Evolution of the CMS Global Submission Infrastructure for the HL-LHC Era

Antonio Pérez-Calero Yzquierdo
on behalf of the Submission Infrastructure team of the CMS Experiment

CHEP 2019 Adelaide, Australia, November 4th 2019

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES
Ciemat Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas
EXCELENCIA MARÍA DE MAEZTU
cfp CIEMAT física de partículas
PIC port d'informació científica

Outline

- The CMS Submission Infrastructure (SI) and the Global Pool
- Exploring HTCondor scaling boundaries
- New features in CMS Sub. Inf. for increased flexibility
- Conclusions

CMS SI Group Charge

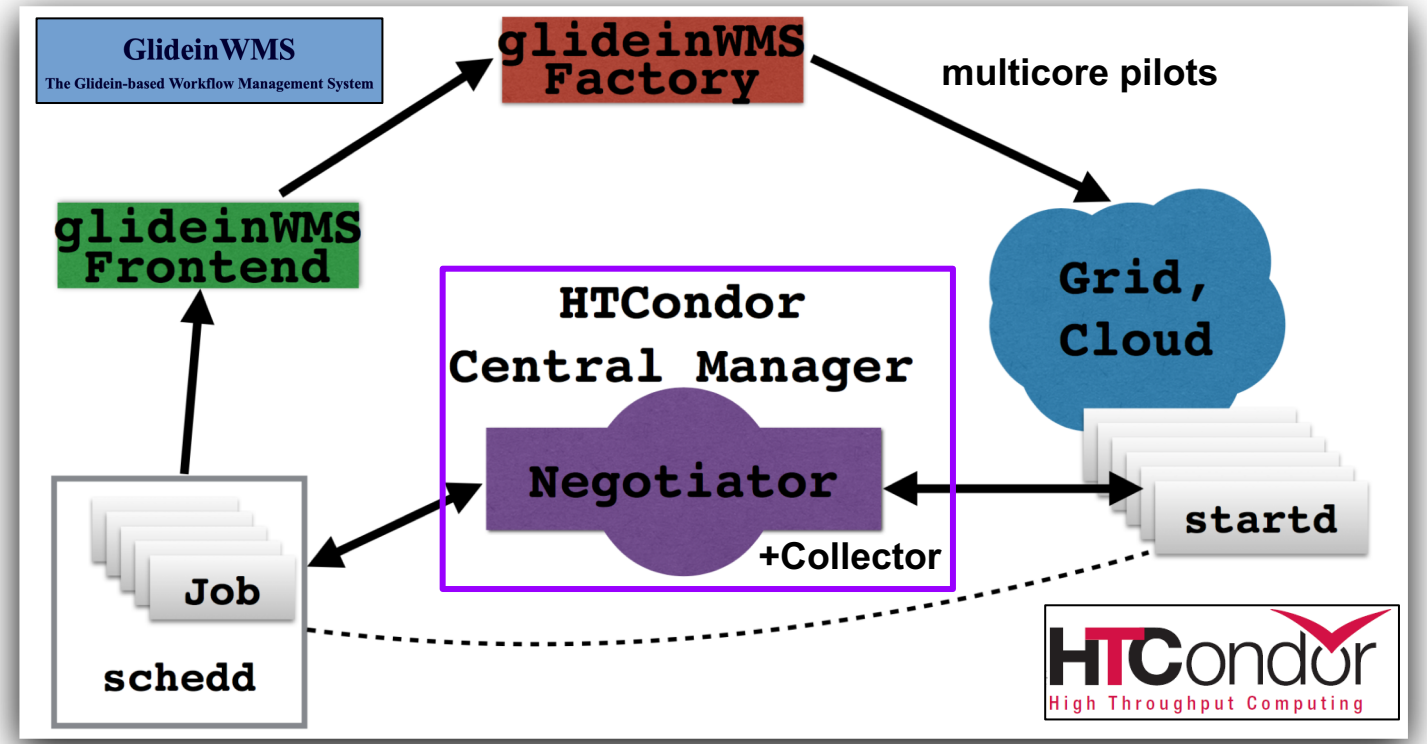
- The charge of the **Submission Infrastructure (SI) Group within the CMS experiment** at CERN is to **run the infrastructure** in which all processing, reconstruction, simulation, and analysis of physics data takes place

- In order to do that, we
  - **Manage** GlideinWMS and HTCondor pool operations in CMS
  - **Communicate** CMS **priorities** to the development teams, to suggest and integrate new features

- SI activities broadly fall into several categories:
  - Overcoming **current operational limitations** or problems
  - Preparing for **future scales** and feature requirements (mid and longer-terms problems)
  - Integration of new, diverse **resource types** and **job submission methods**

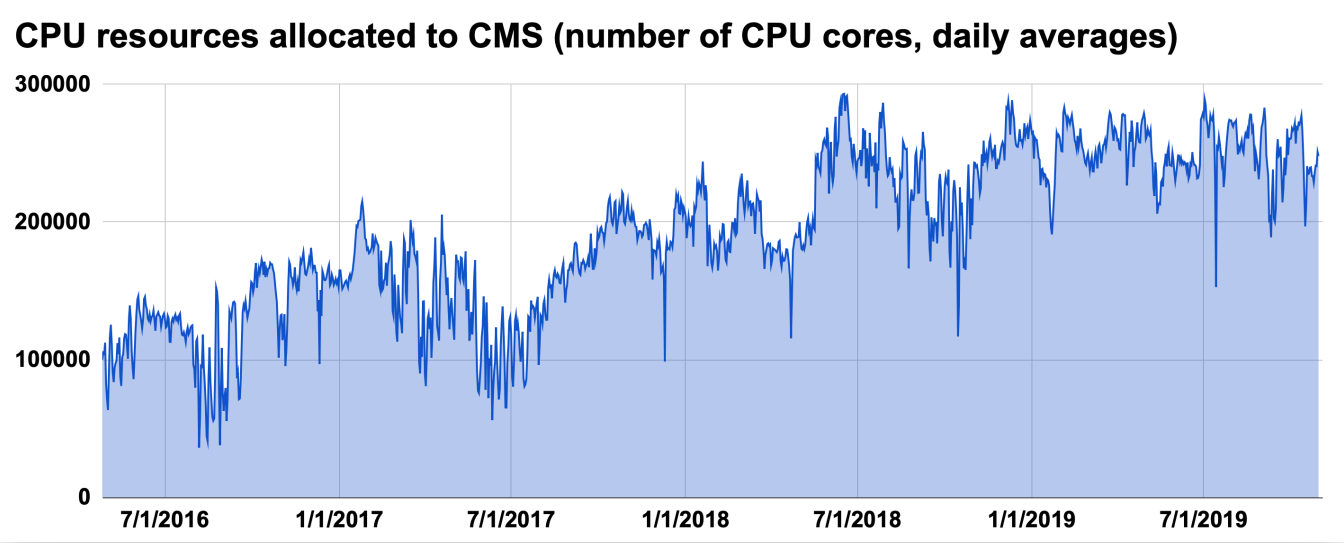
Classical view of CMS Global Pool

The **CMS Global Pool** is both a **GlideinWMS instance** and a **HTCondor pool**.



Increasing scales: Run2

- **Size of the CMS Global Pool** under sustained growth, doubled in size in the past 3 years driven by increasing resource **requests during LHC Run 2**. **Our main HTCondor pool running routinely at 250k cores**
  - **300k cores in peak including the CERN pool**
- During Run 2 and LS2, progressively adding opportunistic (beyond-pledge) resources
  - **HLT farm** commissioned as opportunistic resource, when not in data taking mode
  - Increasing proportion of beyond pledge (**opportunistic-Grid**) and **HPC, Cloud** resources to the mix

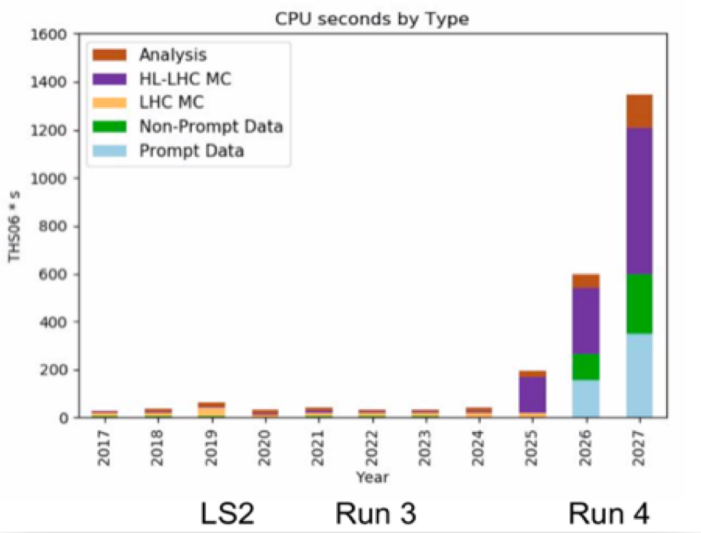


Increasing scales: Run3 and HL-LHC

- CMS processing needs **projected into HL-LHC** era (see e.g. HSF roadmap): grow with increasing number of events collected, and with increased event complexity (PU 200)
- Growth factor predicted at **~20 x CPU compared to current levels by 2027 (*)**
  - Moderate progress in further job parallelization is still expected, reducing the load on the WM infrastructure.
  - Reduce the fraction of workflows to **remain single-core**

- **How many schedulers** will we need to handle such a load?

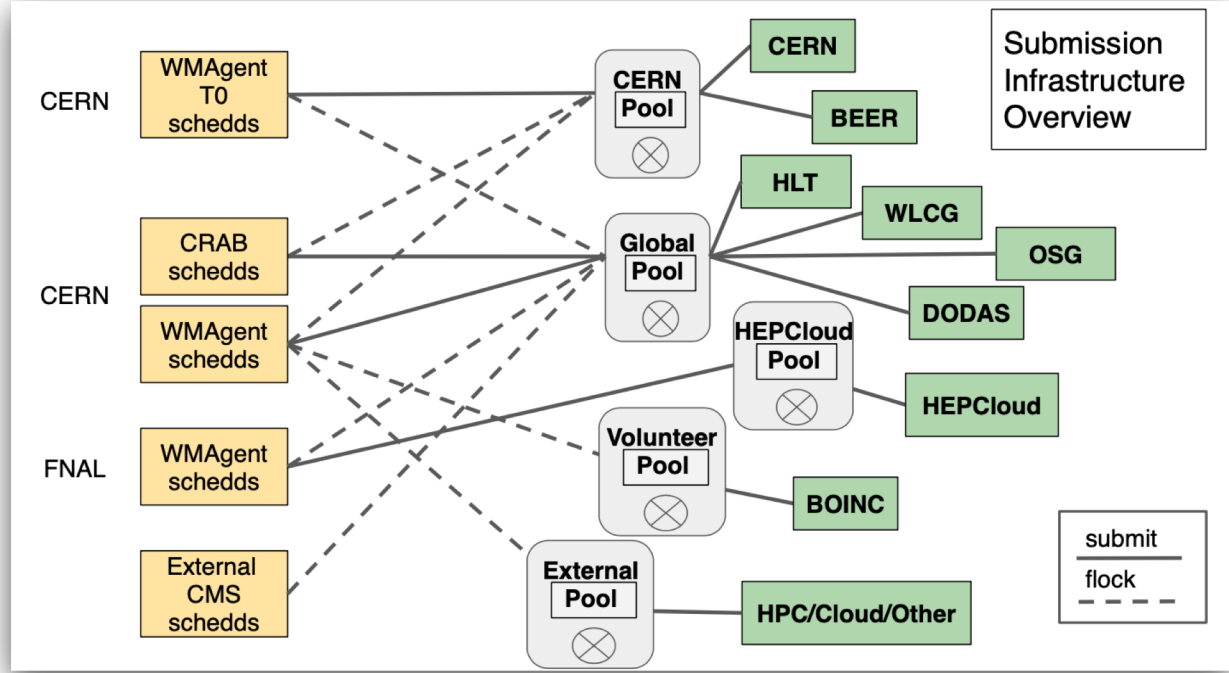
- Growth in CPU resources is likely **not to be obtained from increasing capacities at Grid sites**: increasing fraction of **HPC resources** in the mix



(*) CMS HL-LHC foreseen requirements soon to reviewed, likely to be reduced

Increasing complexity: Multiple Pools

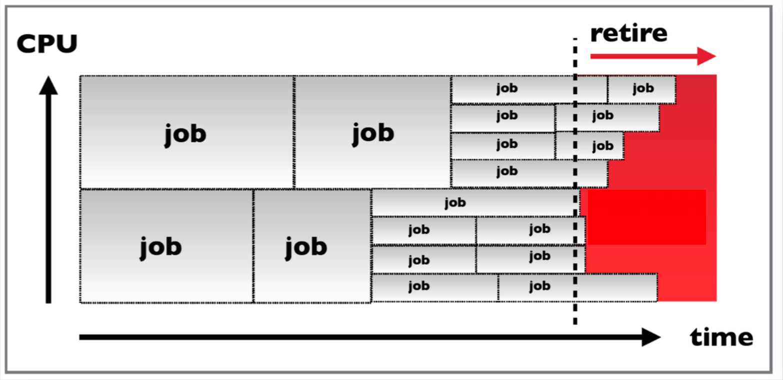
- From a single Global Pool model, CMS Submission Infrastructure **has evolved** into a model of **multiple Federated Pools with centralized submitters**
- Resources are still mostly acquired with **GlideinWMS pilots**, but now also include **vacuum-like instantiated slots** (e.g DODAS), **BOINC** (CMS@Home), **opportunistic** (HLT), etc



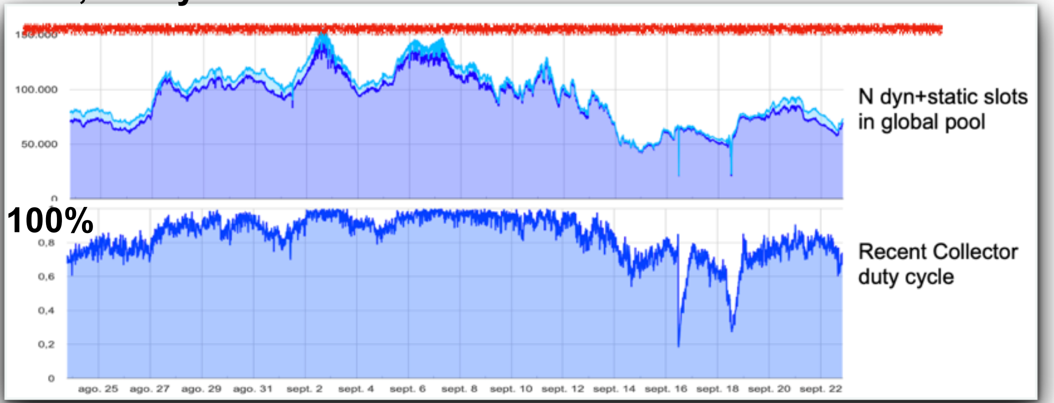
Multi-core pool

- CMS moved to **multi-core pilots** by start of Run 2, supporting **dynamic partitioning** of slots (e.g. CPU, Memory)
  - **Partitionable slots** become fragmented into **dynamic slots, each executing a job**

- **Global Pool dynamic fragmentation** depends on composition of the **CMS workload** (**single and multi-core requests**)
  - **Load** on the Global Pool central infrastructure (***collector***) driven by **number of dynamic slots**
  - **Scalability limits** hinted when our 300.000 cores are divided up to 150.000 running jobs (average thread/job down to ~2, ***single-core job storms***)



150,000 dynamic slots



Challenges

CMS SI needs assessing the **scalability and stability** of our infrastructure with **dedicated tests**:

- Continue exploring limits for a **single pool, pushing dynamic slot limits to higher scales**
  - Gain **increased headroom, so single-core job storms** do not result in current setup vulnerabilities
  - **Anticipate peaks** from out-of-pledge resources (HPC, Cloud) joining the pool in bursts
- Scalability of **federated HTCondor pools**:
  - How many different pools (***negotiators***) can schedulers actually talk to?
- Explore scheduler **submission rate limits (**1 Hz, 10 Hz?**)**
  - Need sufficient **submit capacity** to fill continuously growing resources
  - Test with **realistic I/O sandboxes**
- Study and utilize the **multi-threaded negotiator** to keep matchmaking time under control

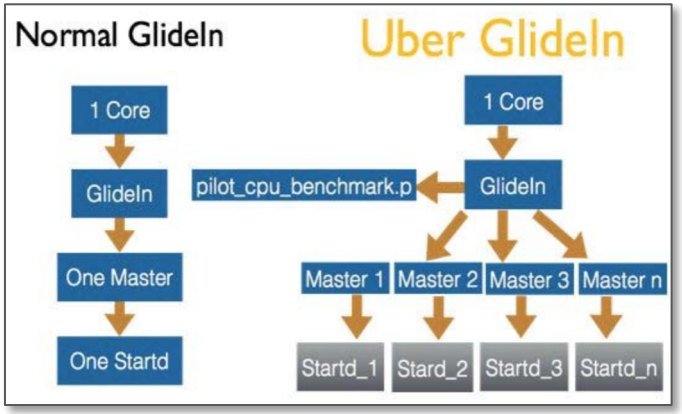
- Continuous re-assessment of general improvements in the **GlideinWMS and HTCondor software** to higher scales, while retaining
  - **high efficiency in pool usage**
  - reliably enforcing CMS **workload prioritization** policies

Scale Tests

- We can generate increasingly larger test pools (ITB) by running multiple startd's per CPU core (*uberglideins*)
  - Simulate a **500.000 startd HTCondor pool** with ~16k physical cores

- Our test HTCondor pool central manager runs on a **256 GB RAM host** at CERN, with **3 negotiator instances**, like the production pool
  - Enabled the multithreaded negotiator option
  - CCB running on a separated host (32 cores, 60 GB RAM)
  - 10 schedds (32 cores, 60 GB RAM) provide job submit pressure

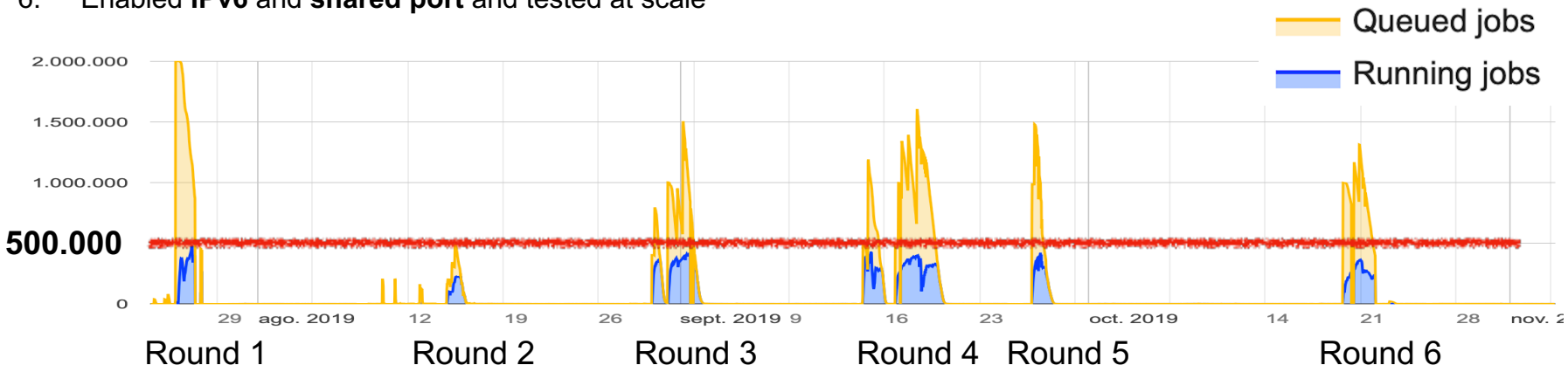
- Workload for the tests:
  - Up to **2 million jobs** in total in schedd queues
  - all **single-core** jobs to **maximize pool fragmentation**
  - **randomly generated jdls** for max **job diversity**
  - Gaussian **job length** (e.g. 8h+/-2h)



2019 scale test rounds

Scale test rounds launched from July to October 2019, technical discussion held at EU HTCondor Workshop in Italy in September, see backup slides for further details

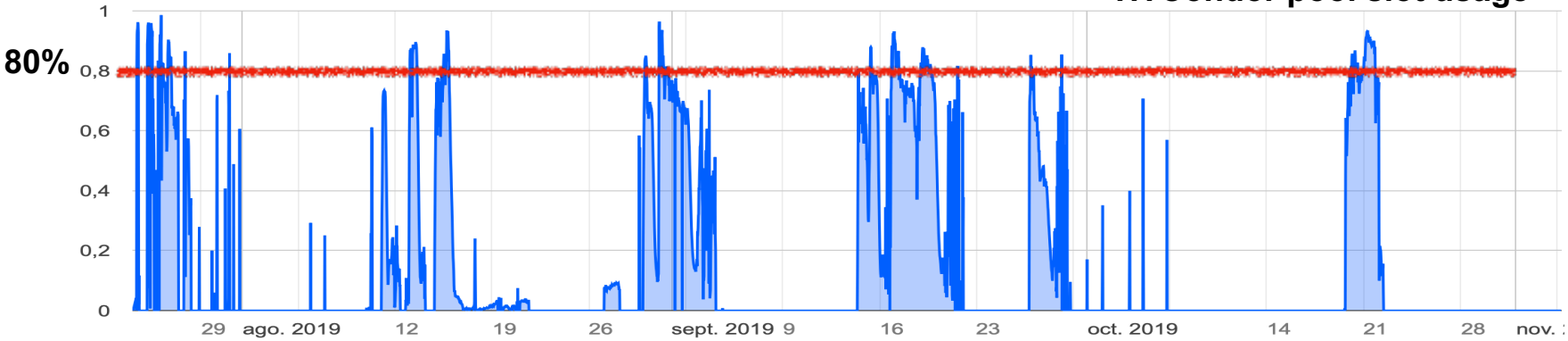
1. Re-run with **2018 set-up (HTCondor 8.7.8).** Problems: slot updates **saturating UDP buffers**; CM host VM **crashed** due to memory starvation at **~450,000 dynamic slots**.
2. Upgraded to **HTCondor 8.9.2**. Reduced slot update rates and first attempt to integrate multi-threaded Negotiator. Problems: **Top collector forwarding updates to itself**, required developers help to fix
3. Further CM (UDP buffer size) and CCB **configuration tuning**
4. New **256 GB RAM central manager host** deployed and fully integrated the multi-threaded Negotiator
5. New test with **HTCondor 8.9.3**, latest development series release
6. Enabled **IPv6** and **shared port** and tested at scale



Summary test results

- Achieved **450,000 running jobs in dynamic slots**
  - Observed **degraded slot usage**: **~80% of slots busy**, some improvement in the last round (plot below)
- **Collector duty cycle ~100%** in all rounds: limitation in **top collector processing slot updates**,
  - Overflowing UDP buffer size limits, **missing slot updates**
  - **Ideas** on how to solve the top collector bottleneck **already being examined** with HTCondor developers team
- **Multi-threaded Negotiator**: **negotiation time within reasonable values** (5-10 minutes) for all rounds, despite increased pool sizes. Greatest improvement in matchmaking phase seen in the final round
- **Schedd's**: maximum job start rate **~4Hz**, maximum running jobs per schedd of **50K**, handling **150K** job autoclusters
  - **Late materialization** of jobs could help further!
- **IPv6**: **1/4 of the startds** connected to the central manager **over IPv6**
  - Also **1/3 of the shadows** connecting to IPv6-enabled test schedd

HTCondor pool slot usage



Site-customizable pilots

New feature: The emergence of **specialized resources** attached to grid sites means that local admins may want to **restrict the types of jobs** that can run on them.

A number of use cases has emerged:

- **Only run production** (not user analysis) jobs on their opportunistic resources, e.g. BEER at CERN
- Only run **production jobs related to a certain physics study** for which an specific resource allocation has been granted, e.g. HPC, with jobs being filtered based on dataset name, physics analysis group, etc
- Only run specific users' analysis jobs on DODAS-instantiated resources, not production
- Run only jobs that require no external source for input data

Method:

- GlideinWMS pilots carry a **standard start expression**, enabling **generic late-binding conditions**.
- Pilots enabled for local customization **read additional constraints** (from a standard location in CMS file space) appended to the default start expressions
- GlideinWMS developers interested in implementing a **generalized solution**

Status: Functional implementation available, first use cases being addressed (e.g. BEER)

Further refinements in CMS SI

We are in close collaboration with the HTCondor developers in finding technical solutions to certain difficult WM-related use cases for CMS:

- **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.
- Enhanced **workflow prioritization**: Multiple high-priority workflows are often found in the queues, a better **management** (or prediction) of their throughput and time to completion would be important for CMS Computing Ops. Not an easy task nowadays

New concept of HTCondor Job and Resource "Sets" promises to tackle such challenges, while also reducing combinatorics of the matchmaking process (scalability).

Many other **interesting GlideinWMS and HTCondor features in the pipeline**:

- Support for GPU-slots (heterogeneous resources)
- Late materialization of jobs (WM scalability)
- SciToken support (GSI migration)
- File transfer improvements
- REST-based API for HTCondor, using the Python bindings [condor_restd].

Conclusions

We in **CMS SI** continue exploring potential limitations in our current infrastructure to **detect bottlenecks preventively and assess scalability into CMS projected needs**

- Expect **increasing scales** in the 2020's, as well as **more complexity** in the resource mix (Grid+HPC+Cloud)

Launched a series of scale tests pushing the size of our test pool up to about **450K running jobs**

- Found **scaling limitations on a key component**: **Top collector**
  - Limiting factor in the speed to process slot updates from secondary collectors, poor knowledge of the pool status leading to degraded matchmaking and **poor slot usage (80%)**
- Nevertheless, limitations found at **factors** in size with respect to current our production pool
  - **not worried about the immediate scale** limits
  - **Bottom line**: even with the present setup, scale should be fine for Run 3
- **Negotiation cycle improved thanks to multithreaded negotiators**
- **Schedds performing successfully**

A number of additional tools are being discussed ands develop to enable CMS SI with further flexibility and reach

Acknowledgement: We wish to thank the GlideinWMS and HTCondor developers as well as CERN IT teams for their continued support and close collaboration

Additional slides

Abstract

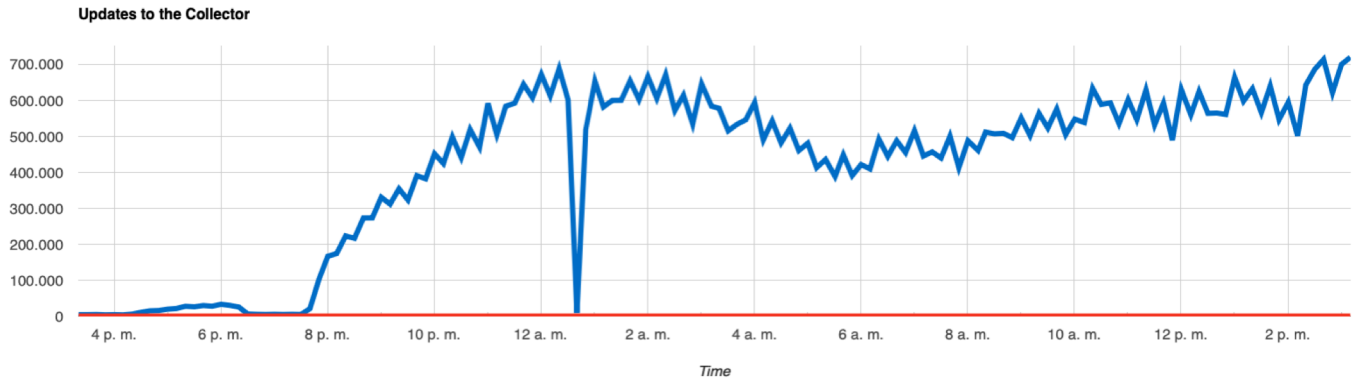
Efforts in distributed computing of the CMS experiment at the LHC at CERN are now focusing on the functionality required to fulfill the projected needs for the HL-LHC era. Cloud and HPC resources are expected to be dominant relative to resources provided by traditional Grid sites, being also much more diverse and heterogeneous. Handling their special capabilities or limitations and maintaining global flexibility and efficiency, while also operating at scales much higher than the current capacity, are the major challenges being addressed by the CMS Submission Infrastructure team. This contribution will discuss the risks to the stability and scalability of the CMS HTCondor infrastructure extrapolated to such a scenario, thought to be derived mostly from its growing complexity, with multiple Negotiators and schedulers flocking work to multiple federated pools. New mechanisms for enhanced customization and control over resource allocation and usage, mandatory in this future scenario, will be also presented.

Related talks at CHEP2019:

- CMS modeling for HL-LHC
- CMS Strategy for HPC resource exploitation
- Exploiting CRIC to streamline the configuration management of glideinWMS factories for CMS support
- Exploiting network restricted compute resources with HTCondor: a CMS experiment experience

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

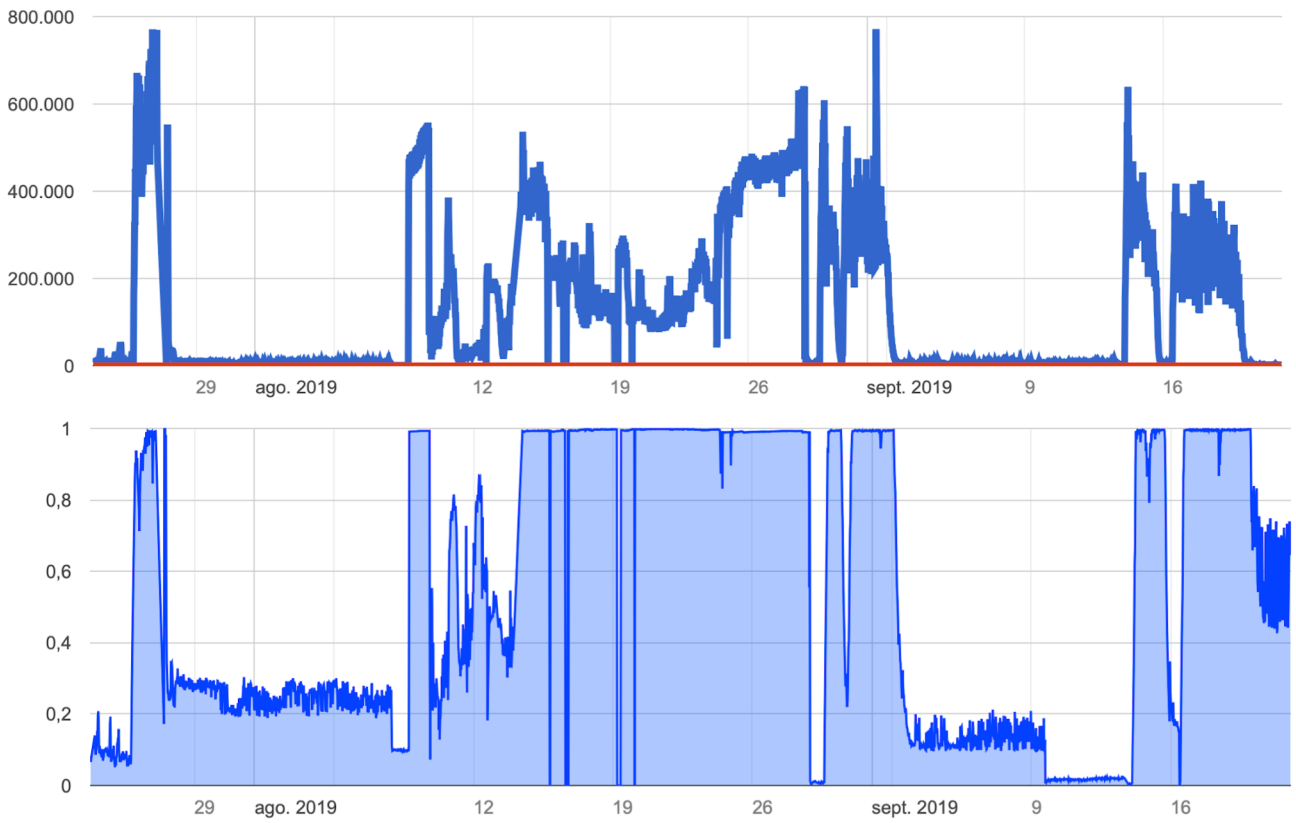
COLLECTOR_FORWARD_WATCH_LIST = State,Cpus,Memory,IdleJobs,Activity,**DaemonStartTime**

- 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**)

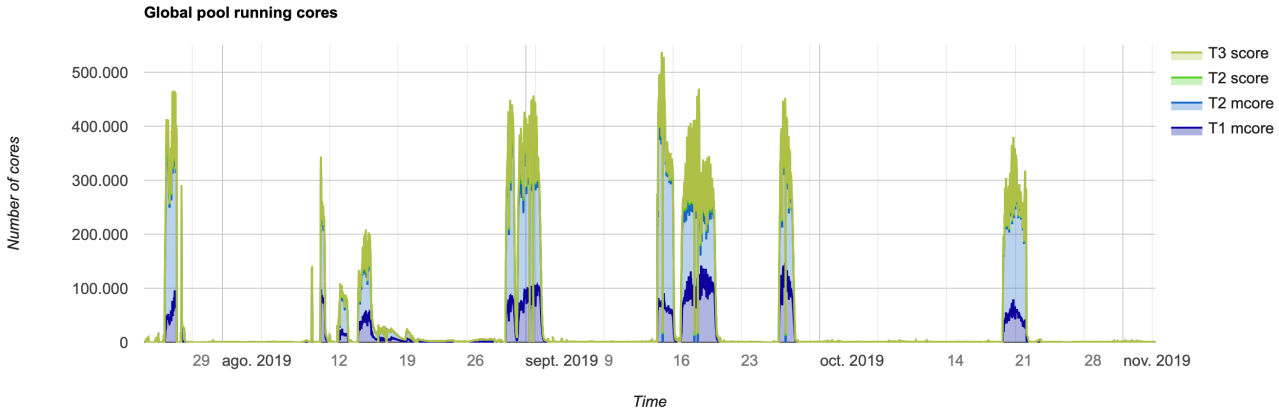
Slot updates & collector duty cycle

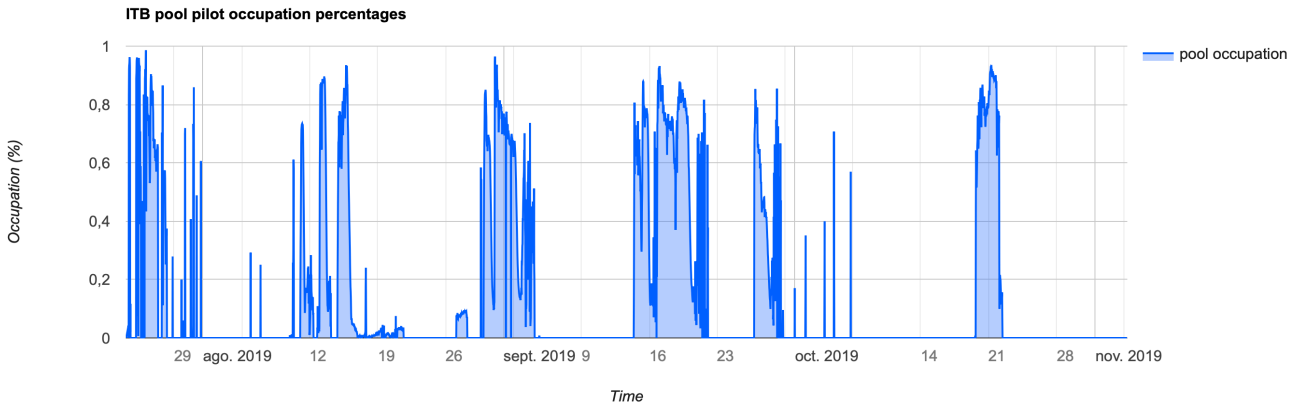


Even after reducing slot update rates, collector duty cycle is still saturated.

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 during the single HTCondor scale tests

Observed degraded pool occupatiuon efficiency at 80%

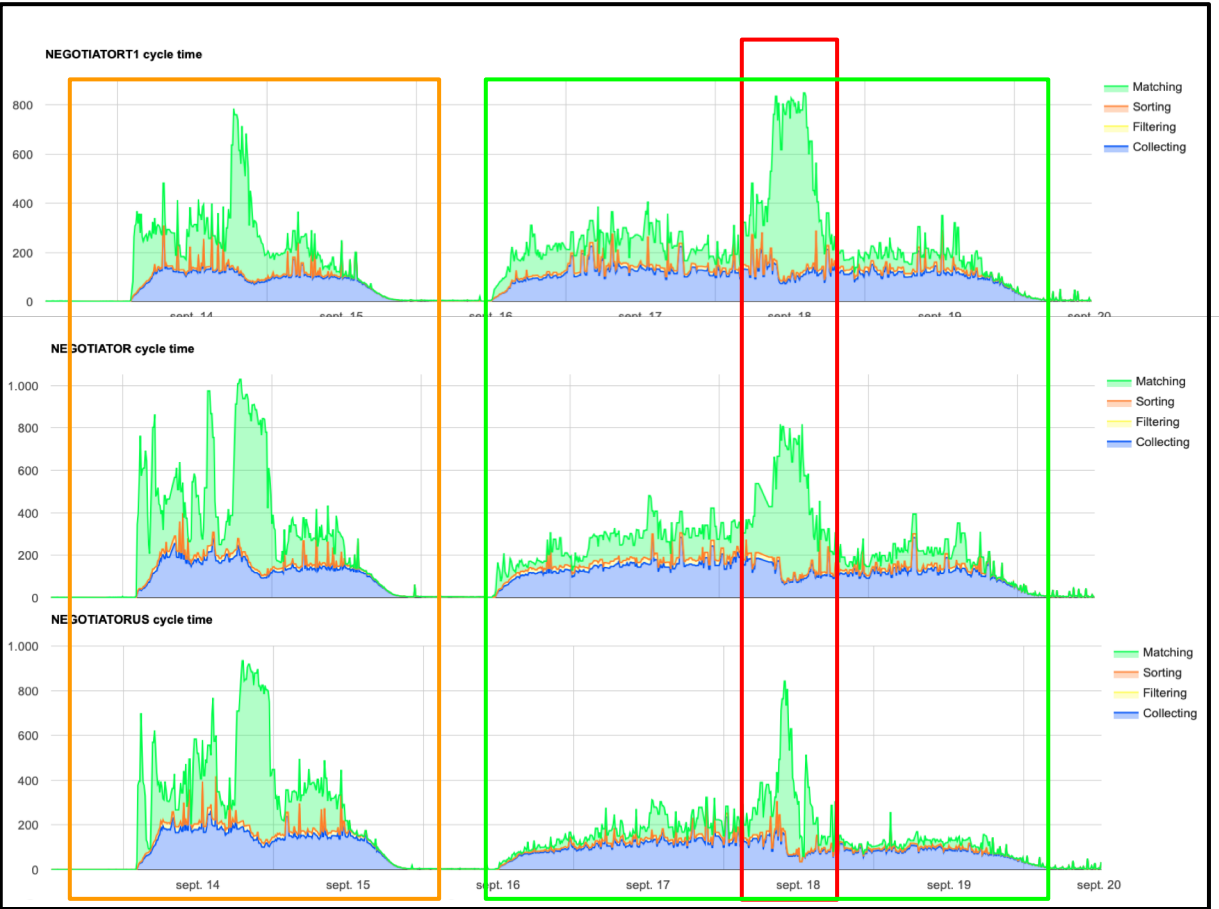
Negotiator (Central Manager)

- 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)

```
216040 root      20   0  313M 54516 11860 S  0.0  0.0   0:00.00  └─ /opt/puppetlabs/puppet/bin/ruby /opt/puppetlabs/puppet/bin/puppe
 23870 condor    20   0  131M 11572  8192 S  0.0  0.0   0:10.36 ─ /usr/sbin/condor_master -f
 24297 condor    20   0 3282M 2968M  9100 S  0.8  1.3 10h36:17   ├─ condor_negotiator -f -f -local-name NEGOTIATORUS
 25850 condor    20   0 3282M 2968M  9100 S  0.0  1.3 47:24.32   │  ├─ condor_negotiator -f -f -local-name NEGOTIATORUS
 25849 condor    20   0 3282M 2968M  9100 S  0.0  1.3 47:26.12   │  ├─ condor_negotiator -f -f -local-name NEGOTIATORUS
 25848 condor    20   0 3282M 2968M  9100 S  0.0  1.3 47:26.51   │  └─ condor_negotiator -f -f -local-name NEGOTIATORUS
 24294 condor    20   0 3424M 3110M  9000 R  3.1  1.3 27h02:50   ├─ condor_negotiator -f -f -local-name NEGOTIATORT1
 24881 condor    20   0 3424M 3110M  9000 S  0.0  1.3  4h41:18   │  ├─ condor_negotiator -f -f -local-name NEGOTIATORT1
 24880 condor    20   0 3424M 3110M  9000 S  0.0  1.3  4h41:37   │  ├─ condor_negotiator -f -f -local-name NEGOTIATORT1
 24879 condor    20   0 3424M 3110M  9000 S  0.0  1.3  4h41:41   │  └─ condor_negotiator -f -f -local-name NEGOTIATORT1
 24287 condor    20   0 4097M 3783M  9096 R  3.1  1.6 15h48:05   ├─ condor_negotiator -f -p 9600
 24869 condor    20   0 4097M 3783M  9096 S  0.0  1.6  1h48:43   │  ├─ condor_negotiator -f -p 9600
 24868 condor    20   0 4097M 3783M  9096 S  0.0  1.6  1h48:48   │  ├─ condor_negotiator -f -p 9600
 24867 condor    20   0 4097M 3783M  9096 S  0.0  1.6  1h48:55   │  └─ condor_negotiator -f -p 9600
 24242 condor    20   0  108M 10600  8444 S  0.0  0.0   0:11.53   ├─ condor_replication -f -f -local-name REPLICATIONUS -p 9903
```

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

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

Latest tests

Some notes on the last testing 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**