Operations and Monitoring of the CMS HTCondor pools

Saqib Haleem, Marco Mascheroni, Antonio Perez-Calero Yzquierdo
for the CMS Submission Infrastructure team

HTCondor Workshop Autumn 2021

Outline

- Goals in operating the CMS Submission Infrastructure
- Automated maintenance of the infrastructure
- General monitoring and metrics
- Alarms
- Historical data retention
- Conclusions

Main goals in CMS HTCondor pools operations

- Stable Infrastructure, it should *simply work* with minimal human intervention
- Scalable to CMS resource needs
- Job scheduling:
  - Users want workflows to complete in a predictable amount of time and with limited manual intervention
    - Either centrally launched with WMAgent, or personal CRAB workflows
    - Don't usually care about efficiency or cost, just time to completion and success rate
    - Sometimes there are *crazy* requirements (e.g. one CPU core + 20GB of RAM)
  - Priorities must be respected (*higher prio work should complete sooner/faster than lower prio*)
- Resources used all the time efficiently!

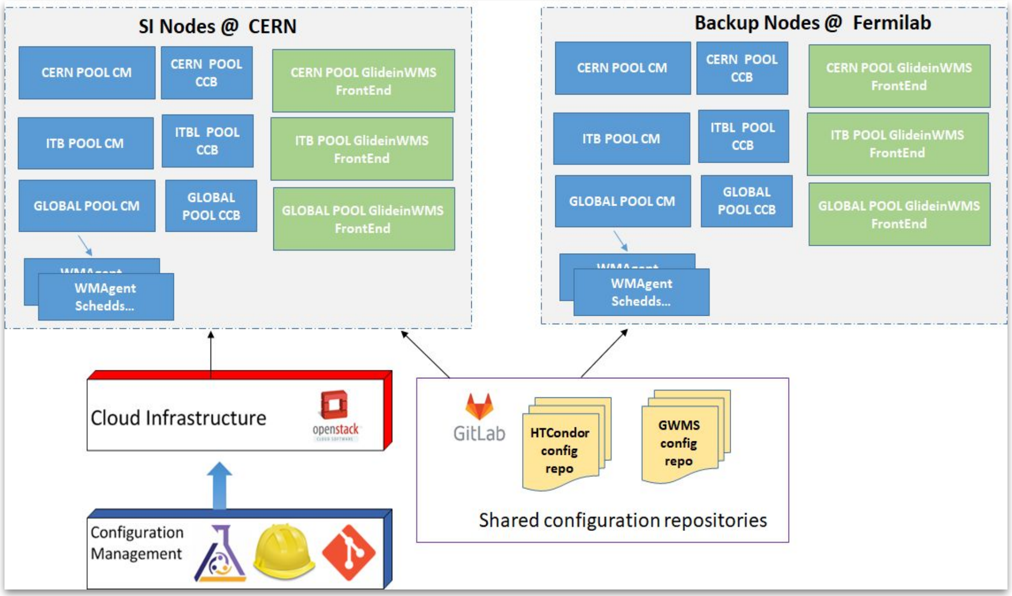
Satisfy CMS needs for processing, analysis and simulation of CMS data!

Maintenance automation

- SI is managing **70+ nodes**(VMs + Physical hosts) running different type of services (include HTCondor Collectors, CCBs, Schedds, GWMS FE, GWMS Factories, and monitoring nodes)

- All of nodes are provisioned and managed with CERN IT hosted Agile Infrastructure. (i.e. Puppet, Foreman, OpenStack, GITLab)

- We have our own separate puppet modules and host groups to automate host/service configurations.



- Shared configuration repository between CERN and FNAL(backup), to keep infrastructure synchronized.

Monitoring resources

Why Monitoring?

- CMS Submission Infrastructure is operating a complex, big and heterogeneous computing pool (as explained in previous talk)

- The resources (startds/execute nodes) in CMS Global Pool are coming from **70+ resource providers** (i.e.Tier Sites/subsites, non-grid resources)

- Moreover, SI is managing multiple pools with 50+ edge nodes (Schedds/submit nodes)

- **Monitoring system is inevitable to efficiently operate such a complex pool**

- Overall SI Monitoring dimensions:

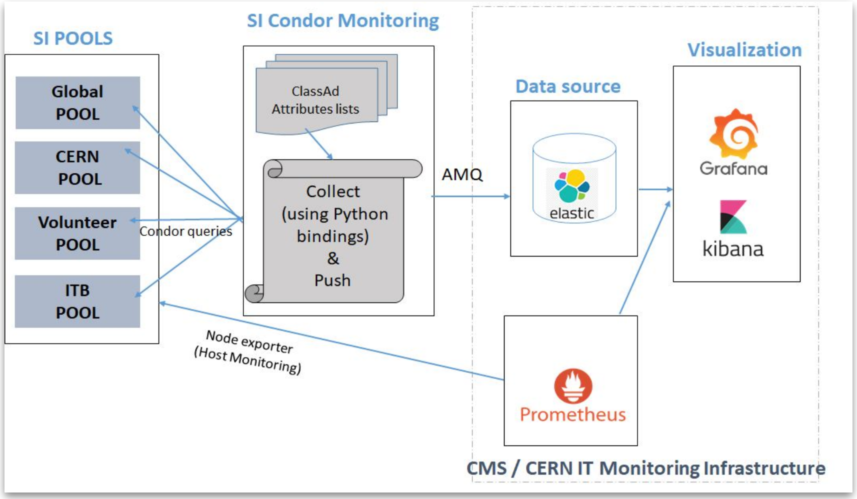
  - **Host Monitoring**
  - **Service Monitoring**
    - HTCondor Pool Monitoring
    - GlideinWMS Monitoring

Monitoring Architecture

SI host monitoring uses Node exporter/ prometheus.

HTCondor Pool Monitoring uses Collect & Push script:

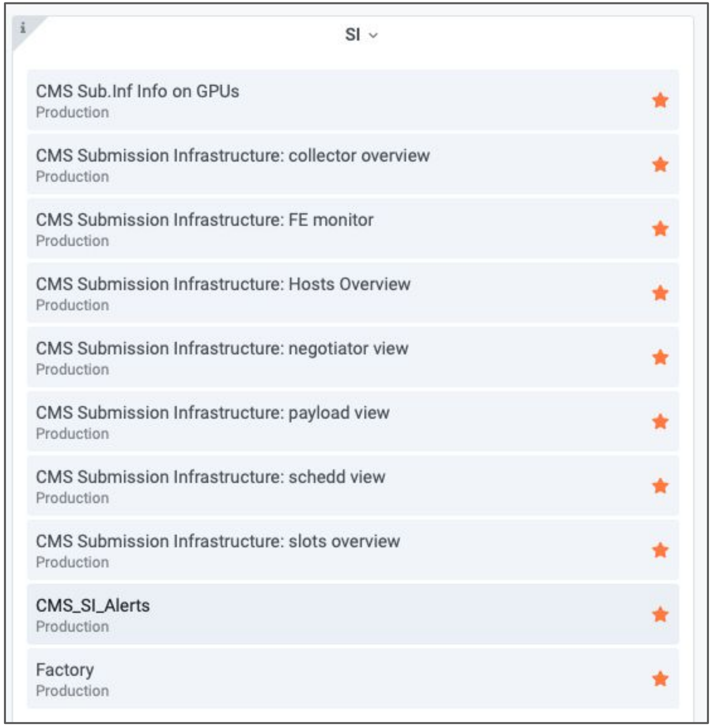
- **Collect** Attributes of interest every 12 mins, using HTCondor python bindings and then convert to JSON using HTCondor native (printJson) method.
- **Push** to ElasticSearch (ES) with same 12 min interval, using stomp, a python library which talk to the AMQ messaging broker of CERN-IT.
    - Collector (14 attributes)
    - Negotiator (20 Attributes)
    - Schedd (16 Attributes)
    - Autocluster (21 Attributes)
    - Startd (~50 Attributes)
- **Average sizes** of the set of documents sent per day are:
  Autocluster: 100 MB
  Collector: 1 MB
  Negotiator: 1.5 MB
  Schedd: 22 MB
  Startd: 25 GB



- **Monitoring queries are always sent to backup collector, to minimize load on primary collector.**

Monitoring

- SI Grafana Monitoring consists 10+ Dashboards
- 100+ different plots.
- Helps us troubleshooting issues, finding sources of inefficiencies and monitor general performance of the infrastructure



Host Monitoring

Basic monitoring of all nodes in our infrastructure

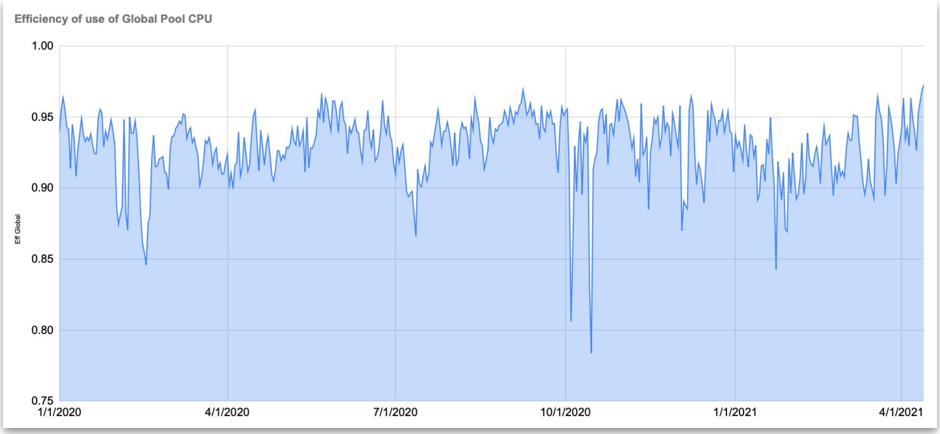
HOST Monitoring
(Prometheus/node exporter)

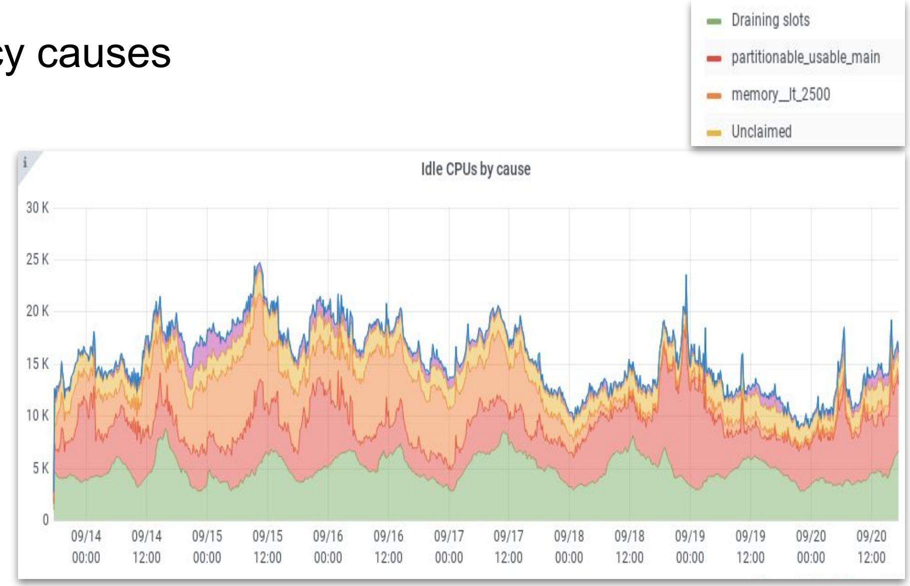
Overall System Health check:
Mainly CPU, Memory , Network connections, Disk I/O, UDP buffers/errors …..

Monitoring pool slots (startds)

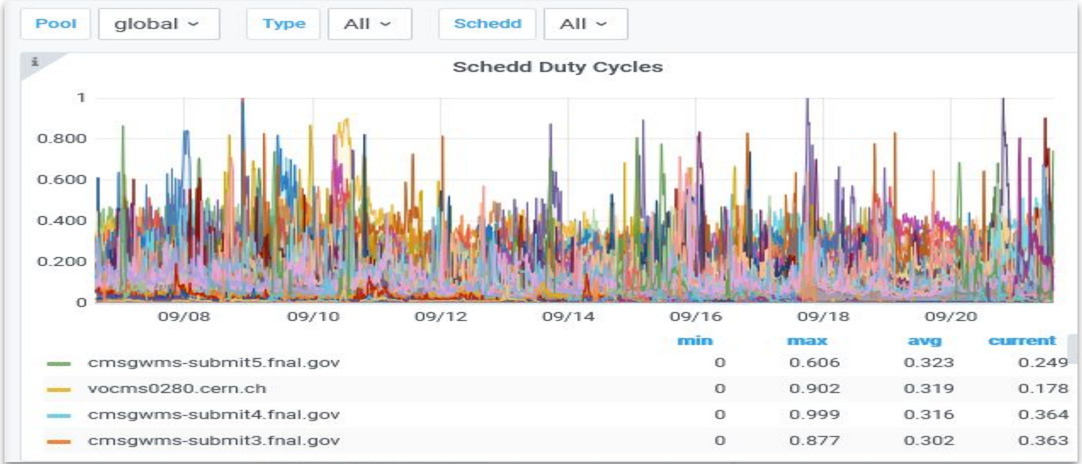
Key to monitor the amount of resources per pool/site/entry and its efficiency of use

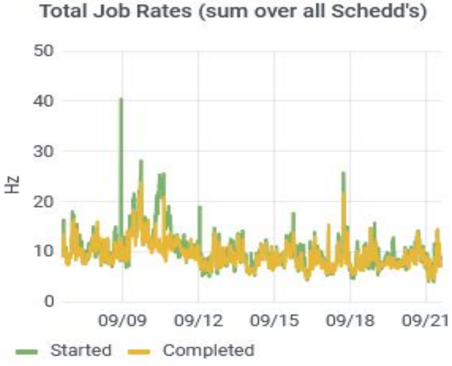
- Efficiency results in the global pool typically in the **90%-95% range** (see e.g. since Jan 2020 over several months)
- Monitoring to **diagnose** slot use inefficiency causes

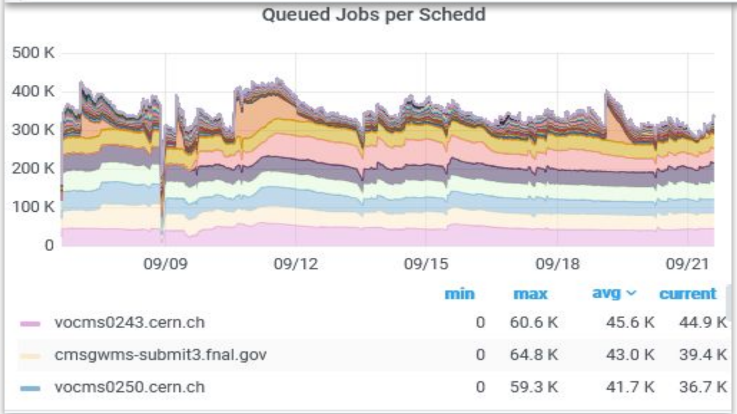


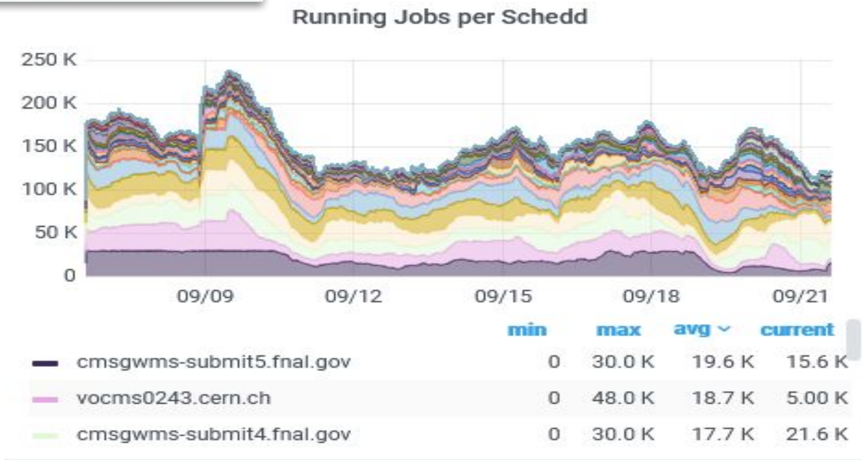


Monitoring schedds and queues







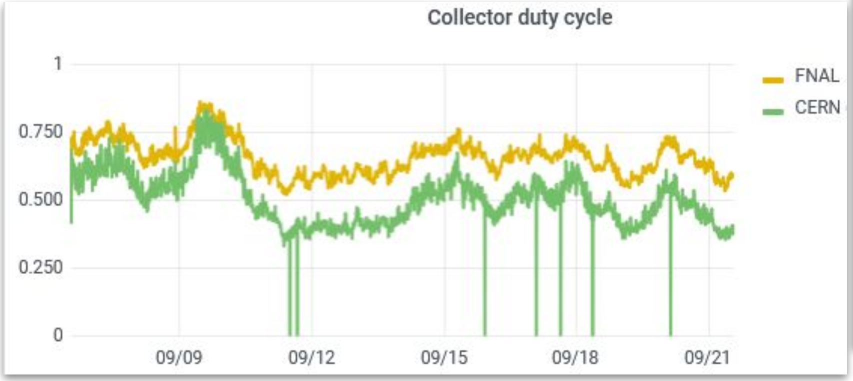


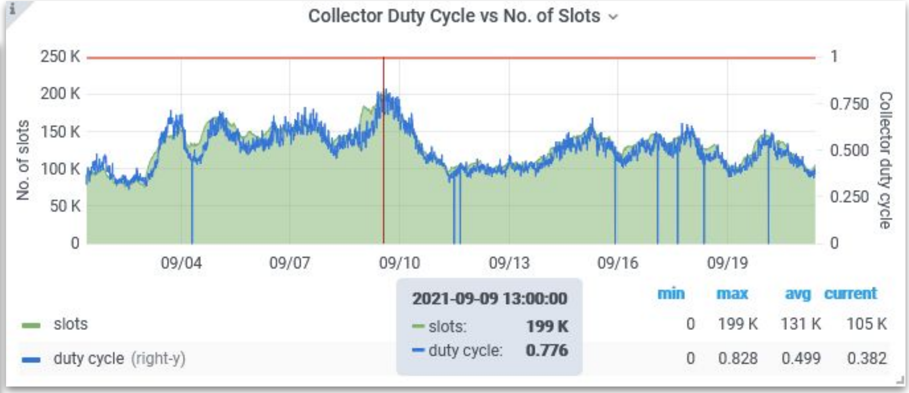
Monitoring collector and negotiator

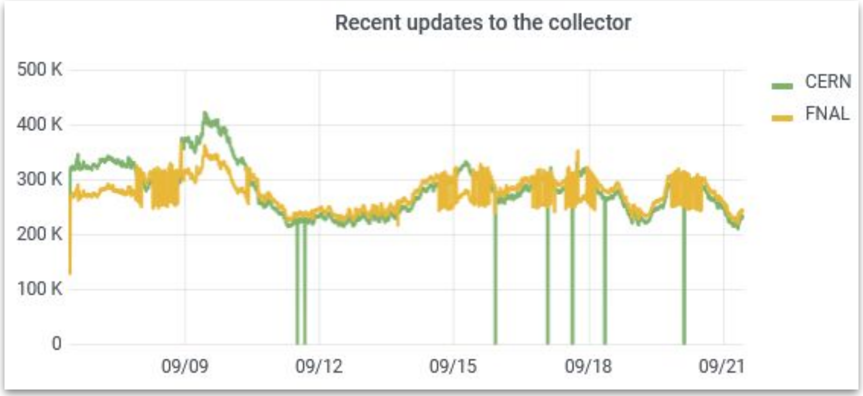
- Scalability and stability of the pool critically depending on collector health status

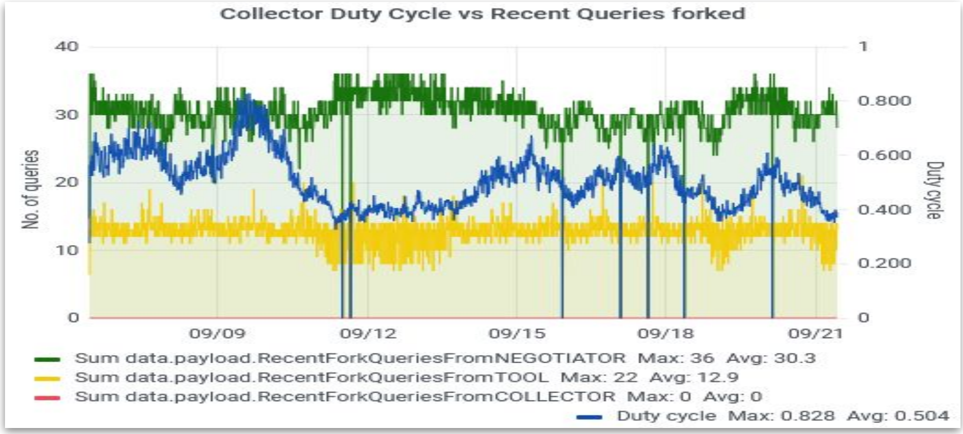
- Efficiency depends on negotiator cycle and timed-out (dropped) schedds

Collector Monitoring



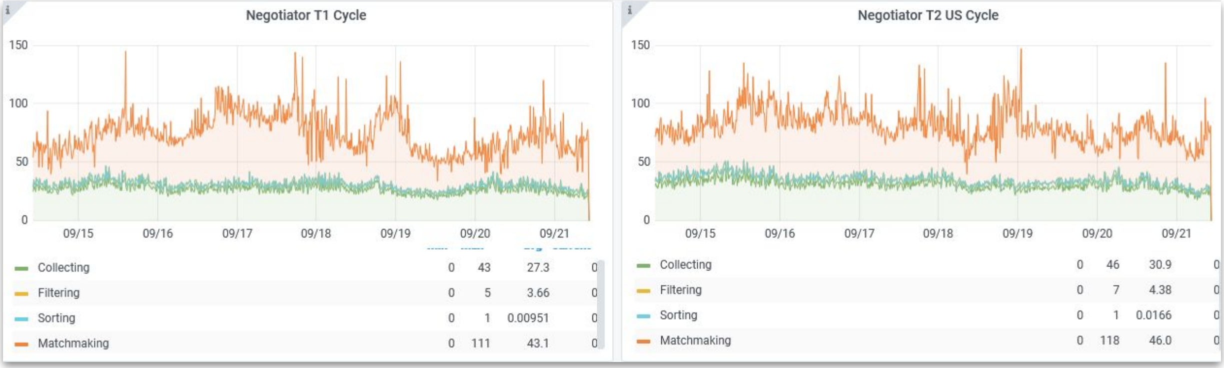






Negotiator Monitoring

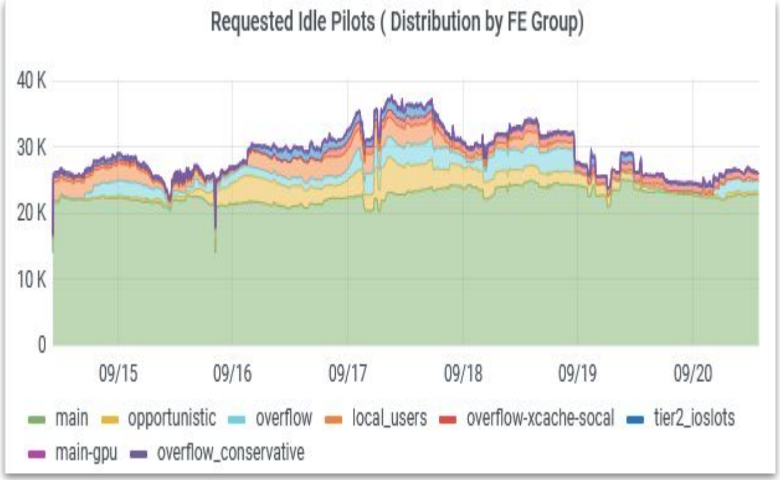
- Negotiation cycle time

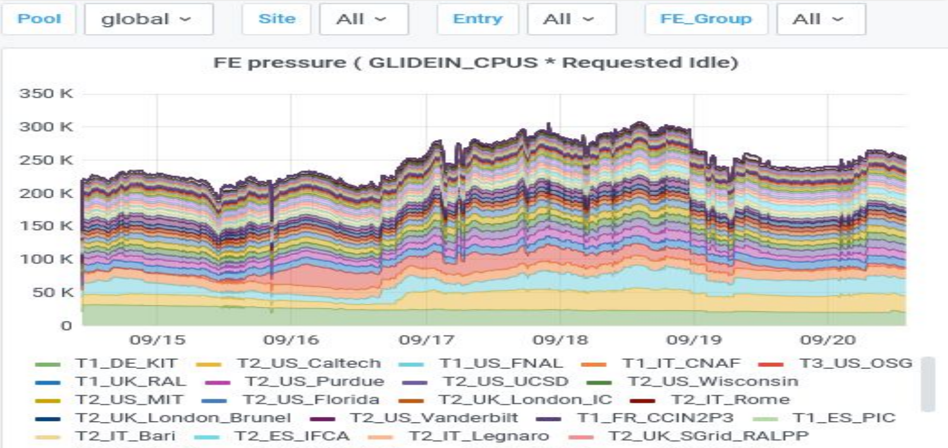


- Matchmaking performance

Monitoring the GlideinWMS FE and Factories

- GlideinWMS service monitoring has also been unified into single visualization tool (Grafana)
- Scripts developed for **GWMS FrontEnd** and **Factories** monitoring, which parses GWMS native XML monitoring pages
- Prepare data in JSON and push to **ElasticSearch** every 12 mins (cron job).





Monitoring the GlideinWMS FE and Factories

Factory Monitoring :

- Number of running, Idle and failed pilots.
- Number of Pilots in Idle in local factory queue vs Remote queue (CE).
- Pilots failing validation/connectivity with pool.
- Percentage of pilots running no user job.
- Detailed Pilots logs monitoring.

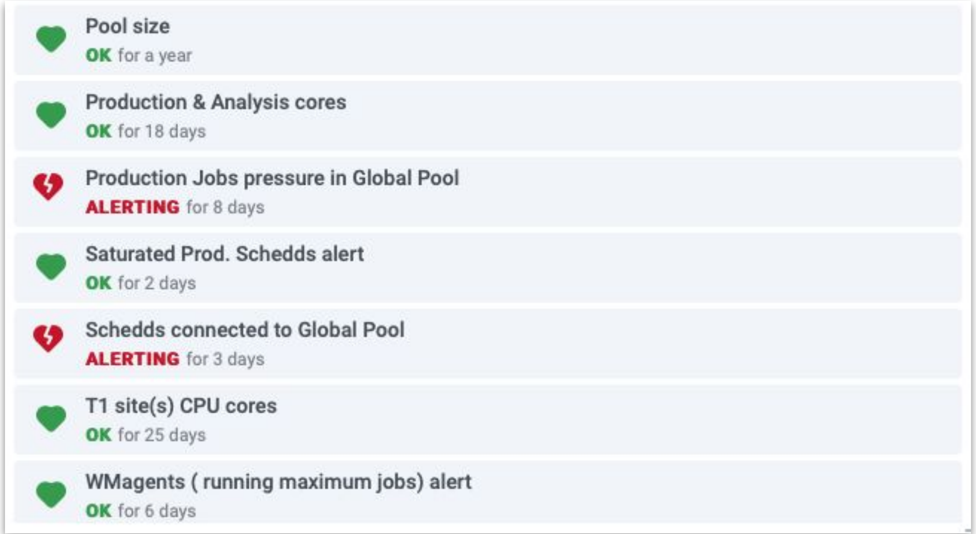
and many other plots helpful for analytics and troubleshooting..

- Few factory monitoring plots

Alerts

Setting up alerts for our pools (I)

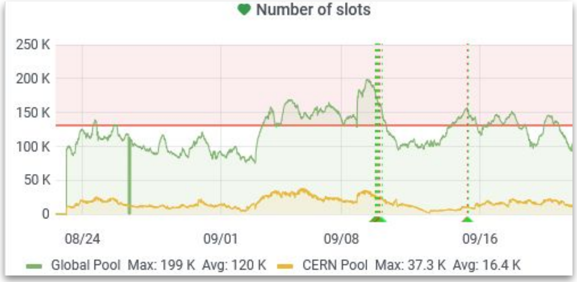
- **Certain metrics have been identified as markers of a healthy pool status**
- *Reasonable* operative thresholds for those metrics allow us to set up alerts
- Grafana email alerts configured
  - Iterative work under continuous refinement



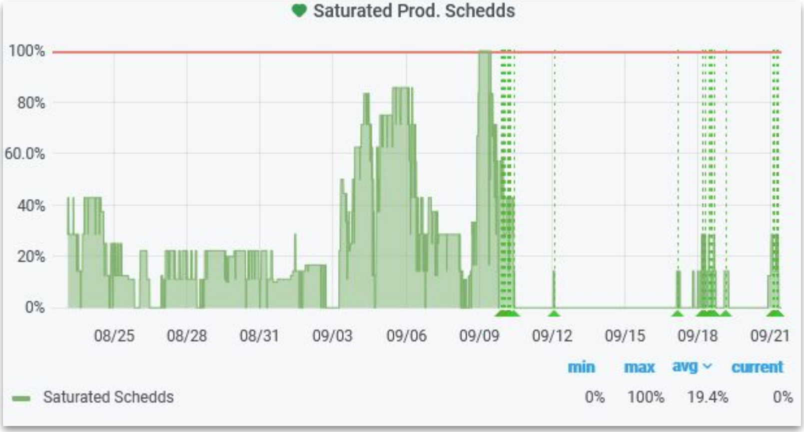
Setting up alerts for our pools (II)

- Some examples

Alert: Production jobs in a queue drop below certain threshold
Danger: reduced workload pressure might indicate issues with the WM layer, resulting in draining and shrinking pool



Alert: active schedds in a pool reach 98% of MAX_JOBS_RUNNING
Danger: Pool will contract as FE will not request new pilots to avoid running them inefficiently



Alert: pool fragmentation inti high number of dynamic slots due to single core jobs "storms"
Danger: stress on the collector and CCB, missing new slots/slot updates, resulting in degraded pool performance

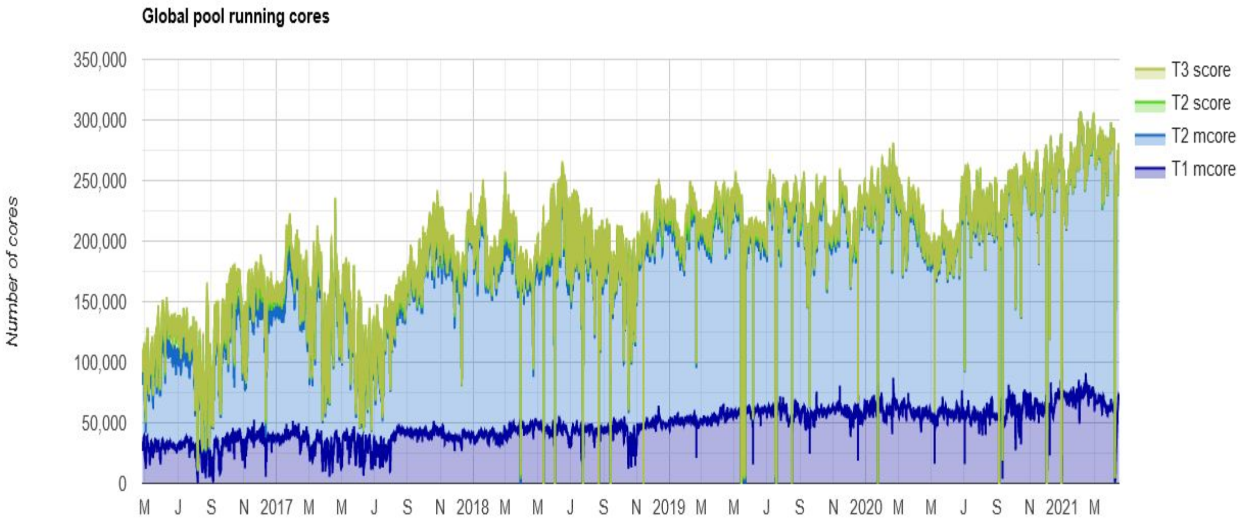
Alternate monitoring and historical data

Alternate/other Monitoring sources

We also have alternate monitoring scripts (non ES), and based on condor CLI queries.

Those were the old ways of monitoring but still useful in multiple ways, for example :

- Functioning as a backup/alternate monitoring[*].
- Can provides Historical data for longer period (~5 Years)[*]
- Single page table showing overall hourly SI summary [**]



Aggregated data: Running cores for last 5 years

[*] https://cms-htcondor-monitor.web.cern.ch/cms-htcondor-monitor/aperezca/HTML/longglobal_pool_size_8640h.html
[**] http://cms-htcondor-monitor.t2.ucsd.edu/letts/production.html

Historical Data retention

To Do: Historical Data in MonIT

- **Motivation**: keep record of the evolution of the Submission Infrastructure in size, complexity, performance, etc.
    - useful e.g. in yearly CMS resource usage accounting reports

- SI monitoring data **retention period is 30 days in Elasticsearch**, (enough for routine operational activities.

- Compressed data is also being archived on hadoop storage for longer period (~1 year or more).

- Need to do data aggregation to keep historical elasticsearch data for 5 to 10 years.

Conclusions

Summary and Conclusions

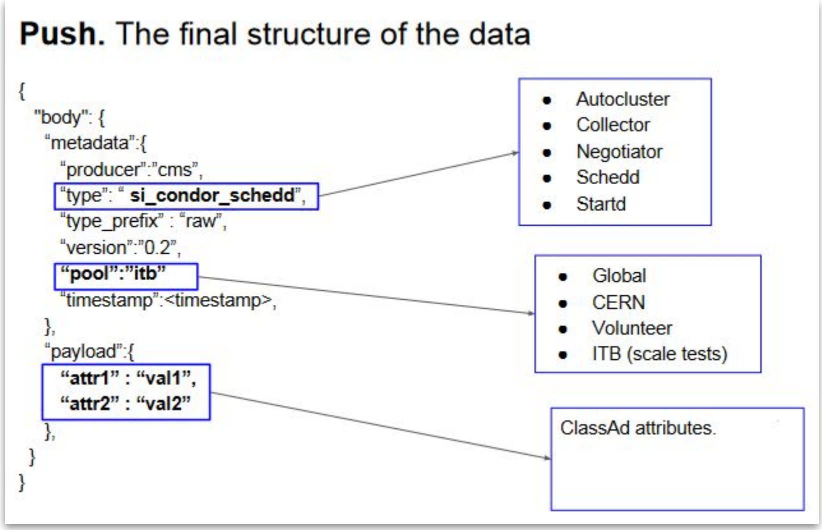
Summary:

- The Submission Infrastructure is a stable and performant piece of CMS Computing, continuously being reviewed, upgraded and expanded
- Automation of maintenance tasks is a must in managing our complex infrastructure
- Continued improvement in SI Monitoring tools required to help us to take proactive and reactive actions timely, keeping the Submission Infrastructure efficient with limited manpower
- Alerts are definitely useful to quickly identify, and possibly correct, dangerous scenarios

We thank the HTCondor development team for the continued support to CMS Submission Infrastructure over the years, a model of excellent partnership!

Extra Slides

Monitoring algorithms



Simple condor query using Python binding

```
import htcondor

collector = htcondor.Collector("collector-hostname.domain")

projection=["Name", "Disk", "Memory"]
condor_ad_type = htcondor.AdTypes.Startd
constraint="Cpus > 0"

#SELECT Name, Disk Memory FROM Startd WHERE Cpus > 0

ads = collector.query(condor_ad_type, constraint, projection)

for ad in ads:
    print ad
```

List of Monitoring Attributes (I)

Collector

Machine
ActiveQueryWorkers
PendingQueries
RecentDaemonCoreDutyCycle
RecentDroppedQueries
RecentForkQueriesFromNEGOTIATOR
RecentForkQueriesFromTOOL
RecentUpdatesLost
RecentUpdatesTotal
SubmitterAds
RecentForkQueriesFromCOLLECTOR
RecentInProcQueriesFromTOOL
DCUdpQueueDepth
CondorVersion

Negotiator

LastNegotiationCycleActiveSubmitterCount0
LastNegotiationCycleMatches0
LastNegotiationCycleNumIdleJobs0
LastNegotiationCycleNumSchedulers0
LastNegotiationCyclePhase1Duration0
LastNegotiationCyclePhase2Duration0
LastNegotiationCyclePhase3Duration0
LastNegotiationCyclePhase4Duration0
LastNegotiationCyclePies0
LastNegotiationCyclePieSpins0
LastNegotiationCycleRejections0
LastNegotiationCycleSchedds OutOfTime0
LastNegotiationCycleTotalSlots0
MonitorSelfCPUUsage
MyCurrentTime
MyType
Name
Machine

Schedd

Autoclusters
CMSGWMS_Type
CurbMatchmaking
MaxJobsRunning
Name
NumOwners
RecentDaemonCoreDutyCycle
RecentJobsCompleted
RecentJobsStarted
RecentJobsSubmitted
RecentResourceRequestsSent
TotalHeldJobs
TotalIdleJobs
TotalRunningJobs
IsWarning
IsCritical

Autocluster

AcctGroup
AutoClusterId
CMS_JobType
CRAB_ReqName
DiskUsage
JobCpus
JobPrio
JobStatus
MATCH_GLIDEIN_CMSSite
MaxCores
MemoryUsage
MinCores
OriginalCpus
RemoteUserCpu
RemoteWallClockTime
RequestCPUs
RequestDisk
RequestMemory
ResidentSetSize
WMAgent_RequestName
WMAgent_SubTaskName

List of Monitoring Attributes (II)

Startd

Activity
ClientMachine
CPUs
DaemonStartTime
DetectedIoslots
DetectedRepackslots
Disk
GLIDECLIENT_Group
GLIDECLIENT_Name
GLIDEIN_ClusterId
GLIDEIN_CMSSite
GLIDEIN_Entry_Name
GLIDEIN_Factory
GLIDEIN_PS_HAS_SINGULARITY
GLIDEIN_Job_Max_Time
GLIDEIN_MAX_Walltime
GLIDEIN_ProcId
GLIDEIN_REQUIRED_OS
GLIDEIN_Schedd
GLIDEIN_ToDie
GLIDEIN_ToRetire
GlobalJobId
Ioslots
Memory
MyCurrentTime
MyType
Repackslots
SlotType
State
TotalIOSlots
TotalRepackSlots
TotalSlotCpus
TotalSlotMemory
CMSSubsiteName
CUDACapability
CUDAClockMhz
CUDAComputeUnits
CUDACoresPerCU
CUDADeviceName
CUDADriverVersion
CUDAECCEnabled
CUDAGlobalMemoryMb
CUDAOpenCLVersion
AssignedGPUs
Machine
TotalMemoryUsage
MemoryUsage
ProportionalSetSizekb
TotalJobRunTime

Abstract

The CMS Submission Infrastructure team manages a set of HTCondor pools to provide the vast amount of computing resources that are required by CMS to perform tasks like data processing, simulation and analysis. A set of tools that enables automation of regular tasks and maintenance of the key components of the infrastructure has been introduced and refined over the years, allowing the successful operation of this infrastructure. In parallel, a complex monitoring system that includes status dashboards and alarms have been developed, enabling this effort to be performed with minimal human intervention. This contribution will describe our technology and implementation choices, how we monitor the performance of our pools in diverse critical dimensions, and how we react to the alarms and thresholds we have configured.