Can we break HTCondor before Run 4 Does?

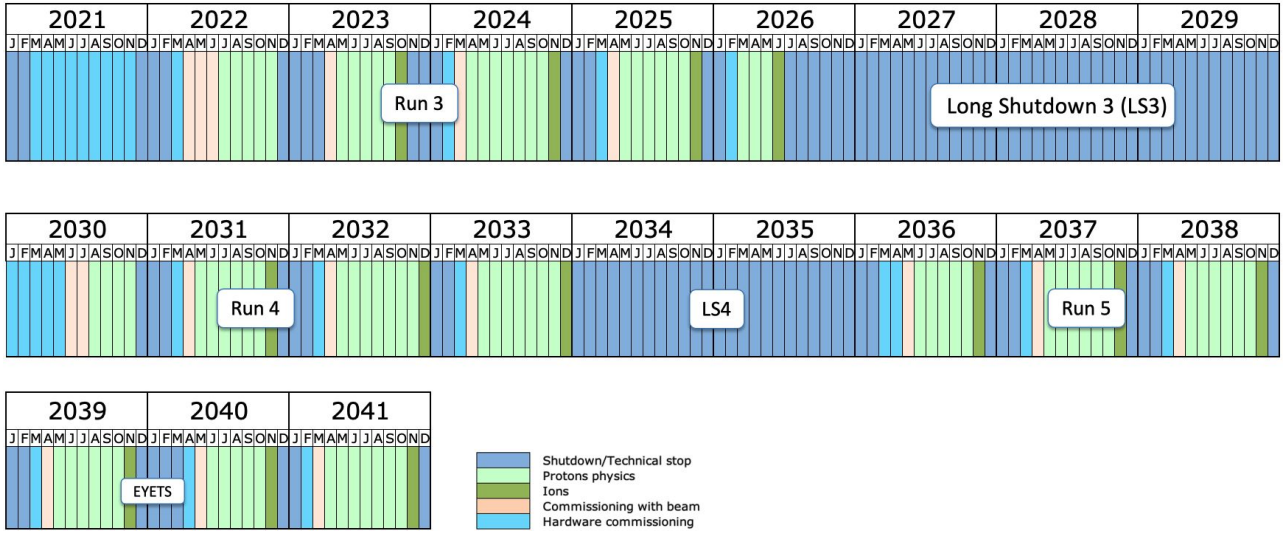
Scale testing HTCondor for Hi-Lumi LHC

Antonio Delgado Peris, Ben Jones, Panagiotis Gknois

High-luminosity LHC

HL-LHC

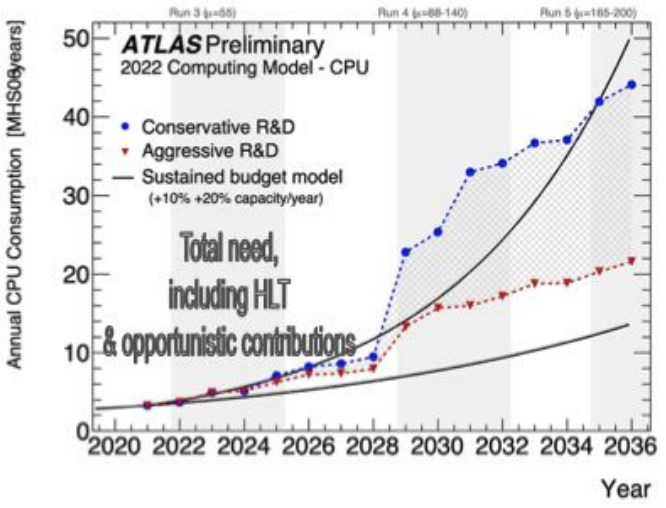
- **HL-LHC production is planned for June 2030**
- **The Accelerator is expected to deliver increasing luminosity: from 200/fb/yr to 330/fb/yr, with a pileup of 132 (Run4) and 170 (Run5)**
  - 2024: 120fb (and this is the target for 2025)
  - 1 inverse femtobarn corresponds to around 100 million million (potential) collisions
- **The goal is to reach 3/ab at the end of 2041 - 83% of the total projected LHC luminosity will be collected during the HL period 2030-2041**

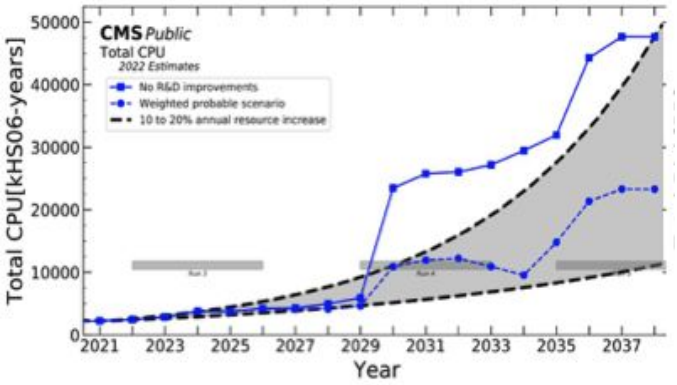


Run	Year	Reference PU	Reference (fb-1)	New Baseline PU	New Baseline (fb-1)
4	2030	100	32	100	32
	2031	132	206	132	206
	2032	132	237	132	237
	2033	132	234	132	234
5	2036	132	208	170	239
	2037	132	271	170	311
	2038	132	276	170	317
	2039	132	276	170	317
	2040	132	229	170	262
	2041	132	284	170	325
Sum Run 4 Run 5			2251		2480
Sum all runs (+520)			2771		3000

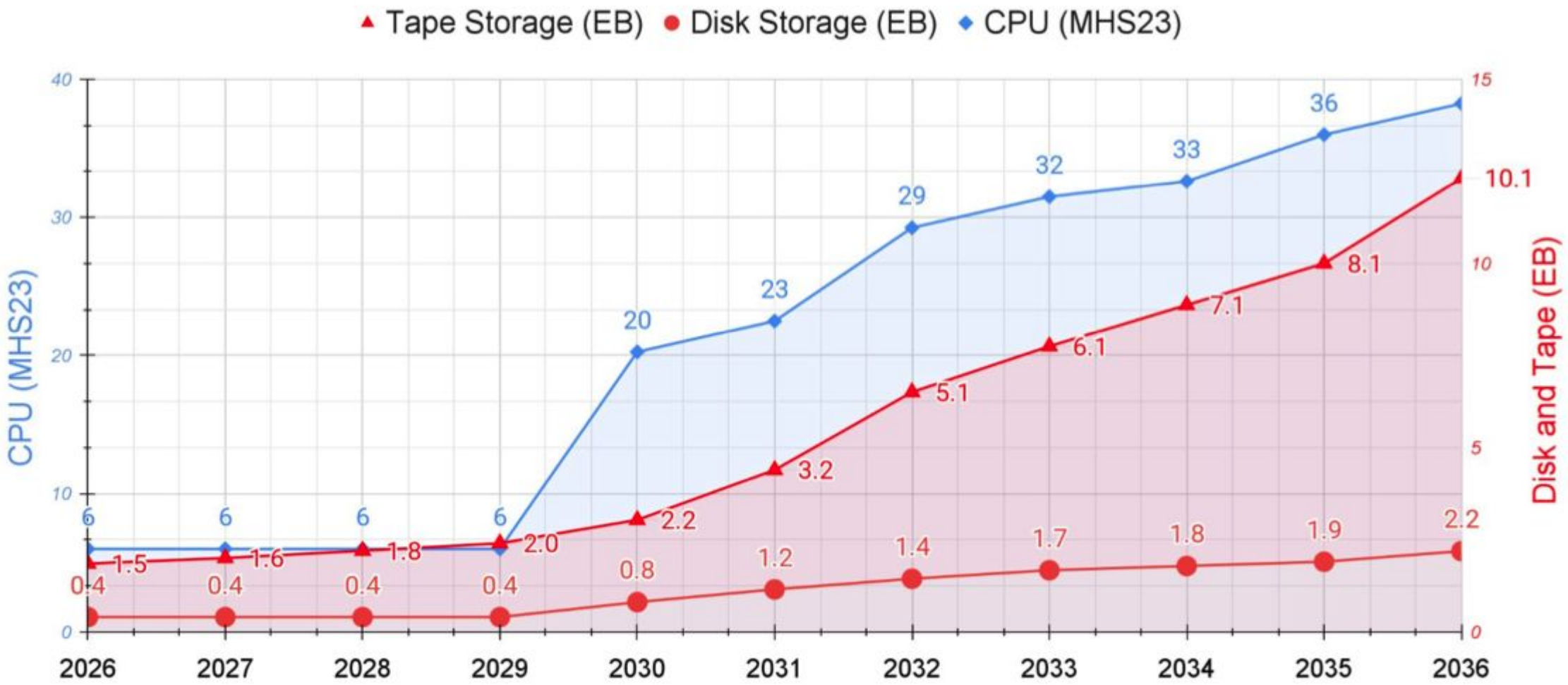
Tier-0 Compute Requirements (..disclaimers)

- In terms of WLCG resources, CERN Tier-0 provides ~25% of CPU capacity
- Our capacity numbers for HL-LHC are based on the latest official plots from ATLAS & CMS dating from 2022
- Experiments' predictions are by definition uncertain, and provide ranges based on R&D & technological evolution scenarios
- Changes in computing models / scenarios will have large impact on capacity and budgets
- Large questions around impact of GPUs, ML, HPC resources





Tier-0 HL-LHC estimated capacity scaling

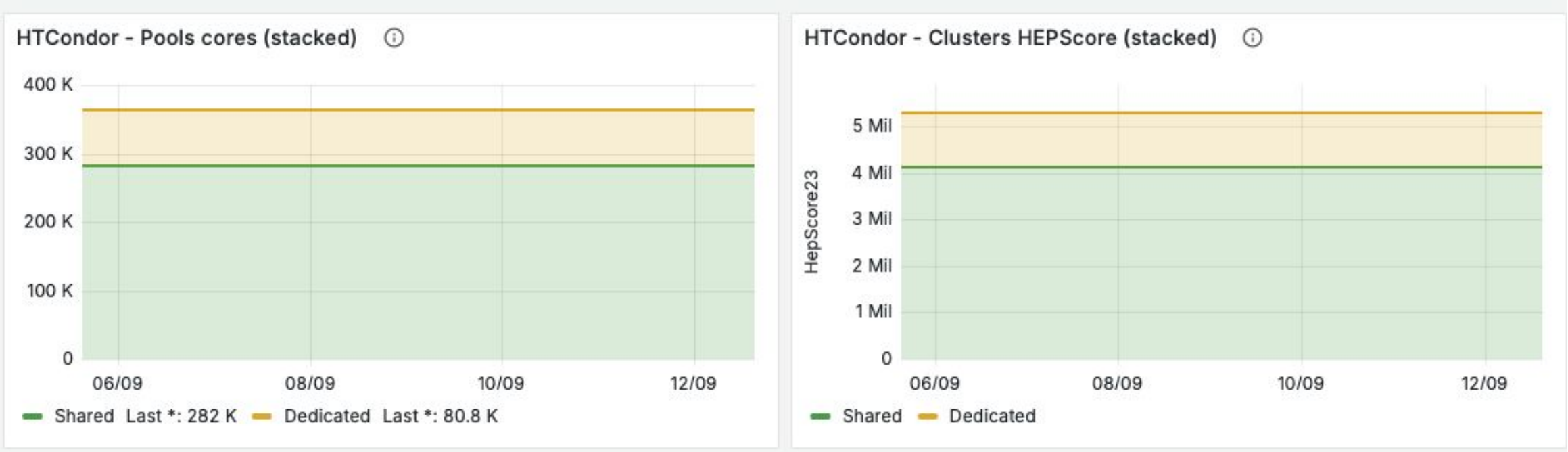


HTCondor @ CERN

Current deployment of HTCondor

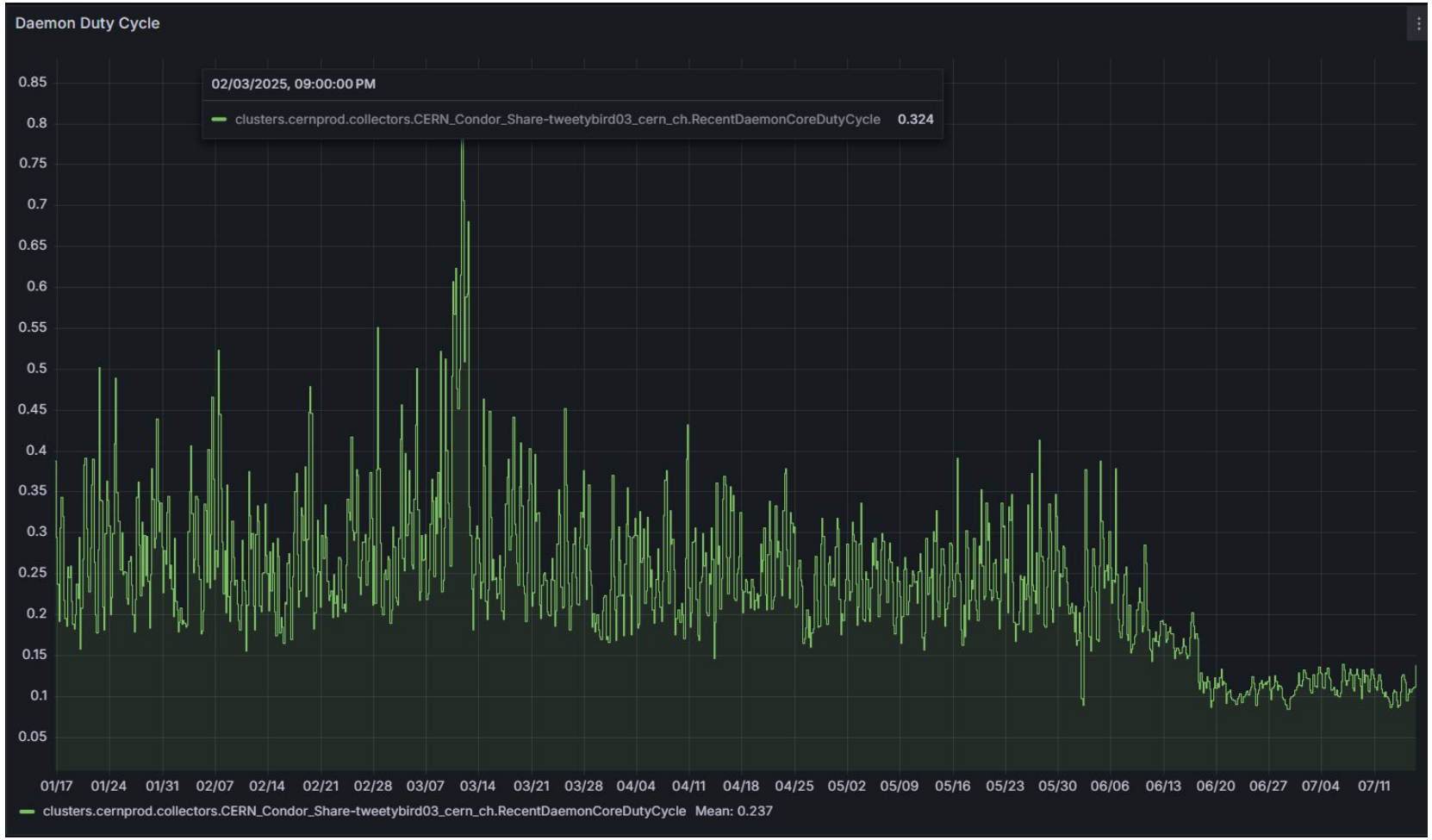
- **We have had two pools at CERN since very early in our deployment**
  - Prior to HTCondor we already had a pool in LSF dedicated to the ATLAS Tier-0
  - Now separate dedicated machines for ATLAS, CMS & AMS
    - Why have more slots for the negotiator in the main pool to deal with if there's no real negotiation?
    - Negotiator pressure in that pool is significantly lower than in the main pool
- **Worker nodes are all physical machines via OpenStack Ironic**
  - * there are some opportunistic VMs but not in any great scale
- **Infra / Control plane is all VMs**
  - Big VMs for the Central Managers (16 vCPU 120 GiB mem)
  - Schedds / APs scaled for ~10k running jobs (8vCPU, 15 GiB mem, CEPH w/ 300MB/s 1000 IOps)
    - `MAX_JOBS_RUNNING = ceiling($(DETECTED_MEMORY)*0.7*1024/1024)`
- **Job submission either:**
  - Local, via kerberos from plain APs with remote submissions (only: no access to schedds by users)
  - Grid, via htcondor-ce
  - Varies but something around 70:30 Grid:Local by cpu time, but 25:75 Grid:Local by # of jobs

Dedicated vs Shared capacity



- **About 3.5k physical machines in main shared pool, 1.5k in dedicated.**
- **Current average number of cores/machine in the main pool is ~78**
  - Older machines 48-64 core
  - Newer AMD machines 128 / 192 core
    - …though some small factor / low core count machines also possible in future

Recent HTCondor versions have helped

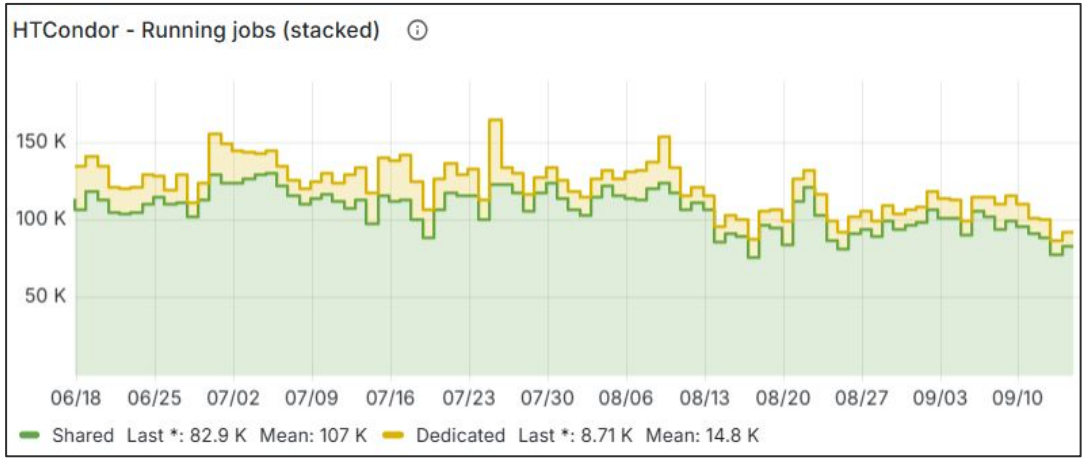


Spot the v24 upgrade….

What happens if we 4x compute ? ….how about 10x?

Will HTCondor cope? Should we partition? By how much? When?

Some preliminary (naive) numbers



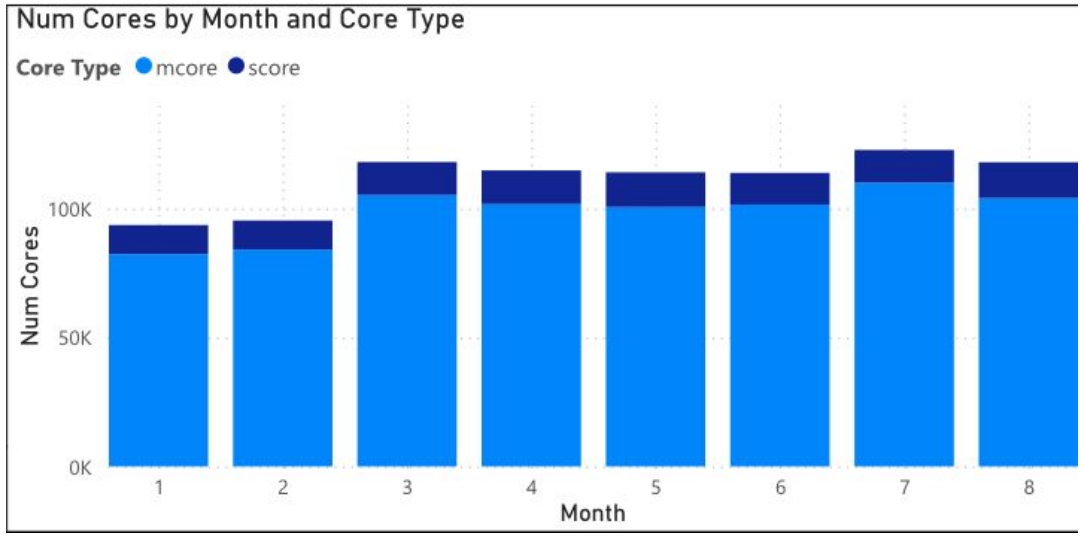
- **Currently:**
  - Total: ~5.5M HS23, 380K cores, 125K running jobs
  - Main: ~4.4M HS23, 300K cores, 110K running jobs
  - Main is 80% of total in terms of HS23
  - Number of cores / running jobs = ~2.7 cores/job

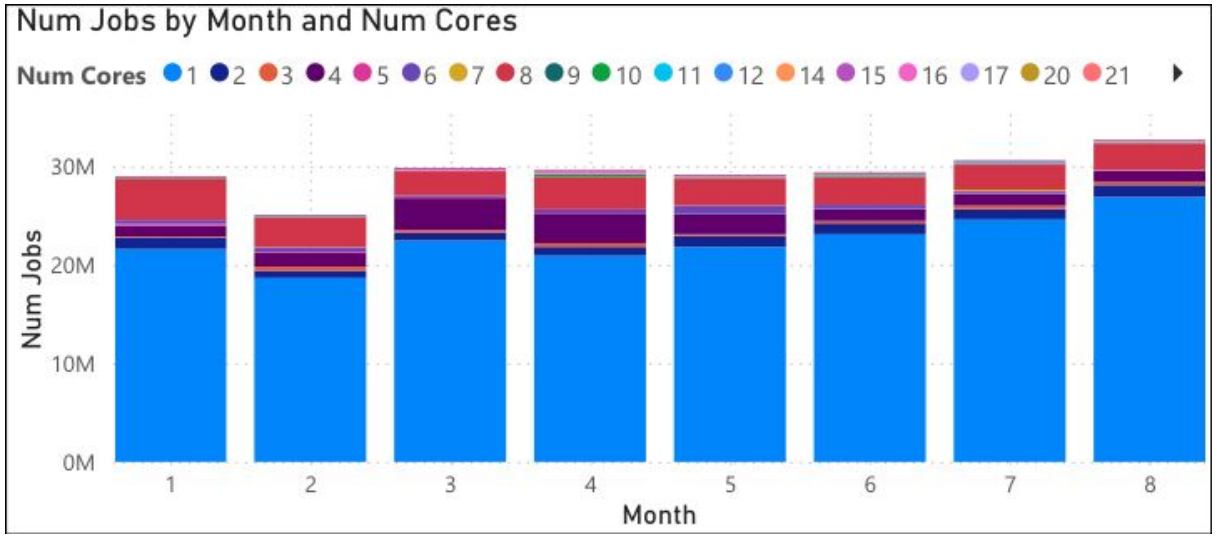
- **2031:**
  - Total: 23M HS23
  - Main: ~18.4M HS23 → 4.18x → 1.25M cores, 460K running jobs
  - But if cores are 10% more powerful every year:
    - 1.25M/(1.1**5) = 776K cores → 285K running jobs

- **2036:**
  - Total: 39M HS23
  - Main: ~31M HS23 → 7x → 2.66M cores, 780K running jobs
  - But if cores are 10% more powerful every year:
    - 1.25M/(1.1**5) + 270K/(1.1**6) + 270K(1.1**7) + ... = 1.4M cores → 514K jobs

Trends - growth will not necessarily be linear

- **Currently:**
  - Jobs are on average ~2.7 cores/job and their duration is ~3h
- **But…**
  - Pilot lengths are increasing for grid jobs
  - Both CMS & ALICE using more “whole node” pilots, further reducing number of jobs
  - For Run3 ALICE also became predominantly multicore, along with ATLAS & LHCb
  - Multicore standard might become 16 rather than 8
  - Larger EPs, with more cores, does not translate into more matchable slots





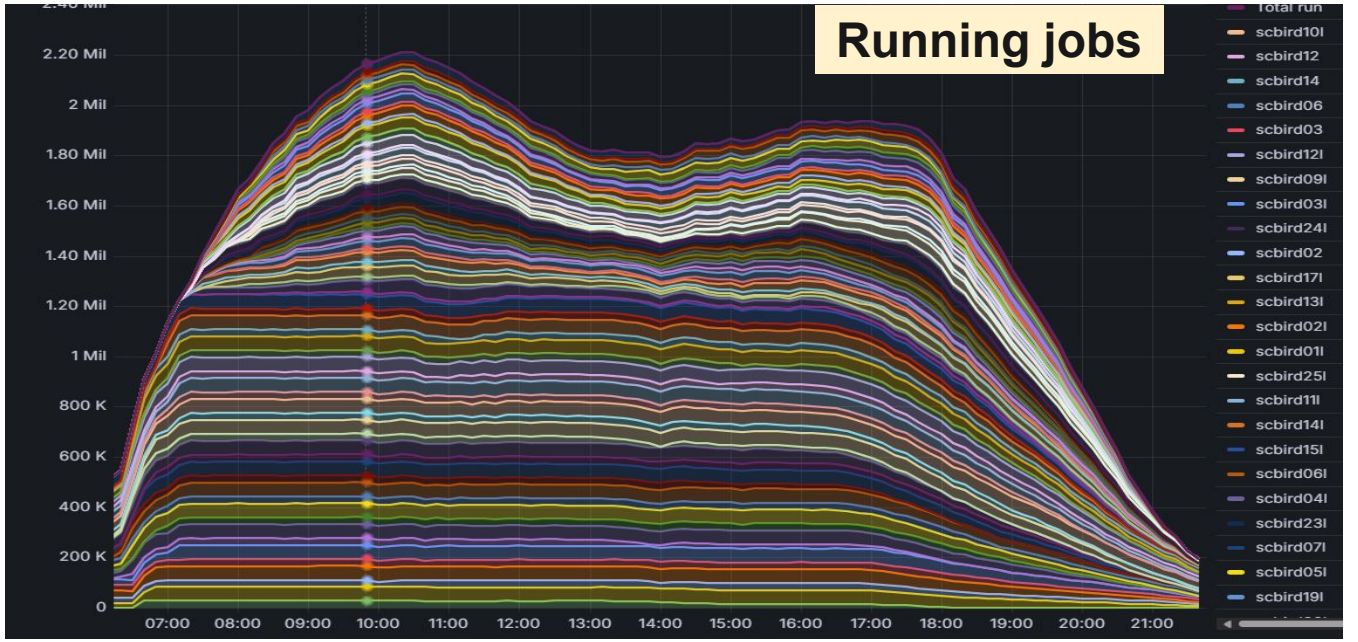
Scale tests

Scale tests

- **We deployed a test cluster to see the effects of running a much larger number of jobs**
  - Objective was to see the Central Manager response
  - So we mainly care about number of jobs and startds (not cores or filesystems)
- **EPs:**
  - ~230 physical machines: Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, 64 CPUs, 192 GB RAM
  - N startds per physical machine (up to 150)
    - Creating `/etc/condor<i>, /var/log/condor<i>…, systemd condor@.service`
  - 64 single slots per startd
- **CMs:**
  - Two large VMs (24 vCPUs, 180 GB RAM) in replication mode
- **APs:**
  - Up to 35 big VMs (8 vCPUs, 30 GB RAM) + 35 bigger VMs (16 vCPUs, 60 GB RAM)
- **Jobs**
  - All run as single core to increase pressure in collector/negotiator

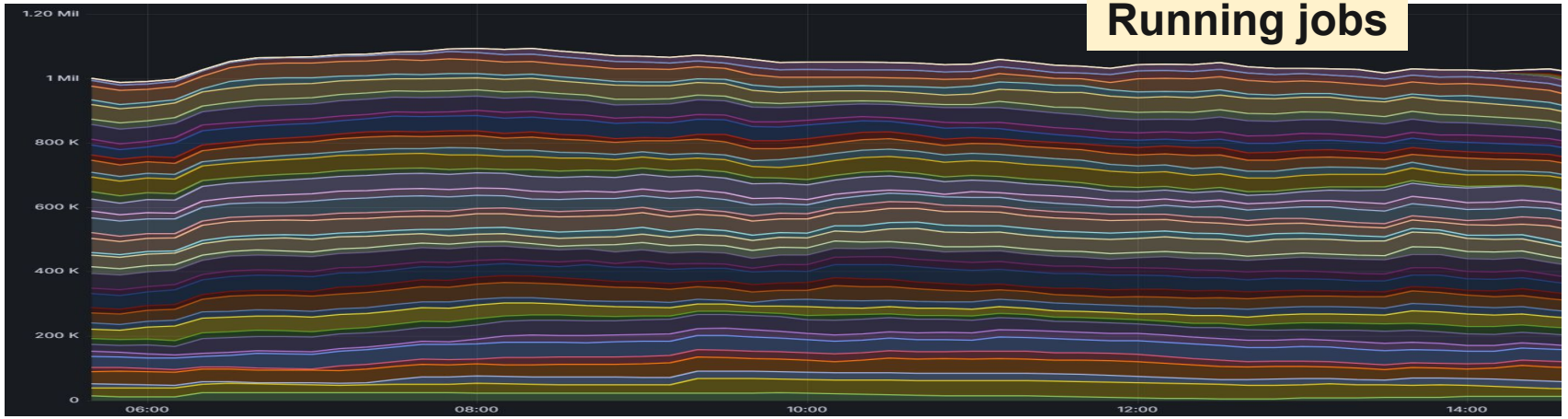
Can we even break it?

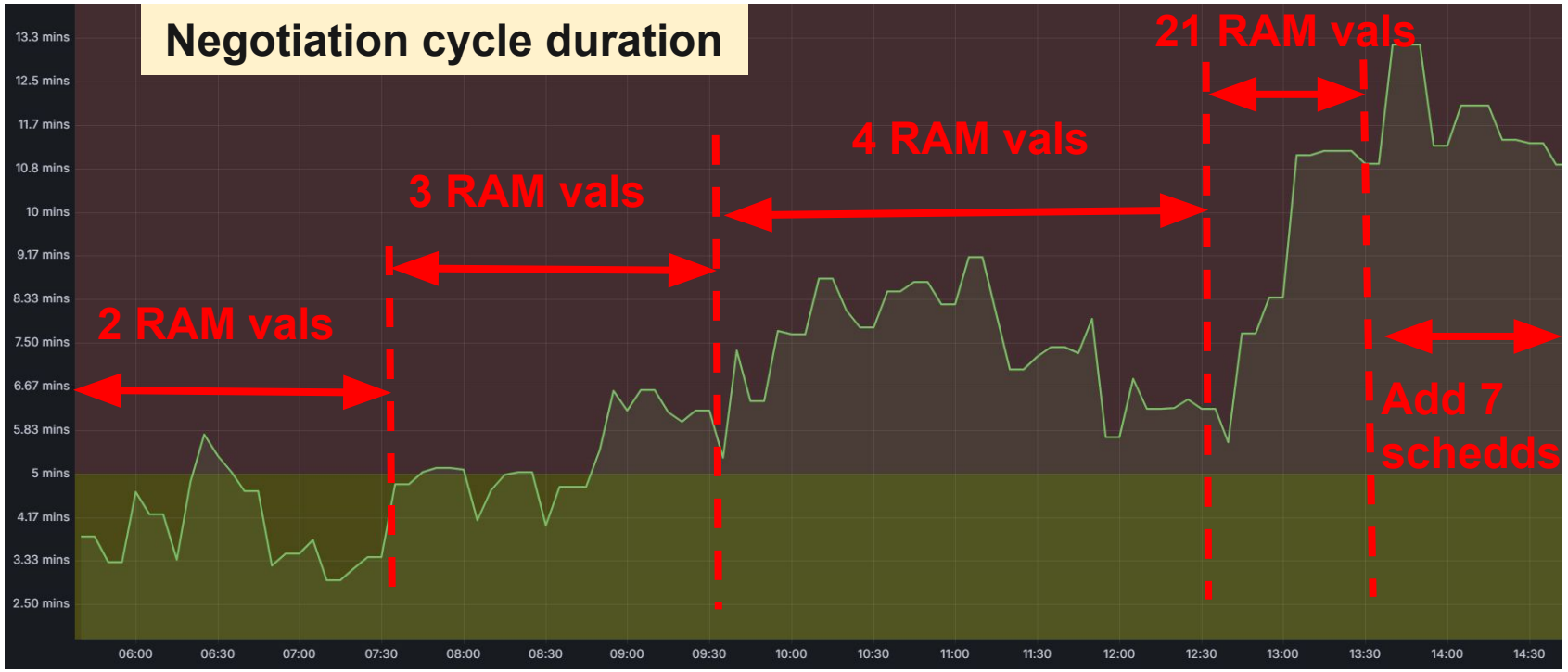
- **We were able to run more than 2 million jobs!**
  - That's ~20x current number
- **Collectors never reached saturation**
  - Not the limiting factor as found by previous CMS tests
- **Negotiation time was still reasonable (~5 minutes)**
- **However, this test used really simple jobs**
  - Homogeneous jobs (no special requirements)
  - Single user and acct group



Varying job requirements

- **Introducing a varying memory request in submitted jobs**
  - Job requests one of N possible RAM values
- **The duration of the negotiation cycle is increased**
- **Addings schedds also puts pressure on the negotiator**





Autoclusters and candidate (trimmed) slots

Autoclusters: job groups with similar requirements that are matched by the Negotiator.

- The more heterogeneous the jobs are, the more autoclusters there are, and the more work the Negotiator needs to do.

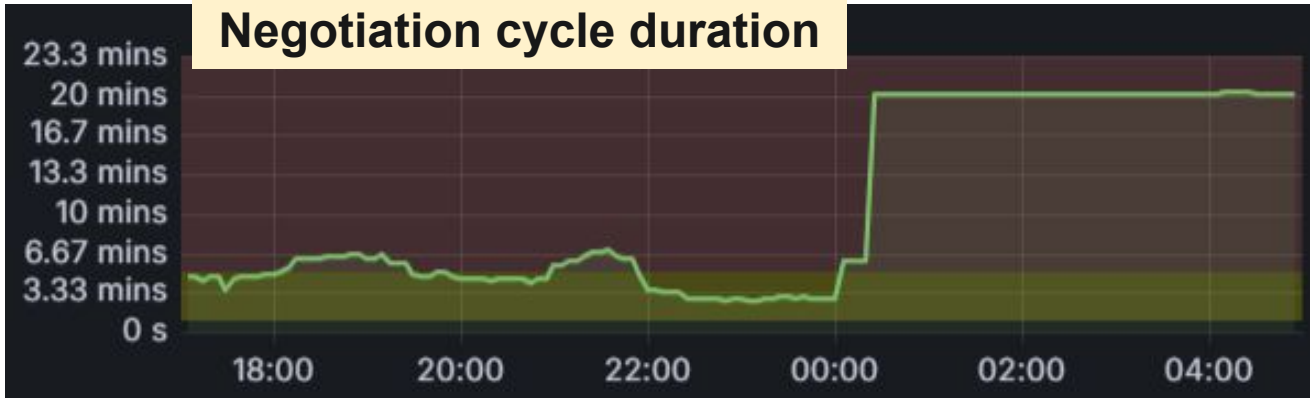
Candidate slots: the more slots to match, the more work for the Negotiator

- But we don't use pre-emption, thus only non-busy slots are matchable
- In our case, this is one slot per startd (the basic partitionable one)

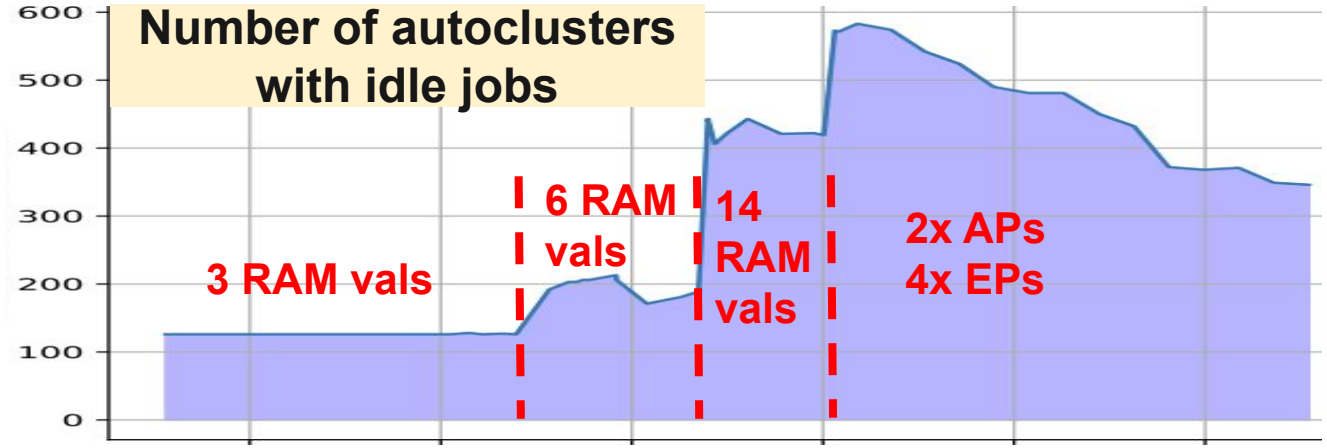
In principle the pressure on the Negotiator will be roughly proportional to:

```
Number-autoclusters-with-idle-jobs  x  number-of-EPs
```

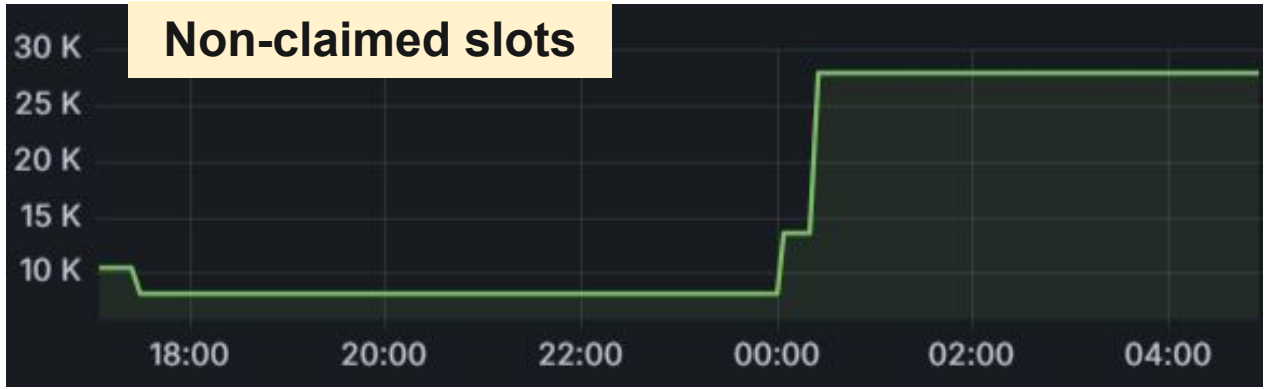
Negotiation cycle duration



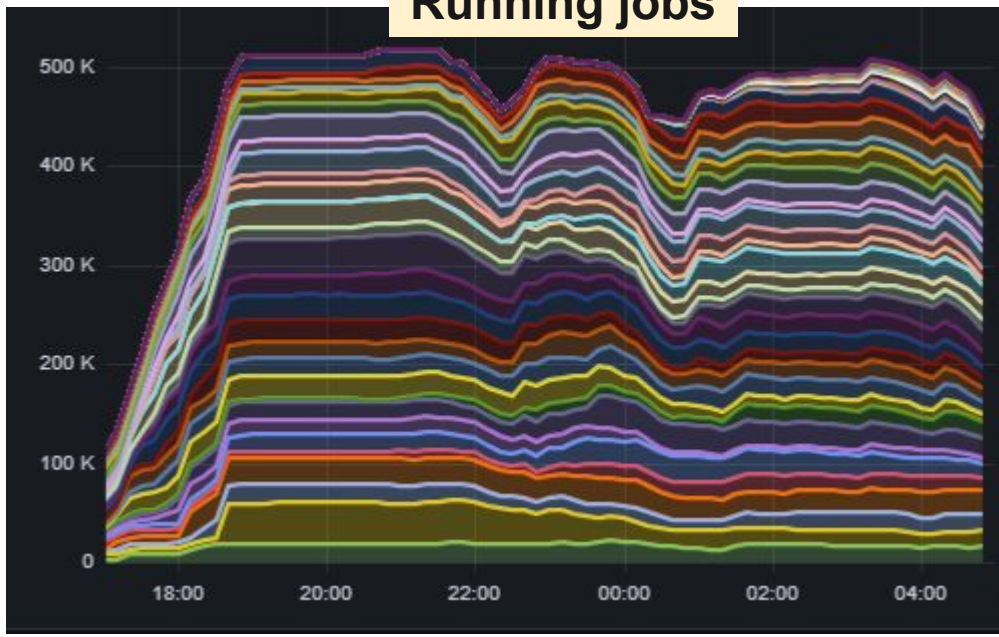
Number of autoclusters with idle jobs



Non-claimed slots



Running jobs



More on autoclusters

- **Autoclusters are created based on jobs having different values for attributes in:**

```
AutoClusterAttrs =
"condor,MachineLastMatchTime,MaxRuntime,PartitionableSlot,Rank,RemoteOwner,RequestC
pus,RequestDisk,RequestGPUs,RequestMemory,SlotID,TotalJobRuntime,ConcurrencyLimits,
FlockTo,Requirements,InStagedDrain,NODE_IS_HEALTHY,ShutdownTime,CvmfsCustomExpressi
on,DiskUsage,FileSystemDomain,NODE_MOUNTS_CVMFS,OpSysAndVer,PreAfsRequirements,PreC
vmfsRequirements,PreLocalRequirements,PreVerRequirements"
```

- **We think this is compiled by HTCondor and sysadmins can't directly modify this, however, may there be ways to avoid creating unnecessary autoclusters?**
  - For example, I saw different autoclusters based on default `RequestDisk` value. Could we perhaps quantize it to avoid too many different values?

- **In any case, we think it may be safe to assume that, for a fixed number of schedds, many more jobs will not necessarily mean many more autoclusters**
  - The variety of type of requests may stay more or less constant

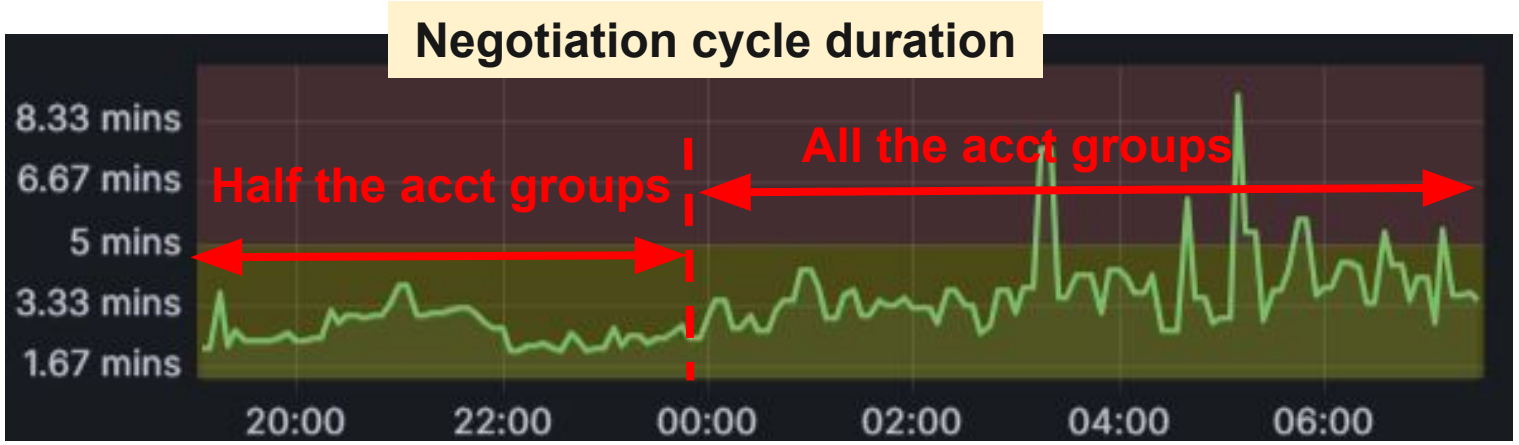
Accounting groups

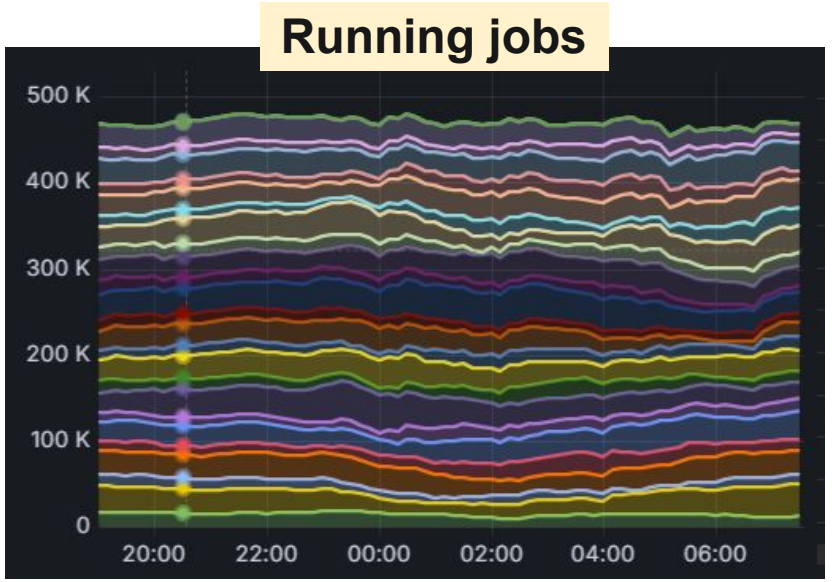
- **In principle, the number of accounting groups considered by the Negotiator should play a less important role than the number of autoclusters and candidate slots with respect to negotiation cycle time.**

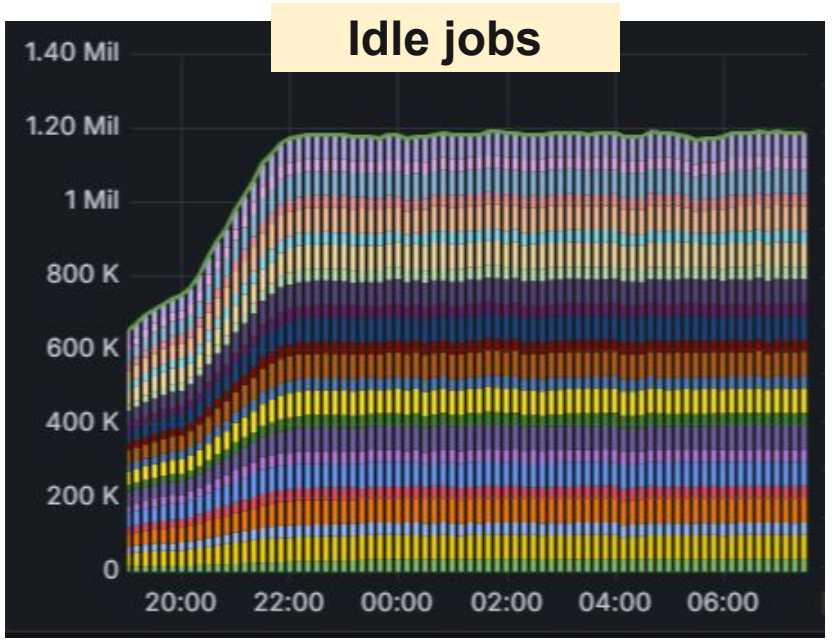
- **However, we did see an impact.**

- **We tested firstly with a single acct group, then setting the accounting group of the job to a random value out of either all or half the supported acct groups at CERN**
  - Matching time increased and fluctuated more
  - Admittedly, random acct groups are not what we see in reality, so we may need to investigate further

Accounting groups





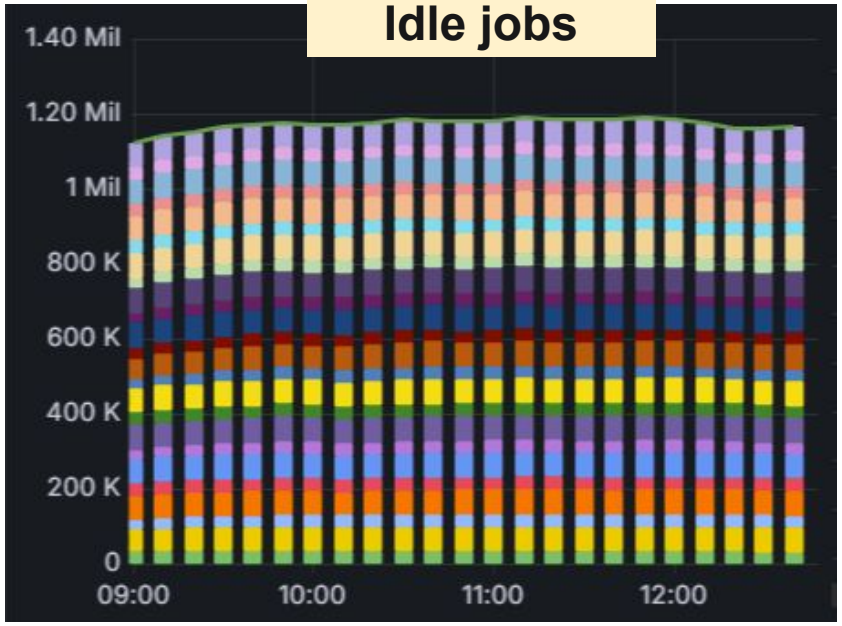


Note: this test was performed with a single memory request value for all jobs.

480K jobs… in realistic conditions?

Running jobs

Idle jobs

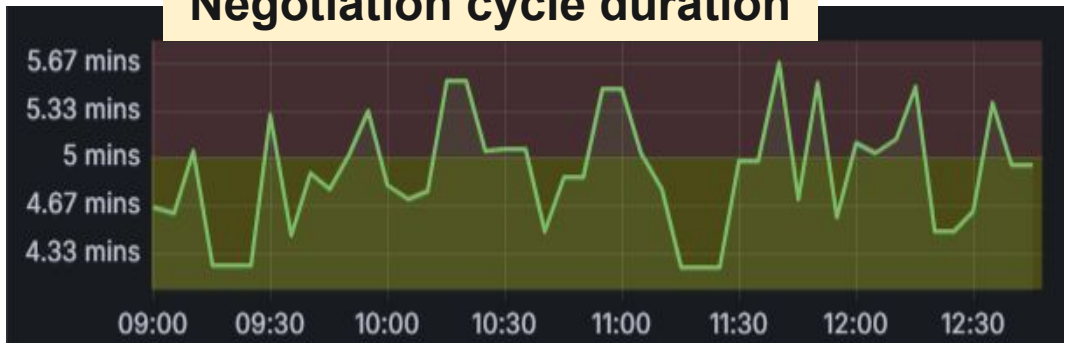


Numbers for this test:

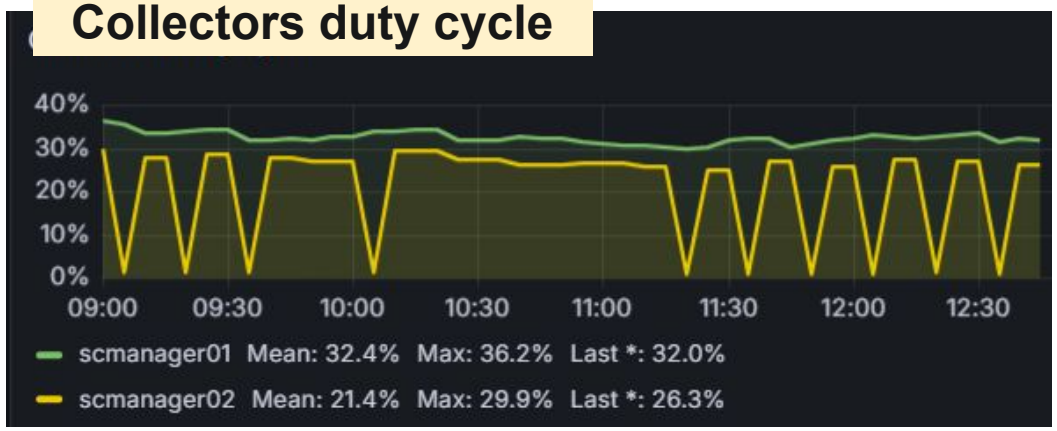
- 7500 startds
- 24 APs
- Randomized jobs
  - Half the acct groups
  - 4 random RAM values
- 200-300 autoclusters with idle jobs

slots*ac = 7500*250 = 1.9M

Negotiation cycle duration

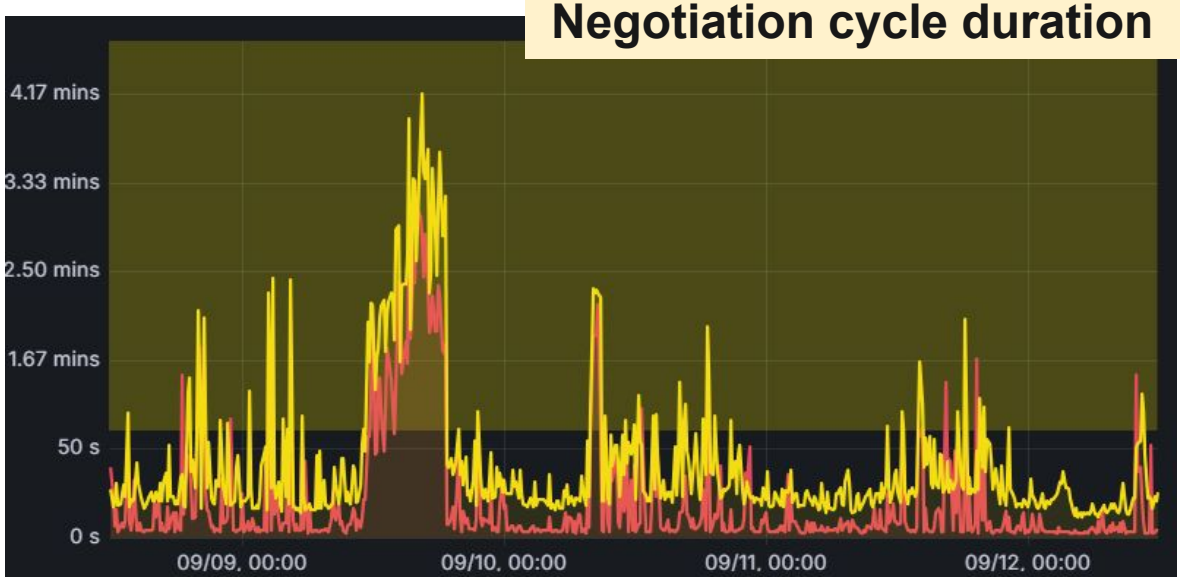


Collectors duty cycle

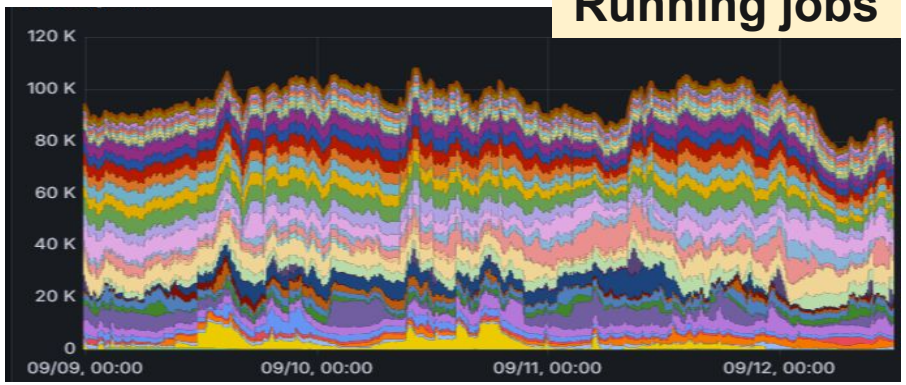


Comparing to the main production cluster

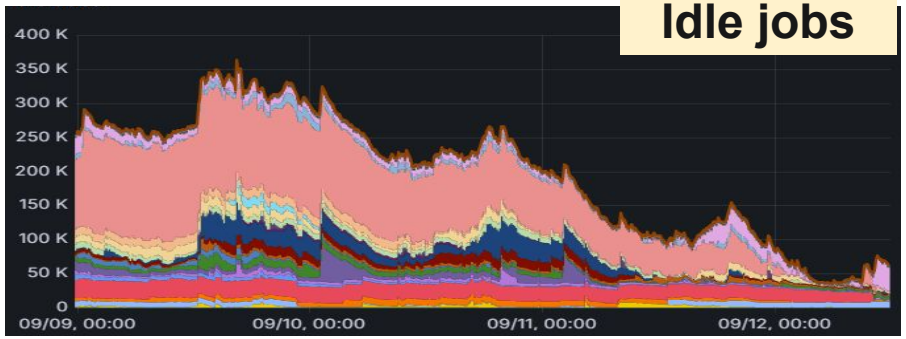
Negotiation cycle duration



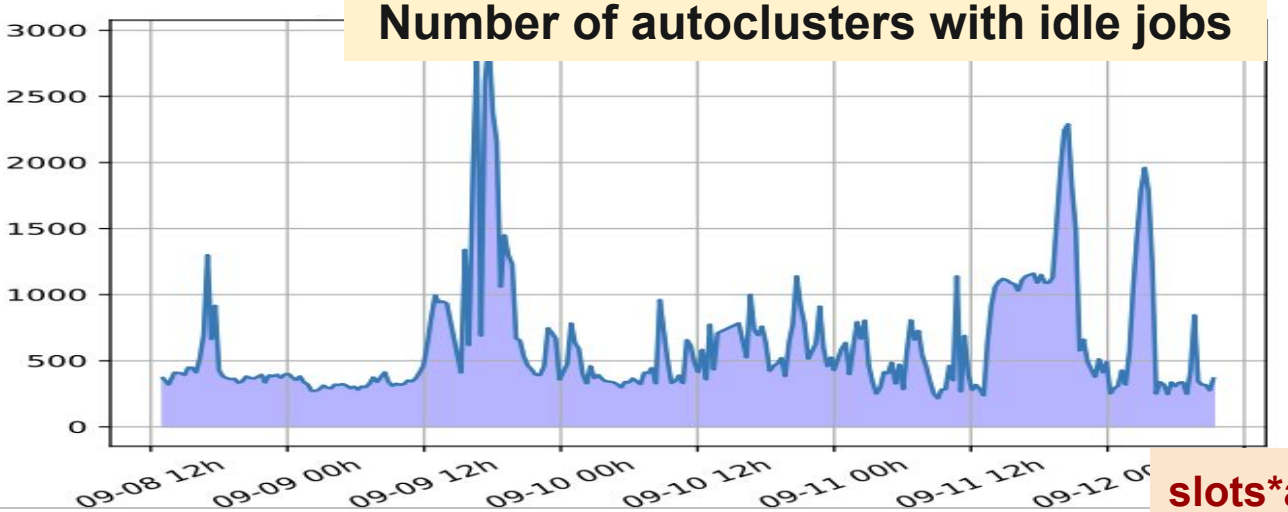
Running jobs



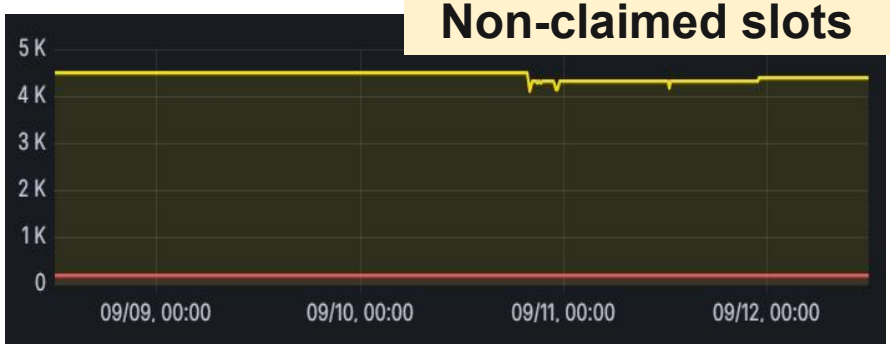
Idle jobs



Number of autoclusters with idle jobs



Non-claimed slots



slots*ac = 4500*500 = 2.25M

Conclusions

Naive numbers revisited

- If we consider that the number of autoclusters will remain constant and in 5 years the average number of cores per physical host increases to e.g. 192, we could increase the total number of cores by a factor of 192/78 = 2.462, without adding pressure to the negotiator.

- However, while the number of autoclusters and non-claimed slots seem to be the main factors for negotiation matching time, there may be others (total number of running and idle jobs, accounting groups).

Other ways to break HTCondor

Even if the Collector and Negotiator can cope with the load, the APs must too

Dangers for APs:

- Shared filesystems (AFS and EOS) are probably the biggest source of problems now
  - What will happen if the amount of handled data is 10x larger?
- Also, lots of spooled files may fill the AP spool area
- We also saw problems with the job queue logfile (when it grows beyond RAM in the host)
  - But that was some time ago. Is this still a concern?
- Number of jobs per schedd must be capped, or memory exhaustion kills Condor
  - We saw communication problems when schedd is too close to RAM/1MB running jobs

Fortunately:

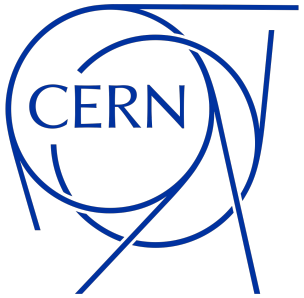
- We are testing ways to avoid file handling kill APs by transparently transferring to the EPs
  - Also we assume proportion of end-user jobs will be lower than experiment production
- APs can scale horizontally (and a bit vertically)
- A large number of APs will contribute to having more autoclusters, but current APs are far from their designed capacity in terms of running jobs (and larger APs can be deployed)

Conclusions and future work

- **The negotiator (job matching) rather than the Collector seems to be the element suffering the most when we scale**
  - The main factors to consider are number of slots to match (one per host) and the number of autoclusters
- **The APs are not saturated and could also be scaled horizontally**
  - With the caveat of increasing the number of autoclusters
- **We need to understand better what role accounting groups have in Negotiator's load**
- **We should investigate possible ways to avoid explosion in autoclusters**
- **Overall, we have room for growth. While we need to understand autoclusters and accounting groups better, scaling to almost 500K jobs seems possible**
  - More so if average number of cores / physical hosts increases
  - We could also run two negotiators on the same cluster (split slots)
  - Other trends may help: higher VO vs users job ratio, more powerful cores, longer jobs, higher core per job ratio

Thank you!

Questions?

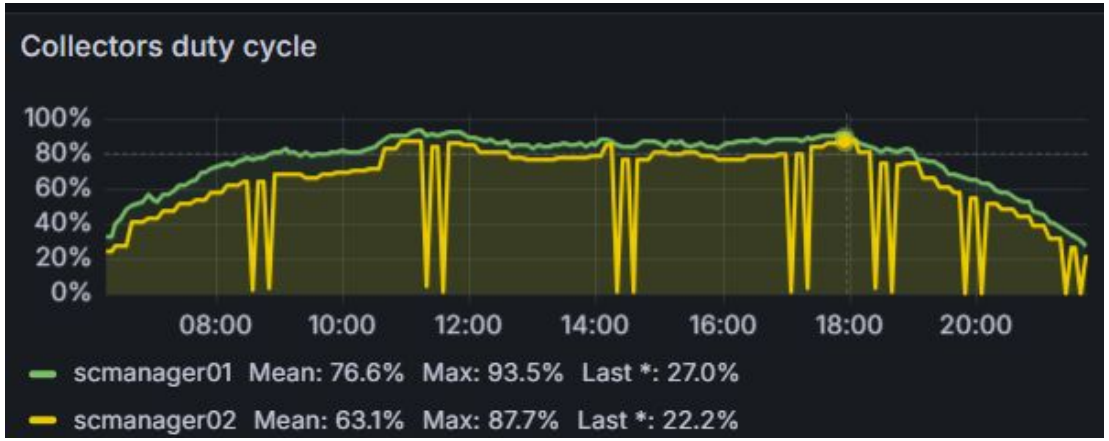


Back-up slides

Running 2M jobs details

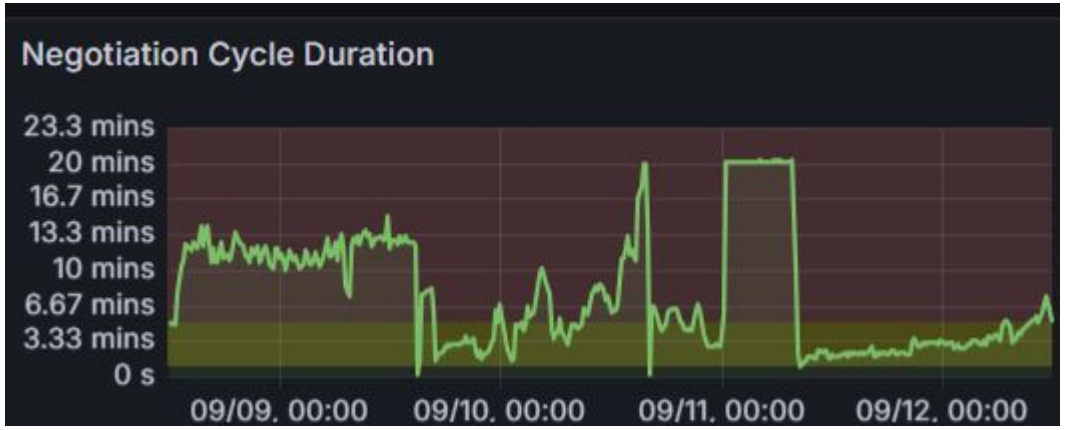
- Submission rate was 16.8K jobs per minute, capped at 3.15M idle jobs
- Number of autoclusters with idle jobs was ~80
- Trimmed slots (candidates) was 34.8K

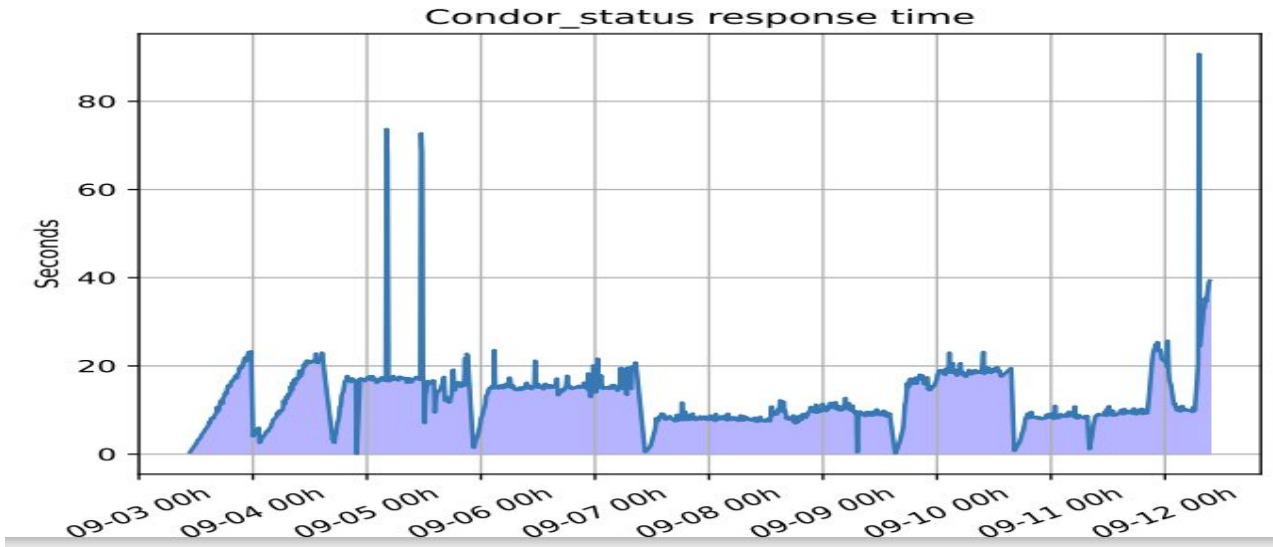
⇒ matching factor = 34.8K * 80 = 2.78 M



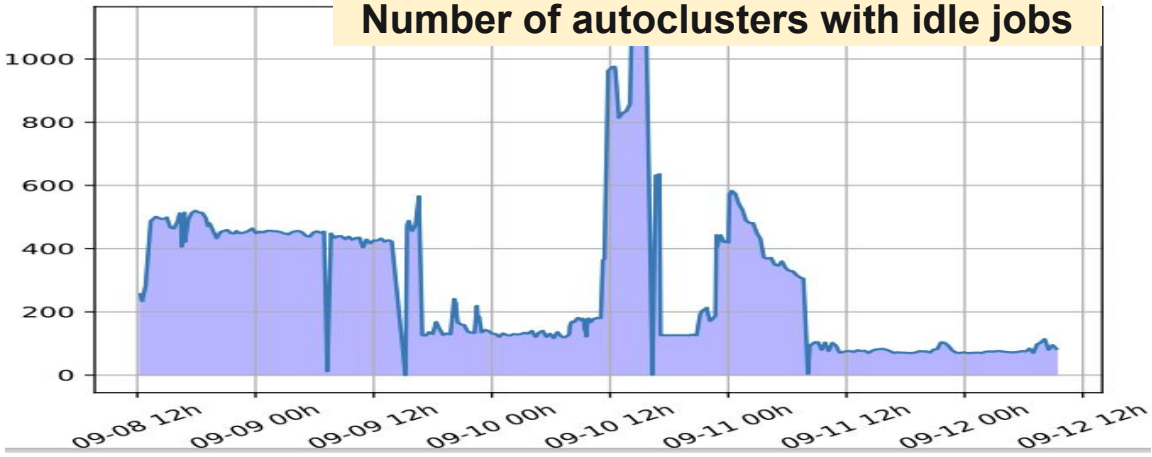
Scale test autoclusters all

Running jobs





Number of autoclusters with idle jobs



Negotiation time vs running jobs

- Negotiation cycle time increases with running (and idle) jobs, even for constant number of autoclusters and non-claimed slots