Evolution of the CMS Submission Infrastructure to Support Heterogeneous Resources

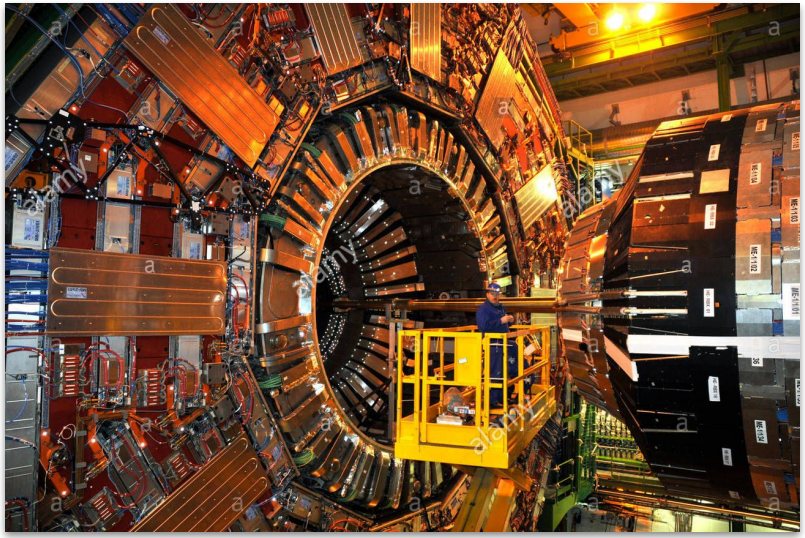
Marco Mascheroni, Antonio Perez-Calero Yzquierdo
On Behalf of the CMS Submission Infrastructure team

HTCondor Workshop Spring 2021

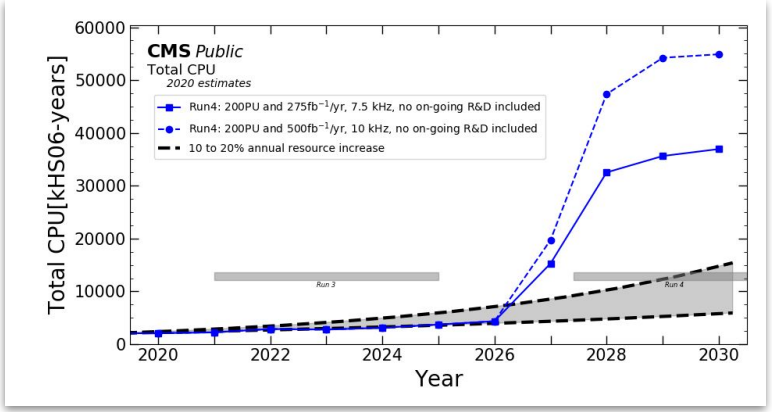
Outline

- The CMS experiment
- The CMS Submission Infrastructure
- CMS interest on GPUs
- Allocating and using GPUs
- Tests on GPUs already available for CMS
- Accounting for GPUs usage
- Conclusions

The CMS experiment at CERN



- **High Energy Physics general-purpose experiment** running **at the LHC** at CERN: proton-proton collisions
- Experimental data is stored, distributed, reconstructed, and analyzed, comparing to simulated data (Monte-Carlo)



- Projections made in 2020 predicted ~4x gap in CPU needs to address resource evolution by the High-Luminosity LHC era (starting in 2027)

[CMS computing public results]

The computing landscape

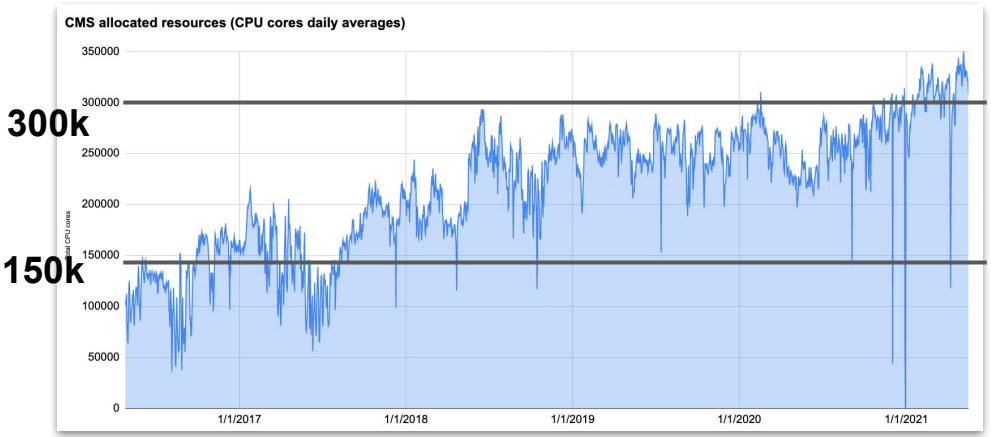
- Data traditionally analyzed using **WLCG Grid resources**
  - Global collaboration of around 170 computing centers
  - Access based on **pledges**
  - **Homogeneous** resources (VO Card)
- Began to exploit HPC as well
  - E.g.: Cori, Stampede, Bridges, Theta, Expanse Frontera, Marconi, BSC, …
  - Access based on **allocations**
  - **Heterogeneous** resources (different architectures, network segregation, **GPUs**, ...)

Within the CMS Experiment, the **Submission Infrastructure (SI)** group organizes HTCondor operations, the technology used to access all of these resources.

The CMS Submission Infrastructure Group

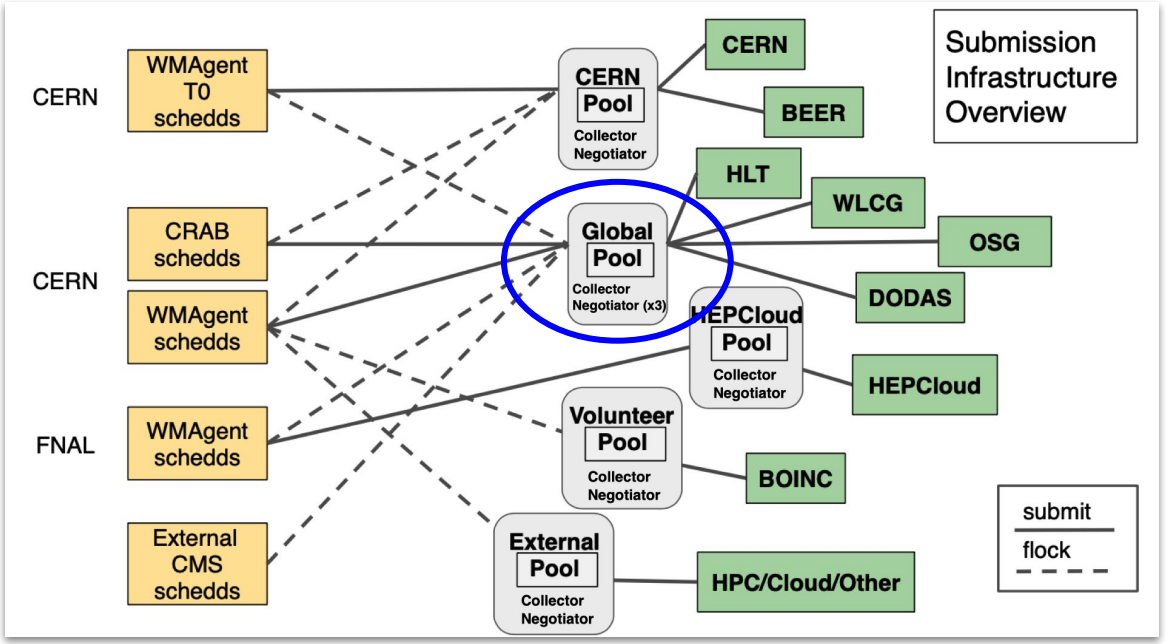
- Part of CMS Offline and Computing in **charge** of:
  - Organizing HTCondor and GlideinWMS operations in CMS, in particular of the Global Pool, an infrastructure where reconstruction, simulation, and analysis of physics data takes place
  - Communicate CMS **priorities to the development teams** of glideinWMS and HTCondor
- In practice:
  - We operate a (vanilla) HTCondor pool peaking at **250k/300k cores distributed over 70 Grid sites, plus non-Grid resources (HLT farm, HPC sites, Cloud)**
  - We regularly hold **meetings with** HTCondor and glideinWMS **developers** where we discuss
    - Current operational limitations
    - Feature requests
    - Future scale requirements

Cores running in the Global Pool in the past 5 years



A Complex Infrastructure

- **The CMS SI model evolved to use federated pools,** with extensive use of **flocking**
- Two sets of workflow managers: CRAB + WMAgent
- **The Global Pool is the biggest** and most important one:
  - **~300k CPU cores**
  - **100k to 150k running jobs**
  - 50+ schedds
  - 3 negotiators
- Redundant infrastructure for HA



- Resources **mainly** acquired with GlideinWMS pilots
- Vacuum-like instantiated: slots (DODAS), BOINC(CMS@Home), opportunistic (HLT)...

Interest for GPUs

- **Availability** of graphics accelerators is **increasing**
  - Not only **HPC**, but also among traditional **WLCG** resources

- Be prepared to **run** substantial part of processing **at HPCs**
  - Shift driven by funding agencies
  - High fraction of the **computing power** in HPCs coming **from GPUs**

CMS Software capabilities

- **CMSSW framework** orchestrates the **data processing**
  - cmsRun is the payload process that ultimately runs on nodes at sites
- Allows to **offload** computation **to GPU accelerators**
  - **In process**: on accelerators present on the the same node cmsRun is being executed. This is the **focus** of **this talk**
  - **External**: connecting to an external process possibly running on another node
- Heterogeneous trigger farm (**HLT**) **featuring** CPUs and **GPUs** for run 3
  - Up to **25%** of the HLT reconstruction code can be **offloaded to GPUs** (source)
  - Offline data processing framework and algorithms are the same used online
- Framework able to dynamically use GPUs **when they are available**, or just use CPUs (and take longer)
  - More on this in a moment!
- NVidia CUDA as language
  - Performance portability libraries are being investigated

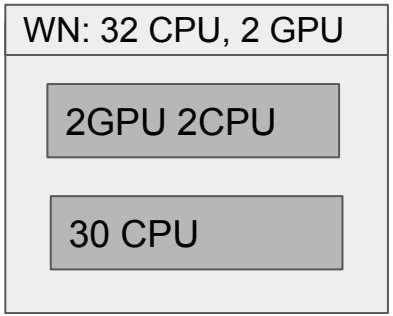
Allocating and Scheduling workloads on GPUs

GPU node allocation: configuration

- Pilot jobs need to be configured
  - Typically we send **8-core partitionable** pilots for CPU sites
  - **Whole node** pilots are also possible
- Need to include GPUs in the slots now

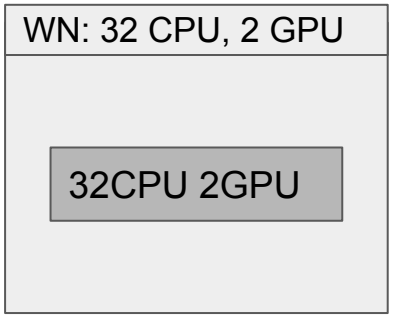
Diverse modes available:

1



Keep GPUs in a dedicated slot with minimum CPU

2



Whole node configuration type

- Currently using a mix depending on site admin preferences
  - Either whole node configuration, or slots with 8 CPUs + 1 GPU
- Flexibility vs efficiency
  - Whole nodes are more flexible but less fragmentation with the 8+1 slots

Matchmaking GPUs in CMS HTCondor Pool

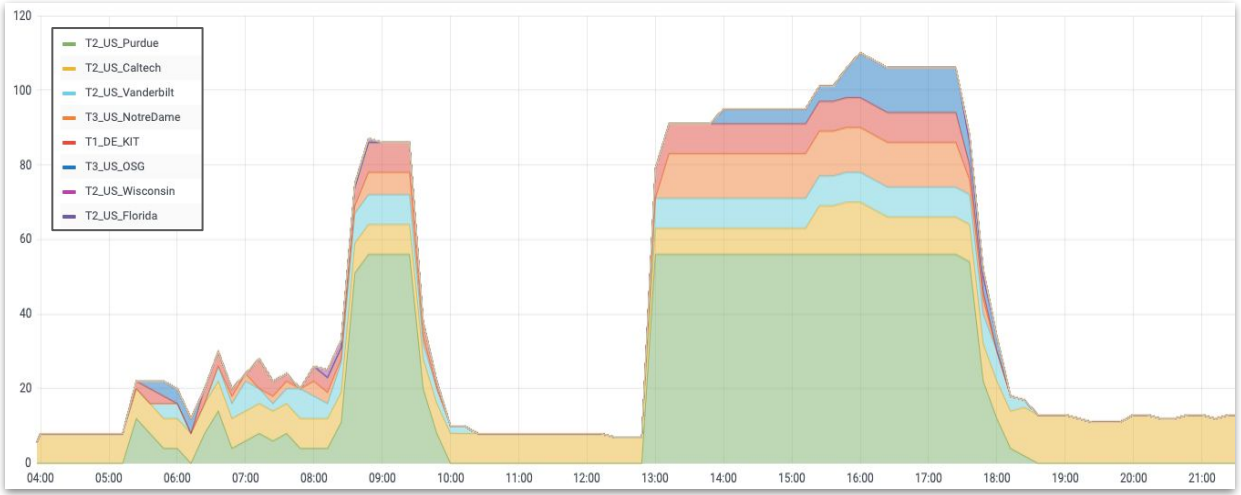
These are the CMS specifications foreseen for GPU-workload matchmaking

- Main switch
  - `request_gpus` to signal the necessity to use a GPU
  - CMS SI was asked to **accomodate** for a use case where payloads **use GPUs “if available”**
- **Mandatory** matching attributes
  - `GPUMemory` (e.g.: 8000Mb, 16000Mb)
  - `CUDACapability` (e.g.: 6.0, 6.5)
  - `CUDACompatibleRuntimes` (e.g.: [11.1,11.2], [12.0,12.5]): list of runtimes provided by running a CMS probing script in the pilot
- **Monitor & Debug** attributes
  - `GPUName`, `CUDARuntimeVersion`, `CUDADriverVersion`, ...

Tests on GPUs (I)

- **Tested matchmaking**: jobs to GPUs pointing to all resources (sites)
- Measure their **performance** by executing a simple script:
  - TensorFlow multiple (10k x 10k), float16, random matrix multiplication
- Record **execution time** as metric correlated to performance
  - A function of GPU model and average for each CMS site

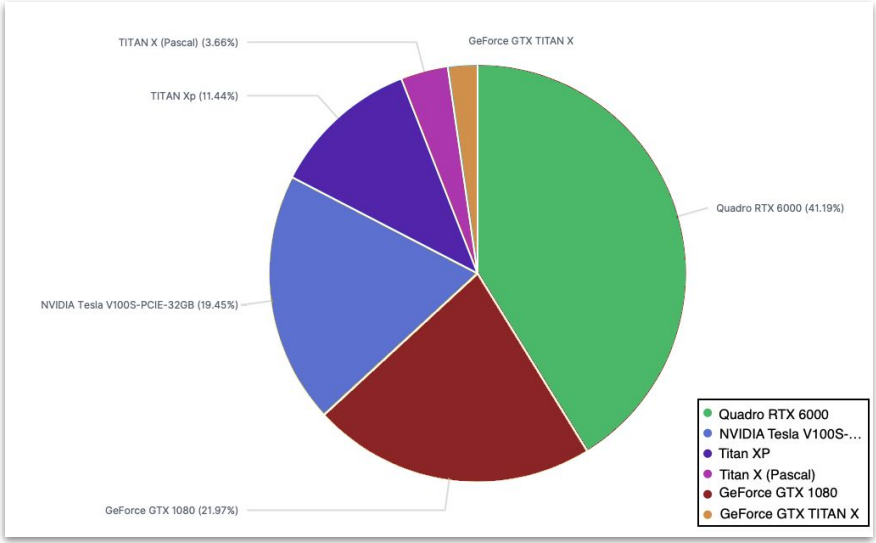
GPUs in use during the tests



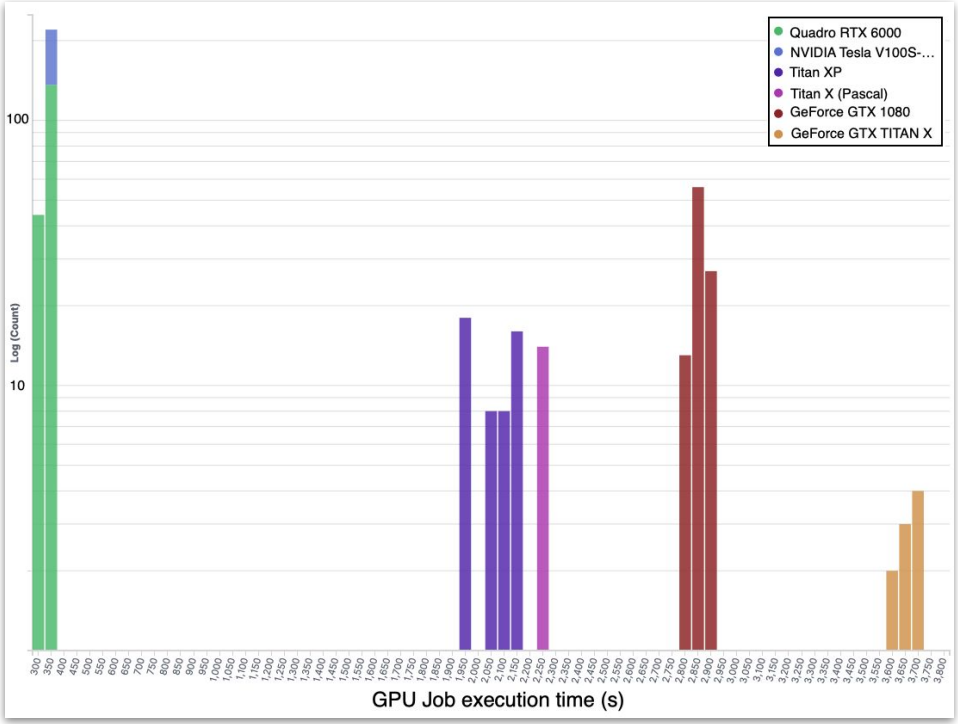
Tests on GPUs (II)

- Execution time of identical payload as a metric of performance, by GPU type

Distribution of test jobs by GPU type



Distribution of test job execution time results



Monitoring GPUs available to CMS

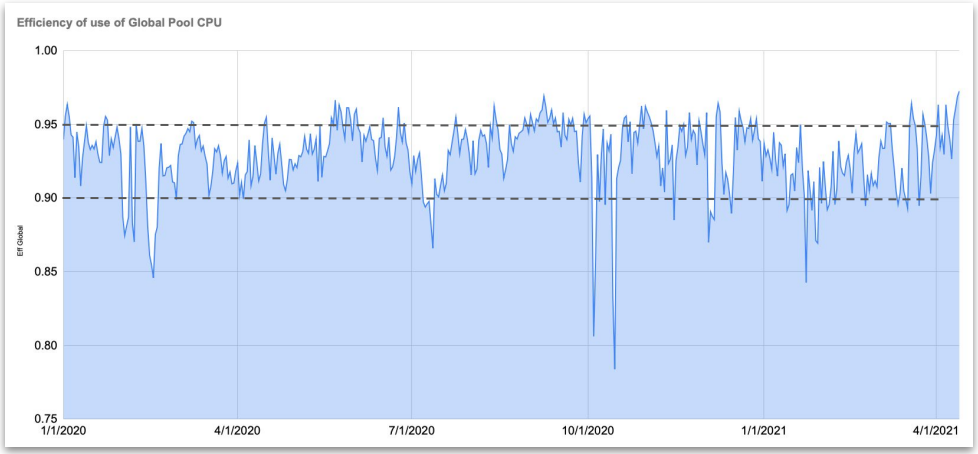
- **Regularly scanning** the Global Pool for **GPU resources** and their properties
  - Submit pilots that run the **HTCondor GPU discovery tool**
- Resources are opportunistic in nature for now
  - sites voluntarily adding GPUs to CMS pool

GPU resources

Site	Entry	CUDACapability	CUDAClockMhz	CUDAComputeUnits	CUDACoresPerCU	CUDADeviceName	CUDADriverVersion	CUDAECCEnabled	CUDAGlobalMemoryMb
T2_US_Caltech	CMSHTPC_T2_US_Caltech_cit2_gpu	5.2	1216	24	128	GeForce GTX TITAN X	11	false	12213
T2_US_Caltech	CMSHTPC_T2_US_Caltech_cit_gpu	5.2	1216	24	128	GeForce GTX TITAN X	11	false	12213
T2_US_Caltech	CMSHTPC_T2_US_Caltech_cit3_gpu	5.2	1216	24	128	GeForce GTX TITAN X	11	false	12213
T3_US_OSG	HCC_US_Omaha_crane_gpu	6.0	1329	56	64	Tesla P100-PCIE-16GB	11	true	16281
T3_US_OSG	HCC_US_Omaha_crane_gpu	6.0	1329	56	64	Tesla P100-PCIE-12GB	11	true	12198
T3_US_OSG	HCC_US_Omaha_crane_gpu	6.1	1582	28	128	GeForce GTX 1080 Ti	11	false	11178
T3_US_OSG	HCC_US_Omaha_crane_gpu	6.1	1709	10	128	GeForce GTX 1060 6...	11	false	6078
T2_US_Vanderbilt	CMS_T2_US_Vanderbilt_ce6_gpu	6.1	1582	30	128	TITAN Xp	10	false	12196
T2_US_Vanderbilt	CMS_T2_US_Vanderbilt_ce5_gpu	6.1	1582	30	128	TITAN Xp	10	false	12196
T2_US_Vanderbilt	CMS_T2_US_Vanderbilt_ce6_gpu	6.1	1531	28	128	TITAN X (Pascal)	10	false	12196
T2_US_Vanderbilt	CMS_T2_US_Vanderbilt_ce5_gpu	6.1	1531	28	128	TITAN X (Pascal)	10	false	12196
T3_US_OSG	HCC_US_Omaha_crane_gpu	7.0	1380	80	64	Tesla V100-PCIE-32GB	11	true	32510
T2_US_Purdue	CMSHTPC_T2_US_Purdue_Hamme...	7.5	1590	40	64	Tesla T4	11	true	15110
T3_US_OSG	HCC_US_Omaha_crane_gpu	7.5	1815	48	64	Quadro RTX 5000	11	false	16125
T3_US_OSG	Glow_US_Syracuse2_condor_gpu	7.5	1620	72	64	Quadro RTX 6000	11	true	22699
T3_US_OSG	HCC_US_Omaha_crane_gpu	7.5	1770	72	64	Quadro RTX 8000	11	false	48601
T3_US_OSG	Glow_US_Syracuse3_condor_gpu	7.5	1815	48	64	Quadro RTX 5000	11	false	16125
T3_US_OSG	Glow_US_Syracuse2_condor_gpu	7.5	1815	48	64	Quadro RTX 5000	11	false	16125
T2_US_Wisconsin	CMSHTPC_T2_US_Wisconsin_cms...	7.5	1590	40	64	Tesla T4	11	true	15110
T2_US_Wisconsin	CMSHTPC_T2_US_Wisconsin_cms...	7.5	1590	40	64	Tesla T4	11	true	15110
T2_US_Wisconsin	CMSHTPC_T2_US_Wisconsin_cms...	7.5	1590	40	64	Tesla T4	11	true	15110

Efficiency in the exploitation of GPU nodes

CPU efficiency in the CMS HTCondor Global Pool: typically high efficiency, we want to keep it that way when moving to heterogeneous resources!



Site policies matter: **preference to use GPU** slots only when GPU workloads are available

Then, saturate the CPU part if possible

But vacate the slot as soon as the GPU part is finished, to enable access for other jobs

- Considering **preemption of the CPU part**? Would be the **first time** we use it in SI

Also, given late-binding, minimize pilots landing on a GPU node when all the GPU jobs are gone

- Plan to keep **removing pilots that are idle** in the site queues **if necessary**, like we do for CPUs
- Common to regular CPU pilots, but more critical for GPUs

Open questions

GPU benchmarking & accounting

- Why **benchmarking**?
  - Pledges definition and resource acquisition
  - A critical element in proper **accounting of resource utilization**

- The **current standard** HEP benchmark: **HS06** (based on 2006 specification, **no GPU**)
  - **HEPScore** as a replacement that **has been developed** by the HEPiX Benchmarking Working Group (source)
  - Inclusion of **GPUs** is an **ongoing development effort** (using event processing throughput as performance metric)

- While the proper procedure for GPU usage accounting is defined, our simple proposal to start accounting GPU usage would be to record all the data related to job execution time, along with GPU model, CUDA libraries, etc.
  - Use “GPU time”, as provided by HTCondor, in addition to `GPUsAverageUsage and GPUsMemoryUsage`

- A **GPU benchmark** will be needed to reasonably predict **scaled execution time of CMS workloads**
  - A key parameter for an efficient matchmaking of jobs to slots
    - Up to now, reasonable assumption of approx. equal run times on all CPU resources
  - The use of **GPU** coprocessors introduce a high degree of heterogeneity in the resources we can access, and their relative throughputs
    - Observed a 10x execution time difference on our very simple test jobs!

Use GPU “if available”

Profit from framework flexibility to dynamically adapt workload to CPU or CPU+GPU environment

However, apparently a simple logic but not so easy to implement!

- **Machine slots** does not simply exist, they **are created through pilot jobs** based un user job requests
  - Do we trigger a GPU pilot if a user job has “`requires_gpu`” `true, false, ifAvailable` ?
- **Matchmaking**: how do you treat those user jobs in terms of shares and priorities compared to other GPU user jobs?
- Still something that **needs to be clarified** internally in CMS by defining clear policies

An example of flexible matchmaking conditions, CMS already has user jobs with variable number of CPUs (**resizable jobs**):

```
RequestCpus = WMCore_ResizeJob ? RequestResizedCpus : OriginalCpus
RequestResizedCpus = (Cpus > MaxCores) ? MaxCores : ((Cpus < MinCores) ? MinCores : Cpus)
```

Focusing on the simple use case for now (`request_gpus` is constant number).

Conclusions

Summary and Conclusions

Summary:

- The Submission Infrastructure is a stable and performant piece of CMS Computing, continuously being reviewed, upgraded and expanded
- The **landscape** of resources for CMS is **evolving** towards higher heterogeneity
  - **GPU resources** already added to the Global Pool.
  - Opportunistic in nature so far, none of it is pledged to CMS up to now
  - Already available for CRAB and CMSConnect users
- **Submission Infrastructure focused on enabling efficient use of GPU** nodes according to CMS preferred policies, which requires customized matchmaking expressions

- The Submission Infrastructure is ready to accept GPU workloads
  - Coordinate with Software and WM teams for realistic workload submission

To Do:

- **Use GPUs "IfAvailable"** remains a problem to be solved
- GPU **benchmarking** and usage **accounting** to be clarified

We thank the HTCondor development team for the continued support to CMS SI, a model of excellent partnership!

Extra Slides

Abstract

The landscape of computing power available for the CMS experiment is already evolving from almost exclusively x86 processors, predominantly deployed at WLCG sites, towards a more diverse mixture of Grid, HPC and Cloud facilities, incorporating a higher fraction of non-CPU components, such as GPUs. The CMS Global Pool is consequently adapting to the heterogeneous resource scenario, aiming at making the new resource types available to CMS. An optimal level of granularity in their description and matchmaking strategy will be essential in order to ensure efficient allocation and matchmaking to CMS workflows. Current uncertainties involve what types of resources will be available in the future, how to prioritize diverse workflows to those diverse types, and how to deal with a diversity of policy preferences by the resource providers. This contribution will describe the present capabilities of the CMS Submission Infrastructure and its critical dependencies on the underlying tools (such as HTCondor and GlideinWMS), along with its necessary evolution towards a full integration and support of heterogeneous resources according to the CMS needs.

References

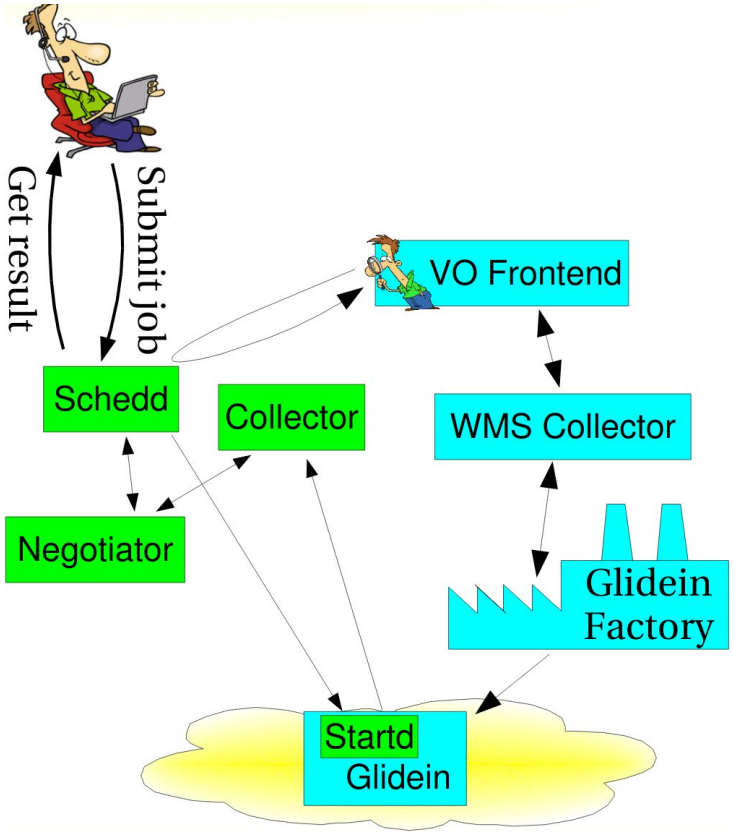
Scalability of the CMS SI:
https://indico.cern.ch/event/948465/contributions/4323961/attachments/2246689/3811125/20210519_SI_vCHEP.pdf

GPU Scheduling in CMS O&C Week:
https://indico.cern.ch/event/1019627/contributions/4292602/attachments/2226284/3771092/Scheduling%20on%20heterogeneous%20architectures.pdf

Online reconstruction on GPUs:
https://indico.hep.caltech.edu/event/883/attachments/648/824/A._Bocci_-_Towards_a_heterogeneous_computing_farm_for_the_CMS_High_Level_Trigger.pdf

Benchmarking:
https://indico.cern.ch/event/948465/contributions/4323674/attachments/2246279/3810655/HEPiX_Benchmarking.pdf

Dynamic HTCondor pool



- GlideinWMS and **pilot jobs**
  - To access **regular Grid resources** (the majority of the CMS resources)
  - The glideinWMS **frontend looks** at the user job **pressure**
  - **Matchmaking** with resources defined in the factory (CEs)
  - **Frontend makes** pilot job submission **requests** to the factory
    - using the WMS HTCondor collector
  - **Factory** then **sends pilot** jobs using condor Grid universe
  - **Pilot job** will then **start** the **HTCondor startd** which will connect to the pool

GPU performance test code

```
import tensorflow as tf
import numpy as np
import time

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

n =10000
iters = 10000
tf.debugging.set_log_device_placement(True)

# Create some tensors and multiply them
start = time.time()

for i in range(iters):
        #print("Iter: "+str(i))
        matrix1 = tf.random.normal([n,n], 0, 1, tf.float16)
        matrix2 = tf.random.normal([n,n], 0, 1, tf.float16)
        prod = tf.matmul(matrix1, matrix2)
        #print(prod.shape)
        #print(" ")

dt = time.time() - start

print("N="+str(iters)+" matrices of type "+str(n)+" multiplied in %f s" % dt)
```