Planning for Future Scales and Complexity of HTCondor Pools in the CMS Experiment

James Letts, Antonio Pérez-Calero, and Saqib Haleem
on behalf of the Submission Infrastructure Group
of the CMS Experiment at CERN

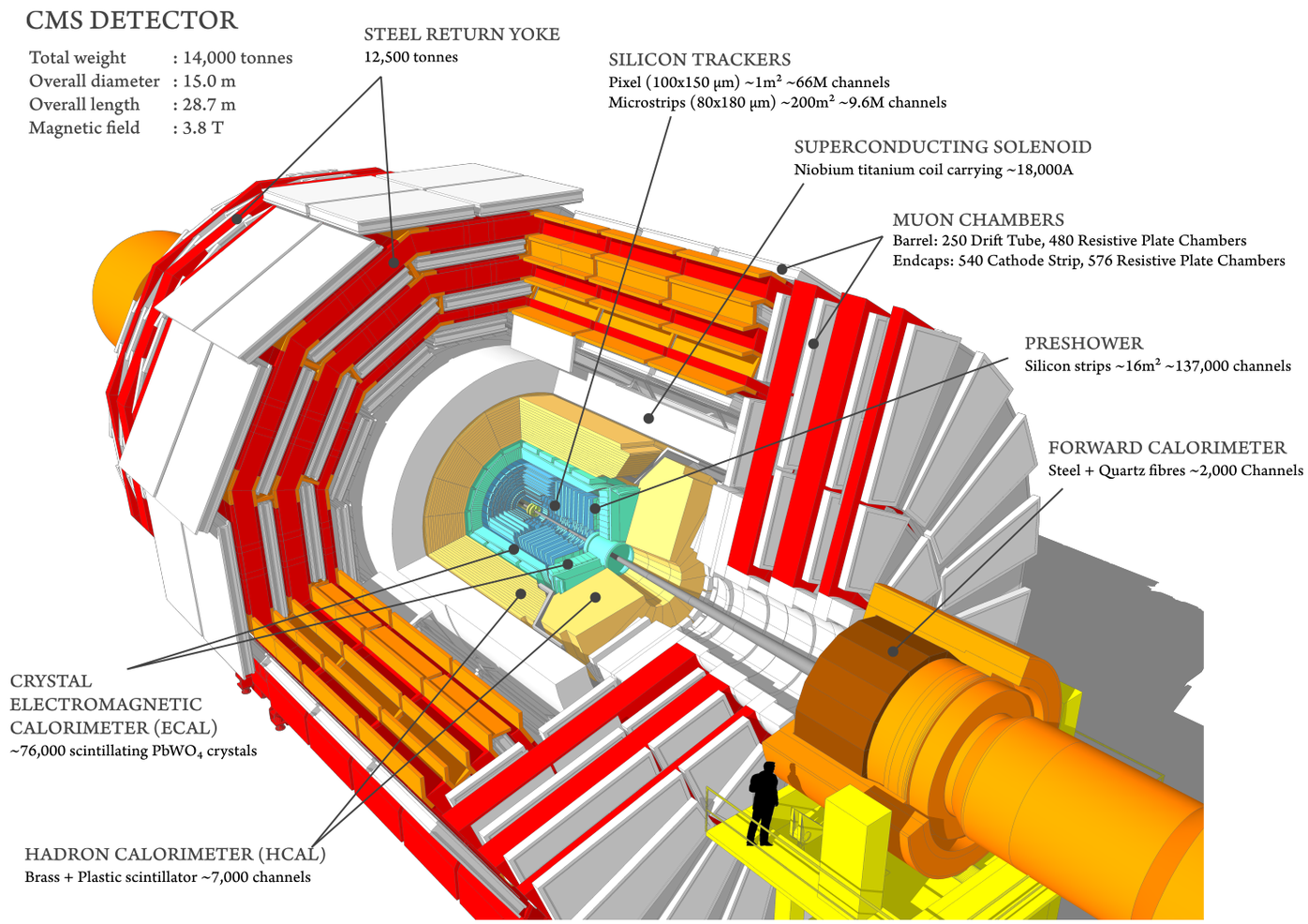
European HTCondor Workshop
Ispra, Italy - September 25, 2019

Abstract

The resource needs of high energy physics experiments such as CMS at the LHC are expected to continue to grow significantly over the next decade, and will be more and more satisfied by computing capacity with non-standard characteristics. This presents challenges not only of scale but of complexity in resource provisioning and allocation. In this contribution, we will present results of recent HTCondor scale tests we have conducted using the CMS Global Pool Integration Test Bed (ITB) employing the multi-threaded Negotiator, where we have pushed the size of the pool to the maximum limits with currently-available hardware and explored effective performance limitations of the submit nodes in our infrastructure with realistic payloads. We will also discuss recent integration of resource-specific job matching conditions to satisfy HPC and Cloud use cases, where resources may not be suitable for running all kinds of workflows. Finally, we will review some specific use cases that we have difficulty solving with the current implementation of HTCondor.

CMS Experiment at the LHC at CERN

- CMS is a general-purpose discovery experiment running at the LHC at CERN.
- Acts like a giant, high-speed camera, taking 3D “photographs” of particle collisions from all directions up to 40 million times each second.
- Output data expected to grow by ~50% in Run 3, and over 20x current levels by Run 4 (HL-LHC).
- Data is written to disk and tape at CERN, from where it enters the computing infrastructure of the experiment.



The CMS Submission Infrastructure Group

- The Submission Infrastructure (SI) Group is a coordination area within the Offline Software and Computing Project of the CMS experiment at CERN.

- We run the infrastructure in which all subsequent processing, reconstruction, simulation, and analysis of physics data takes place after it leaves the experiment.

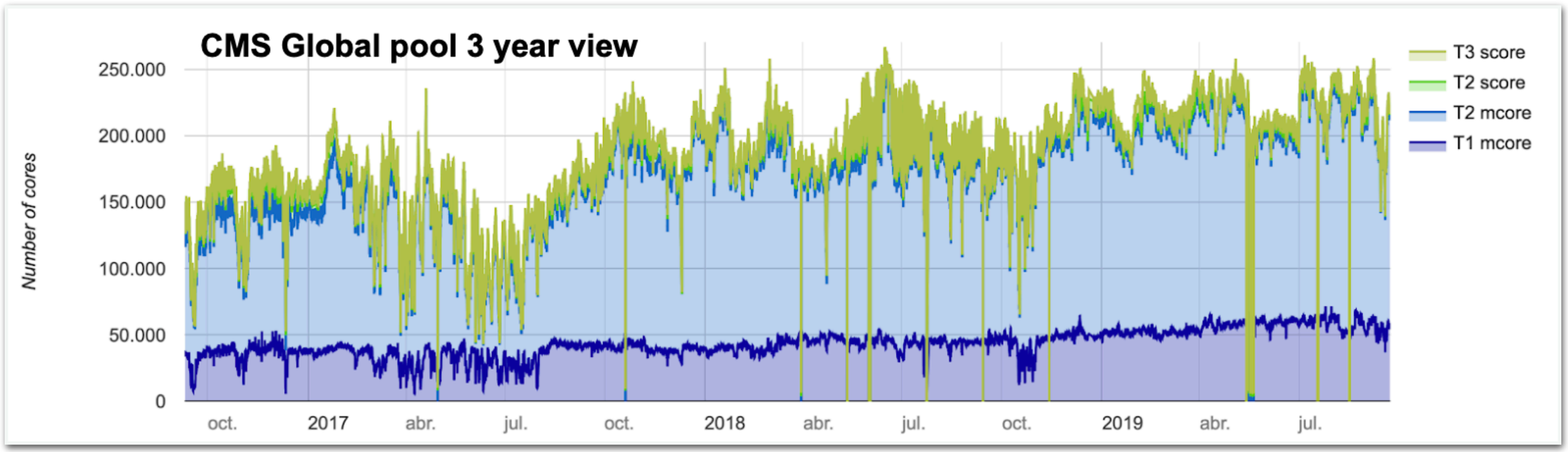
- Our dual charge is:
  - To organize GlideinWMS and HTCondor pool **operations** in CMS
  - To communicate CMS **priorities** to the development teams

- SI activities broadly fall into three areas:
  - Overcoming current operational limitations or problems
  - Integration of new, diverse resource types and submission methods
  - Preparing for future scales and feature requirements

- New main operator at CERN since July: Saqib Haleem

Increasing Scales

- Size of the main HTCondor pool in CMS (the “Global Pool”) has doubled in size during the past 3 years, driven largely by new resource deployments during Run 2 of the LHC. Currently running regularly at 250K CPU cores.
- Including the CERN pool, peaks over 300K CPU cores.
- Progressively adding more opportunistic (beyond pledge) and HPC resources, e.g. HLT farm (when not being used for data taking).



Evolution of Scale

- LHC currently in a two-year shutdown
- Run 3 (2021) processing scale +50%
- Run 4 (2027) processing scale over 20x current levels.
- Current sites will never be allowed to grow 20x in capacity: Increased use of Cloud and HPC resources, a transition that has already started.
- Increasing thread count of jobs (multi-core) and software improvements may make these increases significantly smaller, especially from the submission infrastructure point of view (i.e. number of jobs, or job sets).

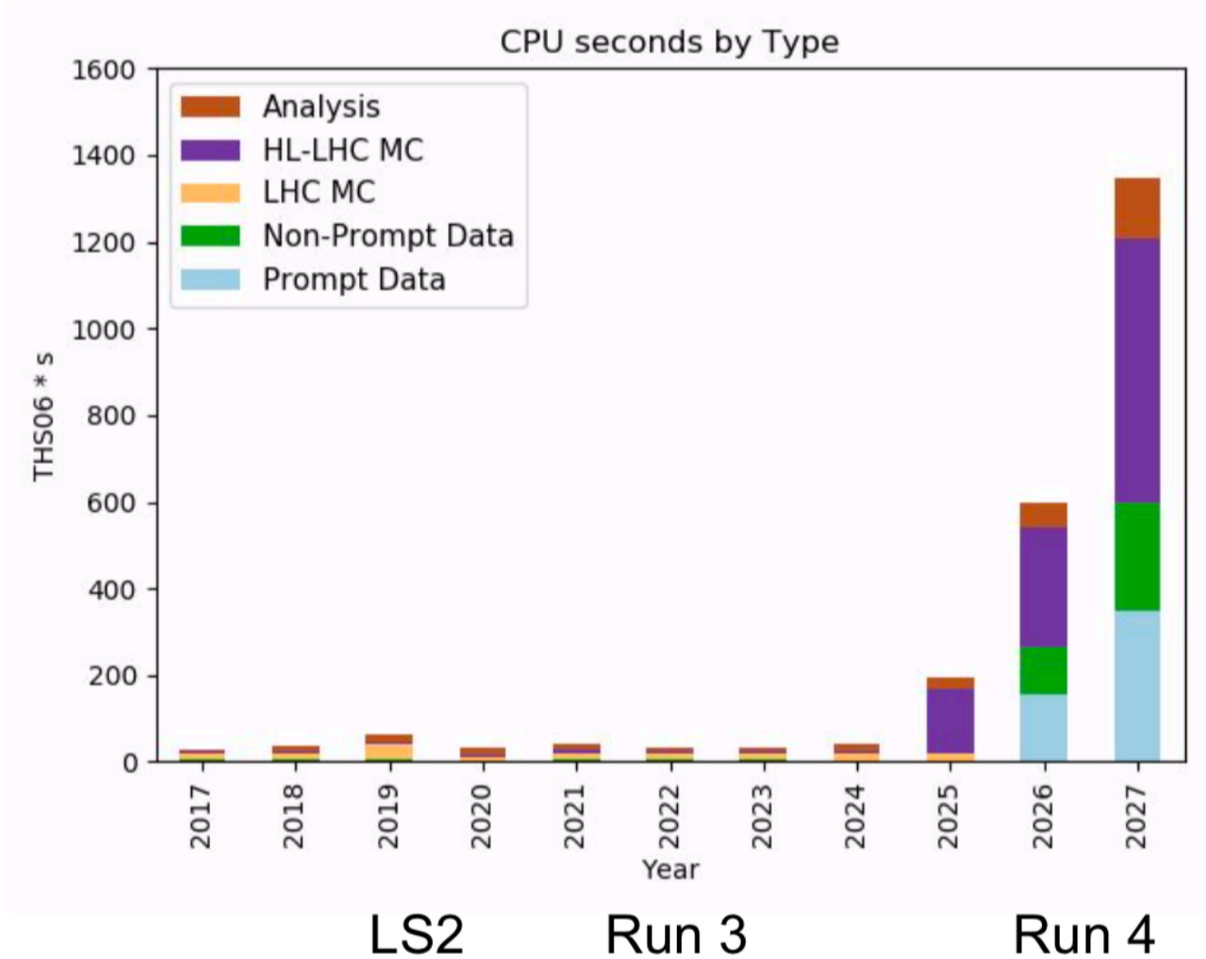
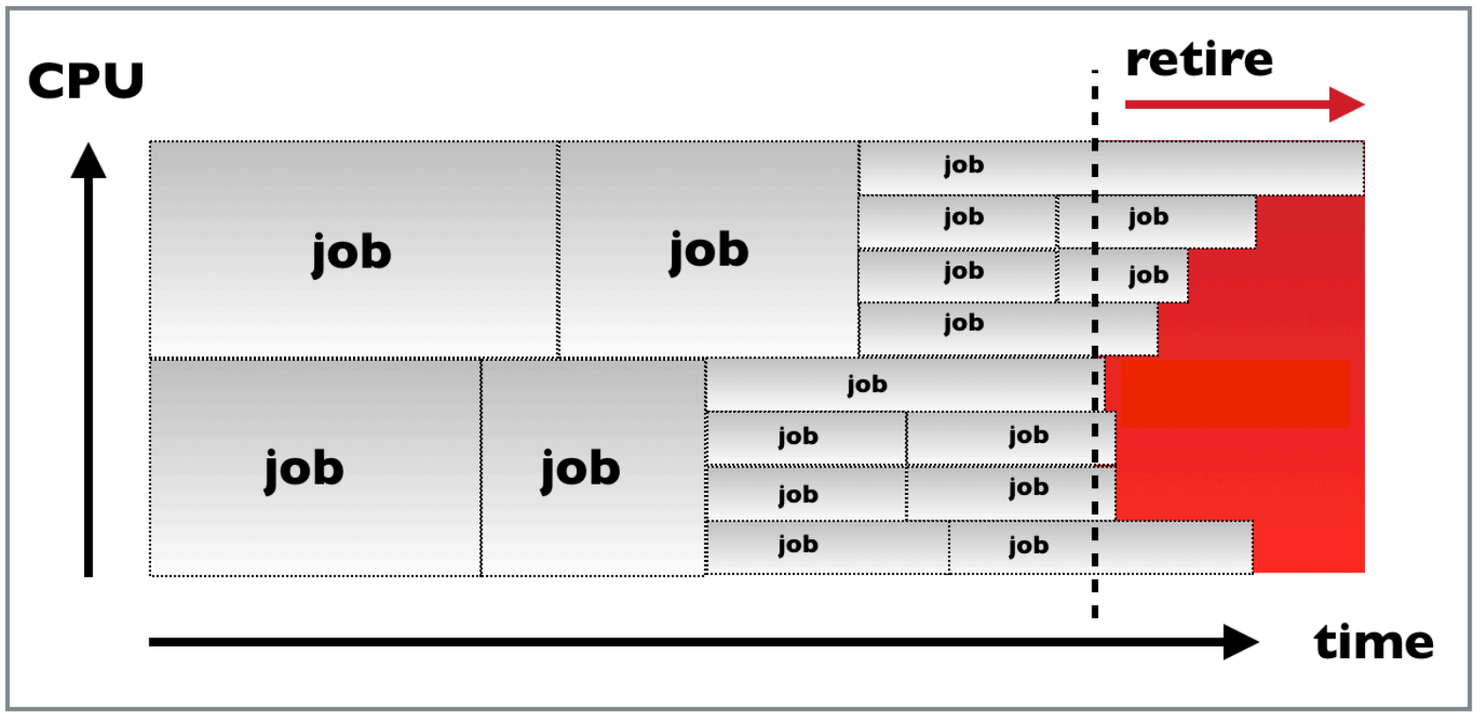


Image source: CMS Offline and Computing Results

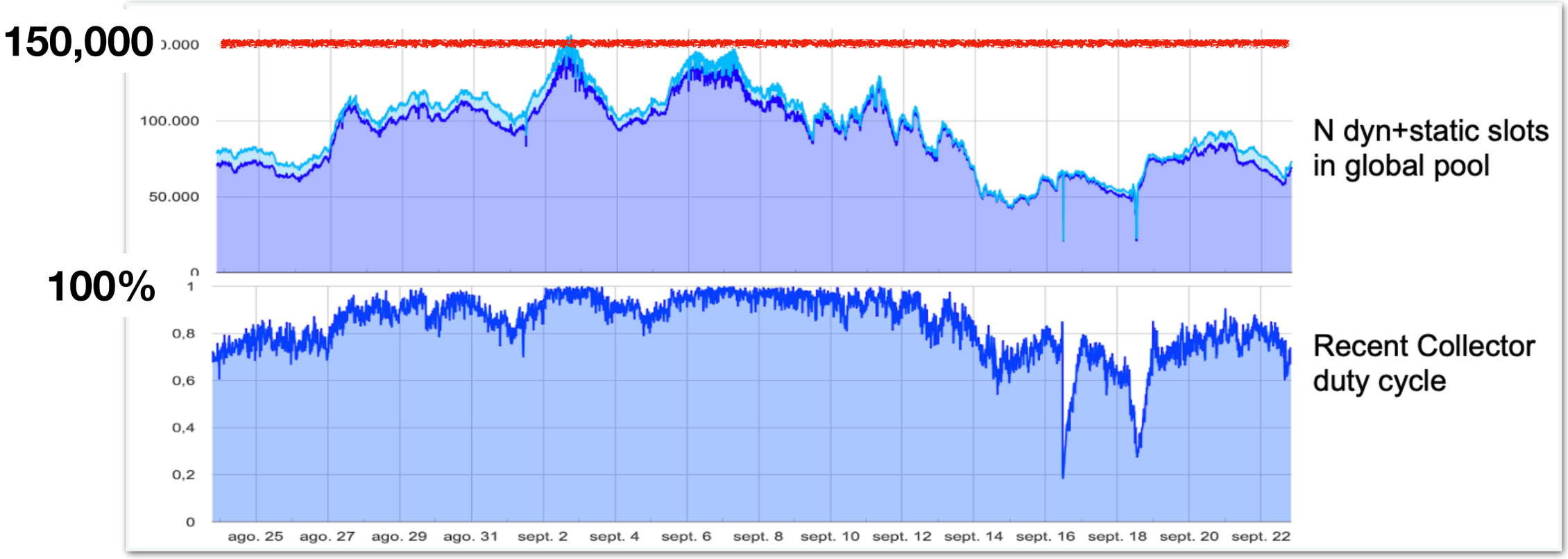
Multi-core

- CMS moved to multi-core workflows several years ago. Our pilot model of resource provisioning creates **partitionable slots** in our HTCondor pools.
- Improved scalability over single-core workflows: fewer jobs and fewer **dynamic slots** on the same number of physical cores.
- Typically CMS runs on up to 300,000 CPU cores globally but only 150,000 dynamic slots.
- P-slots can become fragmented over time: pilots renewed every ~48h.



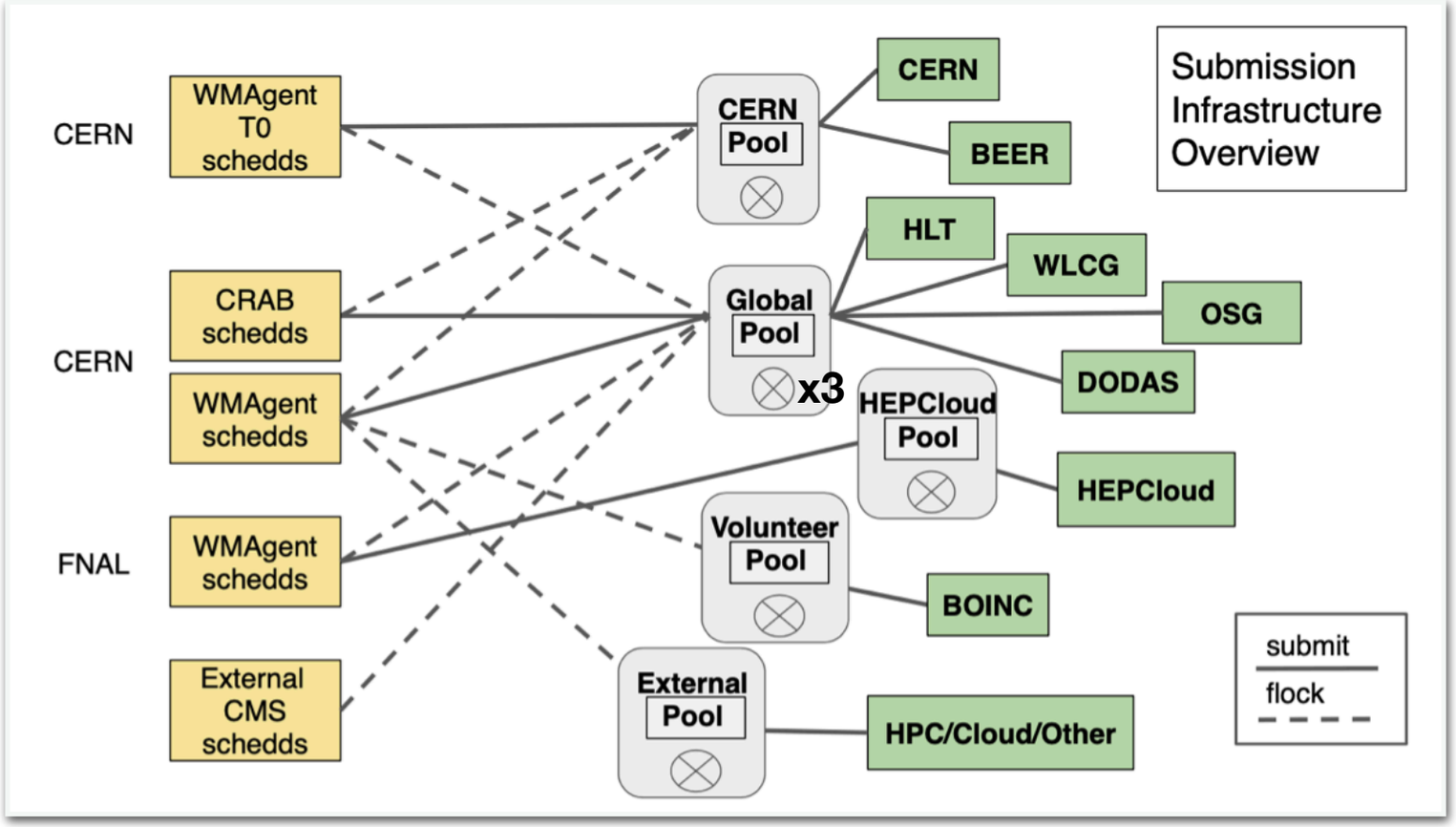
Multi-core

- In our Global Pool, we observe a scalability limit around 150,000 dynamic slots (upper plot).
- Main scaling driver for Collector (duty cycle, lower plot)
- We have not yet deployed the multi-threaded Negotiator in production - but we have in the Global Pool ITB for the scale tests.



Current CMS Submission Infrastructure

- All CMS workflows (Tier-0, Production & Reprocessing (WMAgent), and physics analysis (CRAB) queued on HTCondor schedd's at CERN or Fermilab.
- Work can "flock" to different HTCondor pools (central managers): CERN, Global Pool (all Grid resources not at CERN), HEPCloud (Fermilab), and the new "Volunteer" Pool (CMS@Home).
- CERN pool is separate to minimize risks to stability and scalability, and dedicated primarily to data taking.
- Different resource provisioning mechanisms, mainly GlideinWMS (pilot model), but increasingly using non-pilot instantiated startd's.



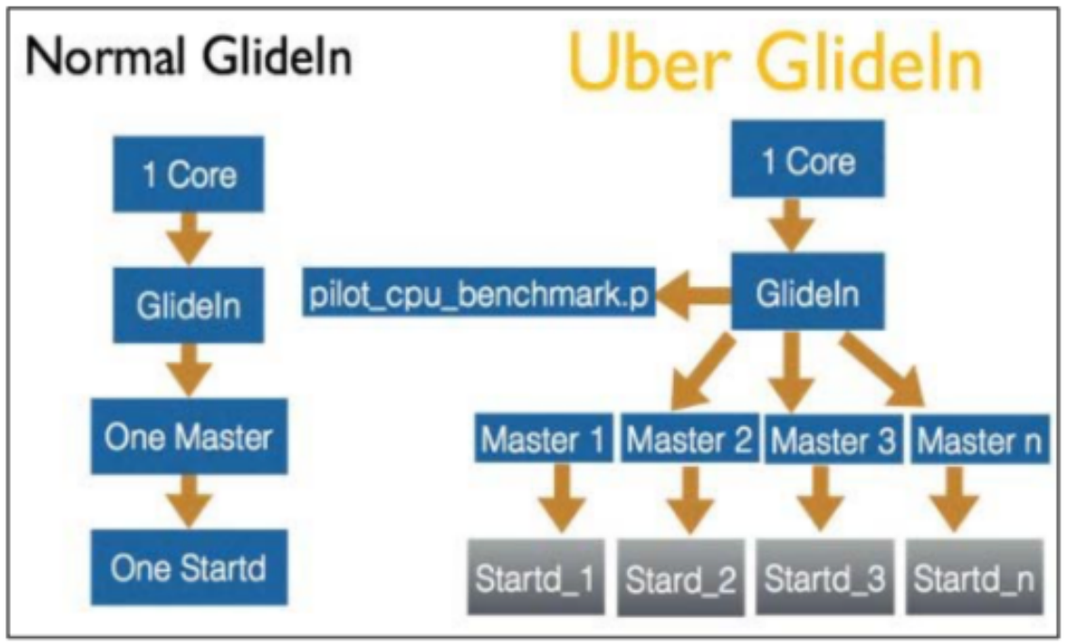
Future Challenges

- Therefore, we have challenges of scale and complexity:
  - Expect a doubling of scale in the next few years (Run 3)
  - Order of magnitude (or more) increase by 2027?
  - HPC resources often sit behind their own HTCondor pools: What is the limit to how many pools can a schedd flock work?

- CMS has run Scale Tests every year or two to study these kinds of questions. Goals for the 2019 Scale Tests:
  - Test scalability of the **multi-threaded Negotiator**
  - Push number of **dynamic slots to maximum** possible given hardware limitations of the central manager - new machine at CERN with 256GB of RAM.
  - Evaluate with realistic sandboxes, core counts, etc.
  - Maximum **job start rates** on schedd's
  - Scalability of **federated pools**
  - Generally assess improvements in HTCondor and GlideinWMS software since the 2018 scale tests

Scale Test Set-up

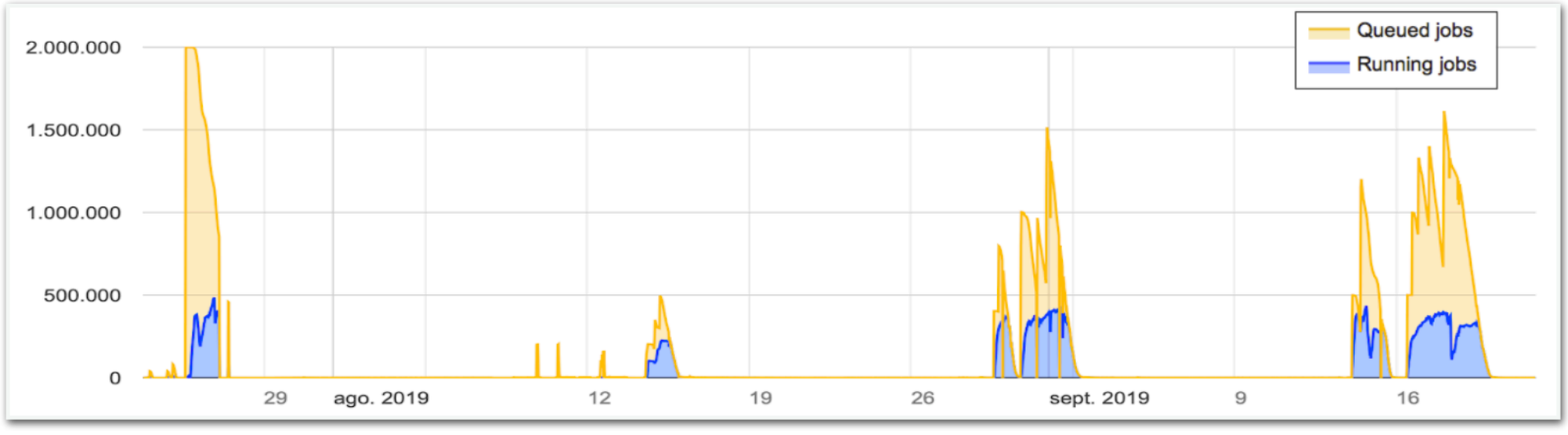
- Use fully-HA HTCondor pool ITB, lately with 256GB RAM primary central manager machine, ~2x larger than the production machine in the Global Pool but with the same configuration.
- Run 32 startd's on each physical CPU on the Grid: simulate 500,000 startd HTCondor pool with only ~16K physical CPU's.
- Integrate the multi-threaded Negotiator.
- Test jobs with realistic distributions of RequestCpus, RequestMemory, job length, input sandboxes, etc.



Scale Test Rounds

Four scale test rounds so far:

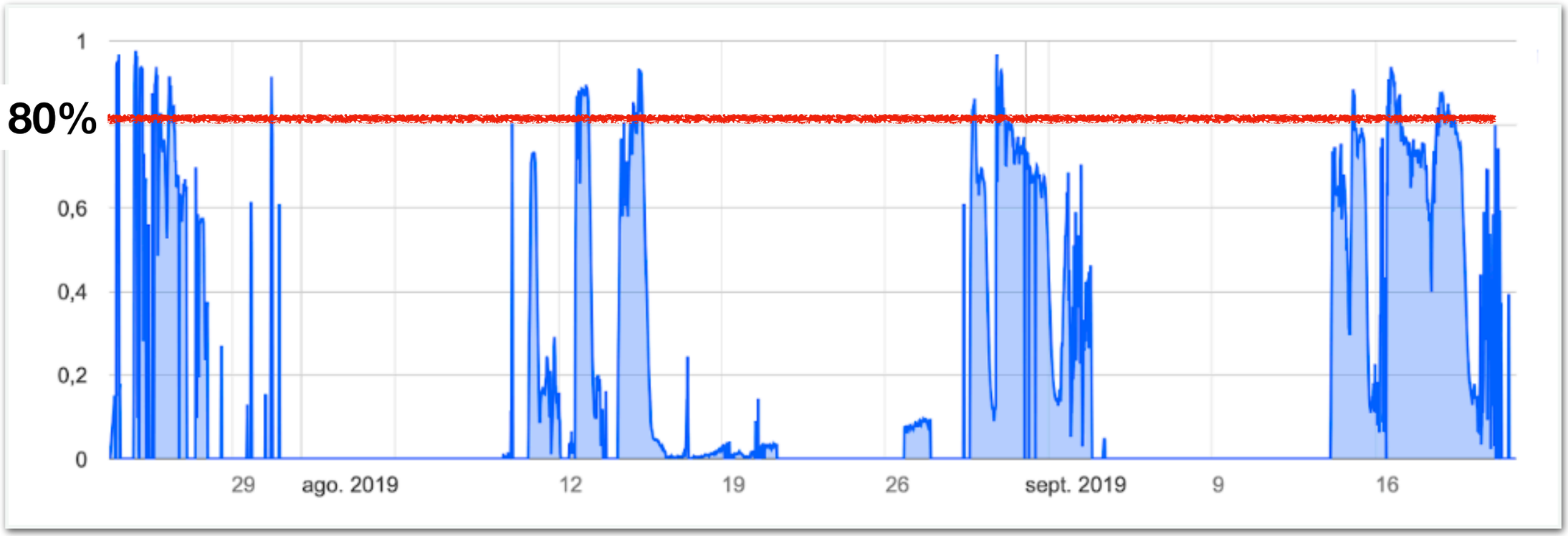
1. Re-run with 2018 set-up (HTCondor 8.7.8). Problems: slot updates saturating UDP buffers; VM crashed due to memory starvation at ~450,000 dynamic slots.
2. Upgraded to HTCondor 8.9.2. Reduced slot update rates and first attempt to integrate multi-threaded Negotiator. Problems: Top collector forwarding updates to itself. Lots of help from Jaime to solve!
3. More UDP and CCB configuration tuning (see backup slides)
4. New 256GB central manager machine deployed and fully integrated the multi-threaded Negotiator.



Round 1 **Round 2** **Round 3** **Round 4**

Scale Test Preliminary Conclusions

- Achieved 450,000 running jobs in dynamic slots.
- However, only 75-80% slot occupancy (plot below). 😕
- Collector duty cycle ~100% in all rounds. Limitation is processing updates in top collector, **hitting maximum UDP buffer size limit of $2^{32}-1$ in Linux.** 😱 Throwing more memory at the problem did not solve the problem.
- Multi-threaded Negotiator: Negotiation time within reasonable values (5-10 minutes) for all rounds. Greatest improvement in matchmaking phase seen in the final round. 😀
- Schedd's: Max job start rate ~4Hz, 150K autoclusters. Maximum running jobs per schedd of 50K. 😀



Future Rounds of Scale Testing

- HTCondor 8.9.3 (released on September 12th) has configurable MAX_UDP_MSGS_PER_CYCLE which may help by processing UPD updates more quickly.
    - Other ideas we have heard about to alleviate this top collector update bottleneck: binary ClassAds, differential updates, no top collector
- Limitations on number of federated pools
- Measure maximum schedd job start rates with realistic job payloads - CMS sandboxes up to 100MB
- Improve scheduling efficiency, i.e. no idle CPU when there is still sufficient job pressure in the queues.

Site-customizable Start Expressions

- The emergence of specialized resources attached to sites means that sites want to restrict the types of jobs that can run on them. Use cases:
  - Only run production (not user analysis) jobs on their HPC or opportunistic resources, e.g. BEER at CERN
  - Only run specific users' analysis jobs on DODAS-instantiated resources, not production
  - Run only jobs that require no external network connectivity

- Method: site-customizable (append) HTCondor start expressions, read from a standard location in CMS file space.

- GlideinWMS developers interested in implementing a generalized solution.

Difficult Use Cases

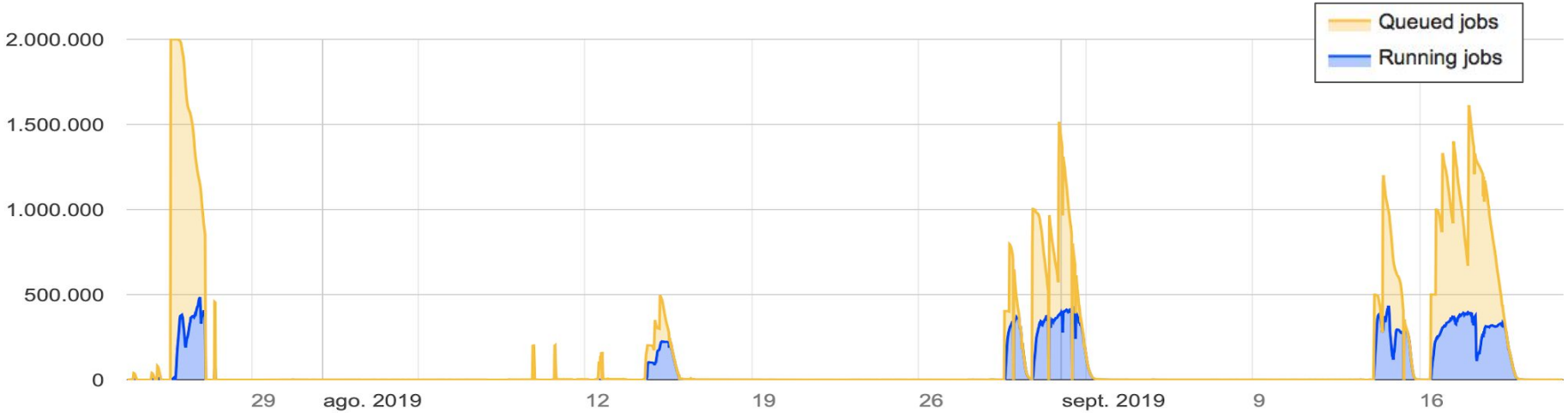
- All of the submission infrastructure serves CMS Workflow Management (WM) for the needs of production and physics analysis.
- WM is the only major project that SI interact with that is not a community project, such as HTCondor, glideinWMS, Rucio (data management), MonIT (monitoring), CRIC (information services).
- Difficult WM use cases:
  - Resource-based fair share, i.e. allow analysis to run on at least 25% of the slots at any given Grid site.
  - Scheduling network, i.e. don't kill sites with too many jobs with remote data reads over the WAN, or with too many high-IO jobs on a sites LAN.
  - Workflow prioritization: We often have several high-priority workflows and would like to manage (or predict) their throughput, without completely starving the rest. Hard to do! Will Job Sets help?

Conclusions

- CMS thanks the HTCondor developers for their close cooperation during the past several years!
- With their help, we have grown the CERN & Global Pool scales to over 300,000 CPU cores running 150,000 jobs at peak.
- Expect increasing scale increases in the 2020's, as well as more complexity (HPC and Cloud, e.g.)
- Scale testing ongoing to find (and fix) potential limitations in HTCondor and GlideinWMS before we find them in production.
  - Achieved scales of ~450K dynamic slots (jobs), ~3x higher than currently in production, but with poor scheduling efficiency (80%)
  - Collector limited by processing updates.
  - Multi-threaded Negotiator ready for production!
  - Schedd's scaling well up to 50K jobs/schedd.

Backup Slides
(Details of Scale Tests)

Overview of scale test rounds



Old CM host
Old HTCondor: 8.7.8
job_t=8 +/- 1 h

→

Old CM host
New HTCondor: 8.9.2
job_t=8 +/- **2h**
Retuned parameters (*)

→

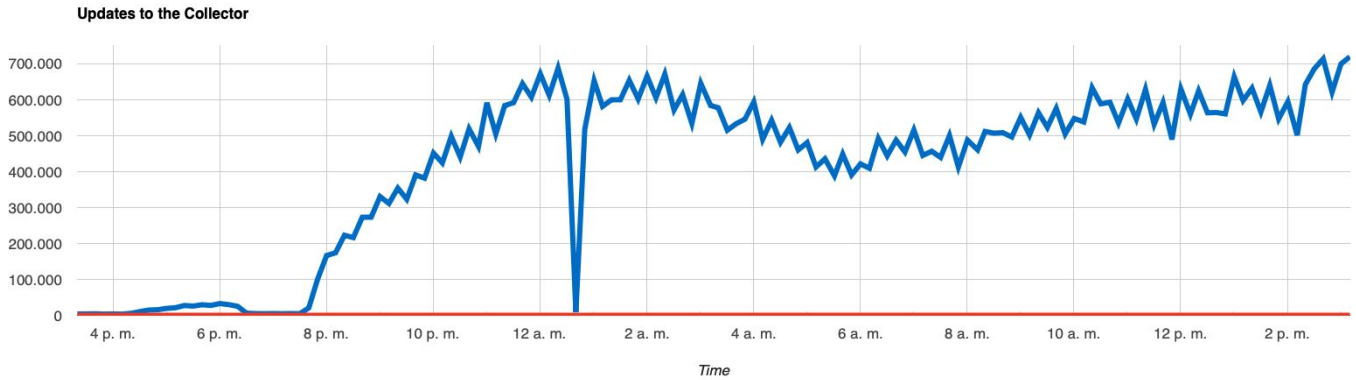
Old CM host
Continue re-tuning parameters (*)

→

New CM host
Max buffer value
Multithreaded negotiator

First round results

- Slot updates from the child collectors saturating the UDP buffer (2xCOLLECTOR_SOCKET_BUFSIZE = 512MB)
  - Duty cycle saturated also at 100%, at 700k updates per 20 min window
  - Top collector missing updates (state, activity)
    - Inefficient matchmaking
    - Pool view (from condor_status) not reliable
- The memory usage in the CM host gradually growing with scale of the pool.
  - Average at 85 GB with spikes at 115 GB observed (additional collector workers)
  - Collector workers at **>25 GB each**
- The whole VM crashed when the size of the pool was about **450k running jobs**



CM parameter tuning

- After the first round, we upgraded HTCondor to 8.9.2 in the CM
- Trying to improve scalability of the collector by **reducing slot update rates**, CM configuration was tuned in multiple parameters
  - CLAIM_WORKLIFE: 0 to 12h
  - CLASSAD_LIFETIME: 600 to 1500 s
  - Increase COLLECTOR_SOCKET_BUFSIZE = 1 GB not working
  - COLLECTOR_QUERY_MAX_WORKTIME no limit, to 120s
  - HANDLE_QUERY_IN_PROC_POLICY from default to “never”

- Aim at **faster matchmaking**:
  First attempt at NEGOTIATOR_NUM_THREADS = 4
  NEGOTIATOR_RESOURCE_REQUEST_LIST_SIZE: from 500, increased to 1000.

- Extended period before idle slots are released: GLIDEIN_Max_Idle: from 600 to 1200s

Issues during the tests

- After upgrade to 8.9.2, observed **top collector forwarding updates to itself**. Spent some days trying to figure out what was going on… kindly solved by Jaime by adding

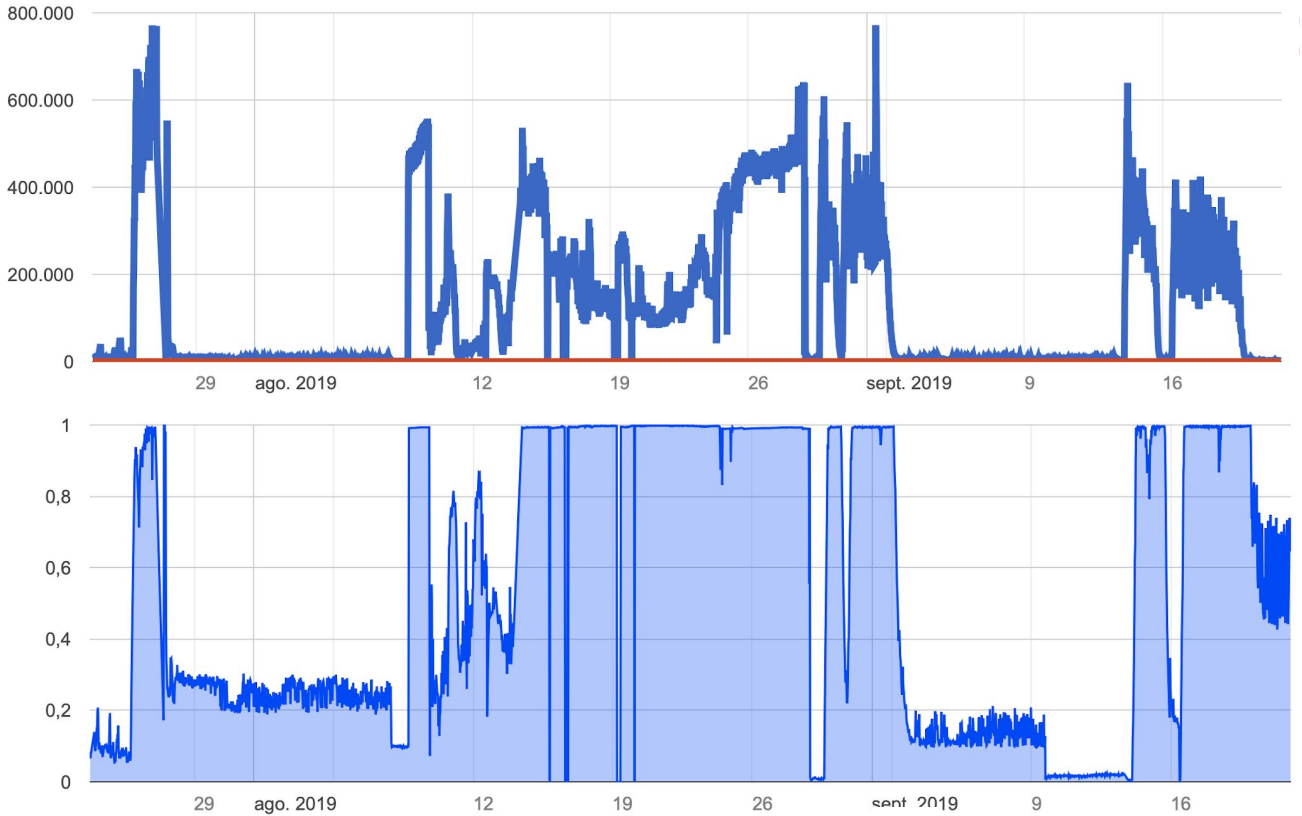
COLLECTOR_FORWARD_WATCH_LIST = State,Cpus,Memory,IdleJobs,Activity,**DaemonStartTime**

- Discussed potentially suboptimal UDP packet fragmentation , not affecting our case as we already had UDP_NETWORK_FRAGMENT_SIZE=60000

- Setting up the max buffer size: Saqib had to fine tune it to “1 GB - Epsilon”, otherwise, it wouldn’t accept 1 GB total (COLLECTOR_SOCKET_BUFSIZE = 1024*1024*1024 does not work)

- CCB shared port daemon running out of **file descriptors**, limits successively increased, from the default at **4096, to 65536** (SHARED_PORT_MAX_FILE_DESCRIPTORS)
- Max **connection tracking limits** on the CCB being hit, also required increasingly larger values (/proc/sys/net/netfilter/nf_conntrack_max, from **262144 to 585552**)

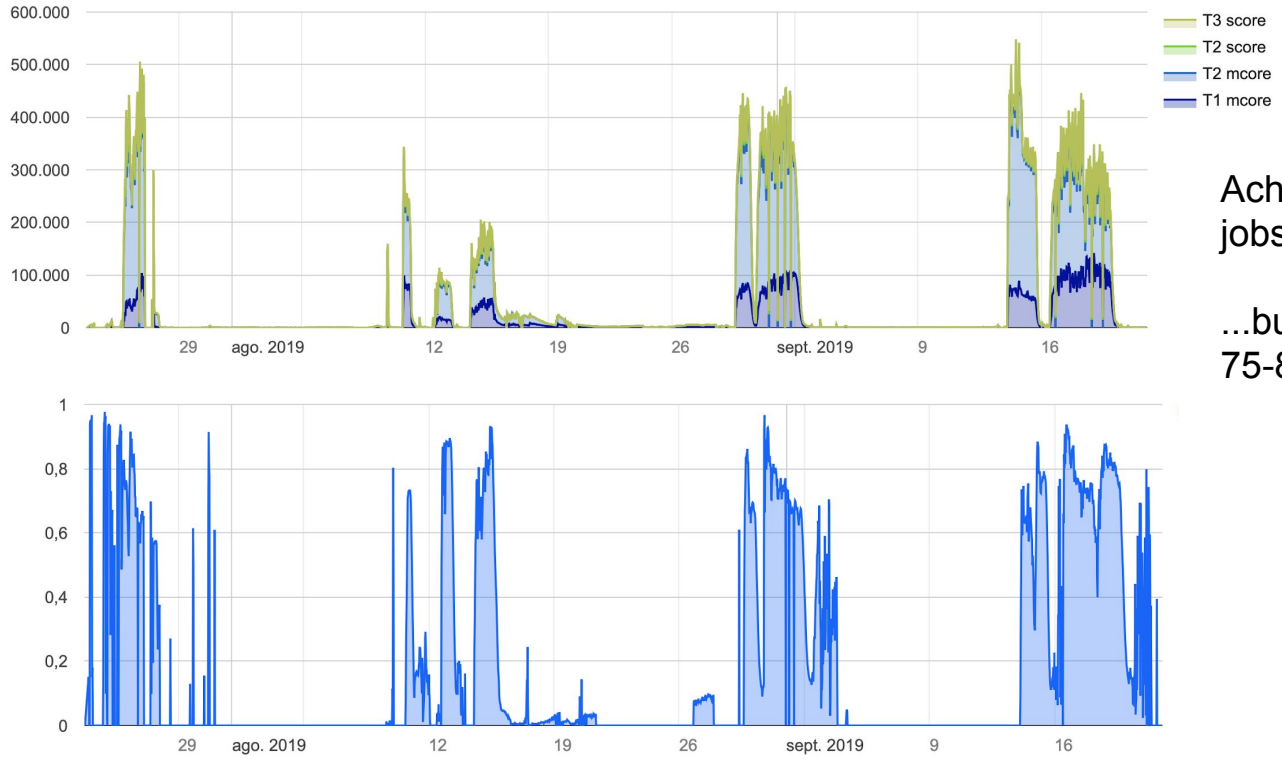
Slot updates & collector duty cycle



Even after reducing slot update rates, collector duty cycle is still saturated.

Missing updates, leading to collector data loss, with worse knowledge of the pool's slot status and thus worse matchmaking performance

Pool size vs efficiency



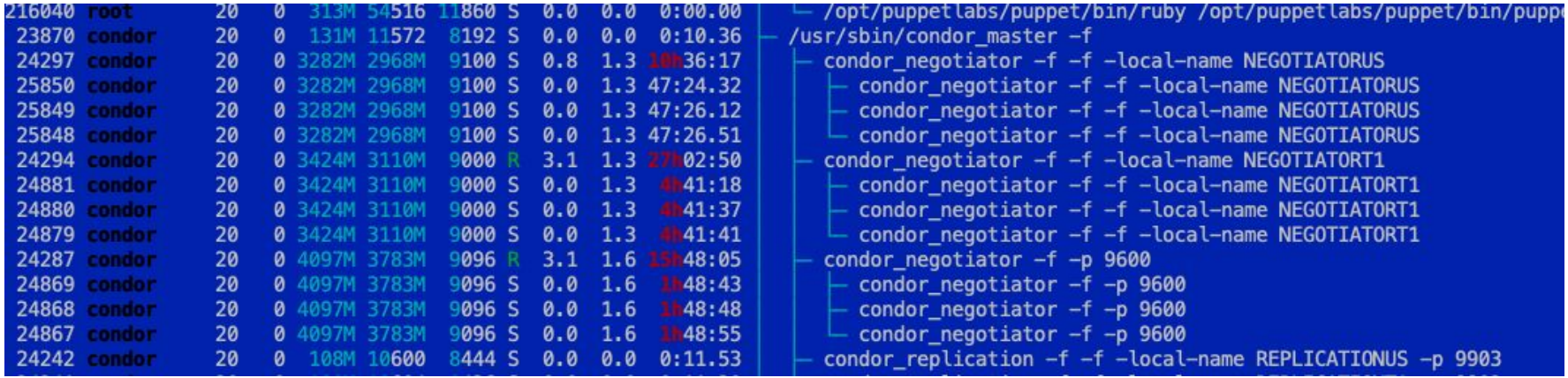
Achieved close to 450k running jobs...

...but with poor pool efficiency at 75-80%

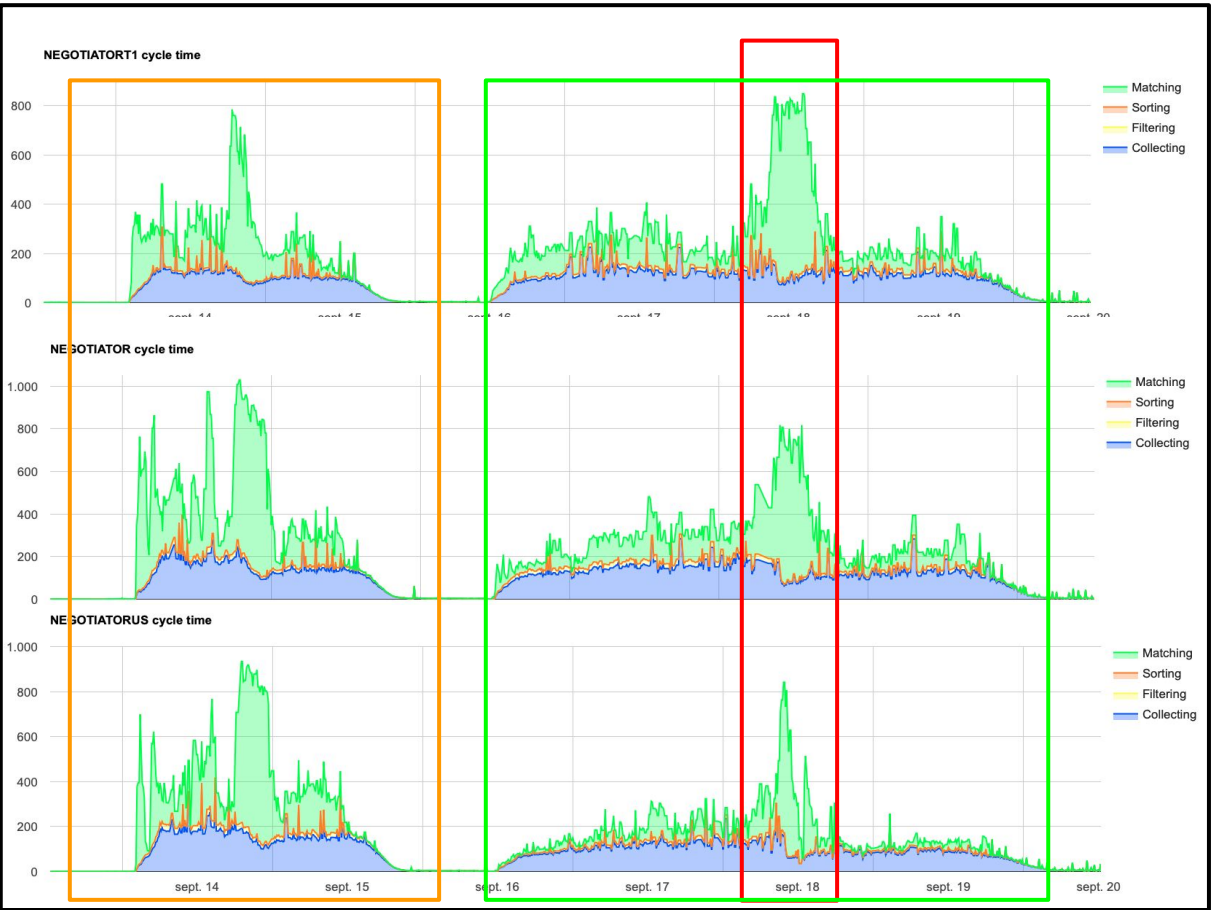
Negotiator (Central Manager)

- Negotiation time within reasonable values (5 to 10 minutes) for the whole testing period

- In the final round, with multithreaded nego on, improvement in reducing matchmaking phase duration (and overall nego cycles)



Negotiation cycle



Compare negotiation cycle time in the 2 rounds launched in the last phase (Sept 15h onwards):

- Multithreaded Negos **OFF**
- Multithreaded Negos ON

Please ignore region with **LHC One network issues** at CERN in the middle of second run, lasting for about 12h

While pool scales and job pressure are similar, **matchmaking phase of the negotiation cycle is clearly reduced**

Ready to be used in real pool, where **matchmaking is the dominant component** of nego cycle

September 25th, 2019

Last tests

Some notes on the last test phase (September 15th onwards)

- Using our new CM host (24 cores and 256 GB RAM), plus incremental changes in configuration implemented along the way
- Collector updates peaking at 600k in 20 mins (**500 Hz**), UDP buffer saturated, collector duty cycle at 99.9%
  - Got close to **450k running jobs** (about a factor 4 in dyn slots compared to current global pool)...
  - ...But with **pool efficiency around 75%** (>100k CPU cores idle!)
- CPU load on CM at 50%, but peaks on number of running processes over 24
  - Do we need our 40 cores back?
- Memory usage peaking at 130 GB, but no collector crashes this time thanks to increased memory
  - Room for increased number of collector workers?
- Tested multithreaded Negotiators ON: as described, matchmaking phase of the negotiation cycle significantly reduced
  - **Ready to be used in real pool**

Schedds

Some notes on schedds performance through the tests: **working OK**

- Duty cycle ok (peaks at 80%, average at 60%), memory not saturated
- Max simultaneous running jobs approaching 50k per schedd,
  - total max running jobs at 480k in one of the rounds
- Increased dispersion in job lifetime (8+/-2h), to reduce synchronization on the termination of jobs
  - which also produces bursts in job start
- Job start rate measured from RecentJobStarts: handling max **45k job starts in total**
  - **4.5k per schedd on average over 20 minutes: about max 3.75Hz per schedd**
- Number of Autoclusters (with queued jobs) peaking at 150k
  - At least a factor 3 higher than measured in our current global pool

Still no results on performance with realistic input sandboxes though