c LINUX
slot1_3@c LINUX      X86_64 Claimed    R

Total Owner Claimed Unclaimed Matched Pre
LINUX       4      0         3          1
Total       4      0         3          1
```

Need admin privs,
May need to run
from cm machine

Typo:
Should be
"Drain**ING**"

CENTER FOR HIGH THROUGHPUT COMPUTING

HTCondor

How does drain work?

```
$ condor_status -af:r Requirements
false
false
false
false
```

condor_drain cancelling

```
$ condor_drain -cancel c
```

```
$ condor_status
Name         OpSys       Arch   State       Activity LoadAv Mem
slot1@c      LINUX       X86_64 Unclaimed    Retiring  0.000 12904
slot1_1@c    LINUX       X86_64 Claimed     Retiring  0.000  2048
slot1_2@c    LINUX       X86_64 Claimed     Retiring  0.000  1024
slot1_3@c    LINUX       X86_64 Claimed     Retiring  0.000  1024

Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain
LINUX     4     0       3         1       0          0        0
Total     4     0       3         1       0          0        0
```

How long does drain last?

- Jobs killed after MaxJobRetirementTime
  - Default is 0
- This may be most underused knob in HTC
- Machine stays in drained state until cancel
  - Even after all jobs exit
- (unless –resume-on-complete) is set

condor_drain command

```
$ condor_drain -start 'BackfillableJob == true' c
```

```
$ condor_status
Name        OpSys       Arch   State      Activity LoadAv Mem
slot1@c     LINUX       X86_64 Drained    Retiring  0.000 12904
slot1_1@c LINUX       X86_64 Claimed    Retiring  0.000  2048
slot1_2@c LINUX       X86_64 Claimed    Retiring  0.000  1024
slot1_3@c LINUX       X86_64 Claimed    Retiring  0.000  1024

Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain
LINUX     4     0        3         1       0          0        0
Total     4     0        3         1       0          0        0
```

condor_drain with backfill

```
$ condor_submit backfillable_job.sub
```

```
$ condor_status
Name       OpSys      Arch   State    Activity LoadAv Mem
slot1@c    LINUX      X86_64 Drained  Retiring  0.000 12904
slot1_1@c  LINUX      X86_64 Claimed  Retiring  0.000  2048
slot1_2@c  LINUX      X86_64 Claimed  Retiring  0.000  1024
slot1_3@c  LINUX      X86_64 Claimed  Retiring  0.000  1024
slot1_4@c  LINUX      X86_64 Claimed  Busy      0.000  1024

Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain
LINUX     4      0        3         1        0         0      0
Total     4      0        3         1        0         0      0
```

Defrag daemon

- Optional, can be enabled (often on CM)
- Just runs condor_drain and –cancel
- Never looks at queues, just at condor_status

```
DEFRAG_DRAINING_MACHINES_PER_HOUR
DEFRAG_MAX_WHOLE_MACHINES
DEFRAG_REQUIREMENTS
```

Thank you

Questions?

Please see htcondor.readthedocs.io