HPC resources integration at CMS

CMS Offline Software and Computing

Abstract

This document identifies minimal set of requirements on HPC based resources in order to run CMS workflows: it is meant to sketch the strategy for an effective exploitation of such machines. It describes possible ways to use HPC machines for any type of workflows, including data reconstruction and Monte Carlo digitisation and reconstruction, besides Monte Carlo generation and simulation, which would represent little value overall if considered alone. This needs the identification of the requirements in order to try to use HPC as any other traditional owned site, and thus dealing with large data input/output, stressing both storage and network.
Introduction

High Performance Computing (HPC) systems are far from standard facilities, but rather custom-built to satisfy use cases largely different from High Energy Physics (HEP) ones. The utilization of these systems by HEP experiments is not trivial: each HPC center is different and this increases the level of complexity from both the integration and operations perspectives.

One clear driver of HPC design is the capability to show the best performance on standard benchmarks, in order to be listed as high as possible in the official HPC ranking\(^1\). Science related use cases range from Lattice QCD, astrophysical simulations, material sciences, and simulations of nuclear systems, as examples.

Such systems show performant node-to-node interconnection, needed for large scale MPI tasks, scarce local scratch disk, and limited capability for data access outside the facility. Recently, HPC systems started to feature accelerators, in order to boost total performance and hence global ranking. Storage systems are optimized for latency and speed, and not for total size.

On the other hand, current HEP High Throughput Computing (HTC) systems are built using different technical needs in mind: note-to-node connectivity is scarcely relevant, large on-node scratch areas are important, and global connectivity is needed in order to access remote datasets. The use of accelerator cards is marginal if not absent.

The HEP workflows are typically data intensive, and systems deploy large storage systems close to the computing farms. Almost 100% of the HTC software stacks are designed and optimized for the x86_64 architecture, definitely the best choice for affordable computing in the last decade.

---

\(^1\) https://www.top500.org/
This highlights two distinct categories of obstacles to the HPC based resources integration in HEP: utilization of accelerators and computing facility integration. The utilization of accelerators is the capability to benefit from accelerator hardware such as GPUs, FPGAs, and in turn translates into an experiment software architecture designed and optimized not only for the x86_64 architecture. The computing facility integration is characterized by challenges such as the amount of available storage, for example the scratch areas on the nodes, the network setup and related policies, as well as security constraints. Often HPC systems are not suitable for I/O intensive workflows: the limited space on the nodes and nearby storage available for user and network seems to be the most limiting factor for CMS. Even outbound connectivity may not be available on the compute nodes due to local security policies. These are largely different types of problems, in terms of people's effort required, type of expertise, as well as impact on experiment operations (e.g. software validation). This document focuses on different aspects of computing facility integration.

Motivations for the document

This document identifies minimal set of requirements on HPC based resources in order to run CMS Workflows. In this respect, it does not intend to address CMS motivations for looking into HPC, but rather to sketch the strategy for an effective exploitation of such machines. For example, describing possible ways to use HPC machines for any type of workflows, including data reconstruction and Monte Carlo digitisation and reconstruction, besides Monte Carlo generation and simulation, which would represent little value overall if considered alone. This needs the identification of the requirements in order to try to use HPC as any other traditional owned site, and thus dealing with large data input/output, stressing both storage and network. Finally the goal should be to make HPC based resources appear as any other CMS site from the CMS central operations point of view.

Starting with a review of relevant inputs based on experience so far, we try to define the minimal set of requirements any HPC system should satisfy in order to allow CMS workflows running as efficiently as possible. This will enable identification of integration limitations and potential obstacles on the human effort, technical, and operational fronts. In addition common areas of development and integration are outlined, identifying a potential preliminary list of items of work.

The next section summarises the motivations of CMS requirements to run jobs. Then a collection of current initiatives and R&D activities about HPC usage is given. Based on the current experience, all the possible integration options for each requirement is summarised. Finally a minimal set of requirements is collected, concluding with a list of open questions.
Requirements to run CMS jobs

The CMS computing environment is based on GlideInWMS and HTCondor. A late binding approach is obtained via the submission of pilot jobs. In addition, CMS supports the so-called vacuum model, where the resource provisioning is not accomplished through the GlideInWMS. Instead, vacuum model worker nodes appear spontaneously “out of the vacuum” at sites, usually either via startup scripts, or initiated by a local batch system not linked to the experiments’ central services.

Regardless of the provisioning model, resources run a pilot, meant as a wrapper around a HTCondor startd, and on a node each pilot job runs in order to check software and services availability, the so-called runtime environment. If all the checks are successful, the pilot job starts pulling physics jobs from the HTCondor queue and executes them. CMS supports both single core and multi-core pilots. CMS runs an HTCondor batch system on top of the globally distributed resources, to address prioritization between different workflows.

Using the GlideInWMS (GWMS) Factory, the first step is the submission. GWMS submits pilot jobs into the resources, the pilot job is started by the batch system at the site, and assesses the runtime environment, communicating back with central services (Central Manager) in order to download the so-called CMS Payloads.

Given the aforementioned approach, a set of constraints comes from the runtime environment. since the runtime includes several key elements:

- **Operating System**: the software stack is based on Redhat Enterprise Linux plus some extra package dependencies. This is currently managed through a container-based solution, namely Singularity, and in general assumes a lightweight virtualization layer is present at the site.

- **Experiment conditions data**: these are managed through local squid servers, which need to be dimensioned properly. In principle this service could be remote with respect to the Worker Nodes (WN) location, although no production sites are running in this configuration.

- **CMS Software (CMSSW)**: this element must be available. The standard solution is to make it available through the CVMFS distributed file system. Technically speaking, CMSSW can be also installed in the local file system; however, given the new release turnaround time (almost daily), it is not a practically viable solution for a general purpose site, perhaps supporting other HEP experiments.

- **Job input data**: this step can follow multiple paths. The first option is that jobs read data locally relying on local storage with enough capacity. Integration with the CMS Data Management system is implied because that is the component responsible for data pre-placement. The second option is to avoid all of the above and to rely instead on streaming data over network. Although both solutions are supported, they are not completely transparent to the experiment. While the former requires a huge storage system and Data Management support, the latter is more lightweight but less performant in term of CPU efficiency. An intermediate solution is to use remote reading, but
mediated by a local cache. Caches are LRU systems, which do not need any remote / local management, since they simply make available the most used files based on real usage patterns.

- **Job output data:** produced data is typically staged to local disk and, in the assumption of a data management system, further moved to the final storage location. Otherwise, remote stageout can be enabled so that jobs write produced output to a remote storage endpoint.

To summarize, in order to run CMS jobs, a resource provider needs to guarantee that

- **Submission is enabled:** the GlideInWMS factory can establish pilots on the computing resources and pilots can call back to HTCondor Central Manager to fetch work
- **Runtime environment is properly setup:** OS and dependencies must be available as well as access to CMS software and conditions.
- **Data management** is enabled for high I/O workflows. Access to input data as well as the ability to write output data must be granted, either locally (preferred solution to guarantee peak CPU efficiency) or remotely

**HPC challenges and identified scenarios**

In this chapter all possible scenarios are defined which are foreseen for each one of the categories of requirements identified above. Two macro scenarios are defined: HPCs where worker nodes have outbound connectivity and where they do not. A code is used to label the scenarios:

- **P** for the preferred, often standard, solution
- **C** for a change required in the standard configuration which is viable with some effort from central CMS operations and/or local site support
- **I** for signalling a blocking issue

**Availability of Internet on Worker Nodes**

If worker nodes (WNs) have internet access, pilots can fetch payloads from the HTCondor queue so the issue to solve is how to get pilots into the WNs.

**Submission**

Depending on the presence of a Computing Element (CE - an edge service) three possibilities are available:

- **P** there is a CE: nothing needed apart from central configuration changes
- **C** there is no CE. The only current tested solution on a HPC system is BOSCO. BOSCO is a technology to allow HTCondor to submit jobs directly into a remote batch cluster by using an ssh tunnel to a head node and some wrapper code on that head node. BOSCO is integrated into the GlideInWMS system
● C there is no CE but a vacuum solution can be exploited. Pilot jobs are launched internally by the HPC center, either via Vac/Vcycle or DODAS.
● C there is no CE but OSG supports “hosted CE”. In this case, OSG hosts the CE, and submits into the site via ssh. This allows VOs like CMS to always have a CE available to them, but there are implicit needs of communication between the CE and the HPC site.

Runtime Environment

If one focuses on operating system and required dependencies, the first topic to consider is containerisation, more specifically Singularity:

● P Singularity is available: no need to change anything.
● C Singularity is not available: any other container technology could work for example Docker, Shifter or udocker. It is clear that CMS would have to invest some effort for building and adapting the software for those products.
● C no "light virtualization" is available but the base system is CentOS7 compatible: CMS could run natively. This requires local sysadmins to allow for the installation of local packages on all the compute nodes.
● I no "light virtualization" is available and a Linux OS very different from CentOS7 is in place: the effort for running CMS workflows might be simply too high.

One additional detail with regards to the Runtime environment is specific to squid proxy access conditions. Some possibilities are:

● P Squids run locally in the LAN of the HPC center, correctly dimensioned with respect to the number of computing resources: no work to be done.
● C Squids can’t run locally. This scenario is still acceptable because CMS can also use squid proxies at remote CMS sites if necessary. However, also depending on the workflow, an efficiency loss can be foreseen.

Finally, the access to CMS software and resource specific settings is to be considered. Software is delivered through CVMFS, therefore:

● P CVMFS is mounted on computing nodes: no difference with respect to usual CMS sites.
● C CVMFS is not provided: both software and settings must be accessed on a local file system. This requires providing a much more complete container image. The risk is that the resulting image becomes prohibitively large. Creating workflow specific images could mitigate the size issue at the cost of additional work on the CMS side.

Data Management

Data management could be completely avoided for diskless HPC systems relying on remote I/O, however it has been already shown how workflow performances in this scenario is highly dependent on specific circumstances such as network connectivity as well as the scale of the operations. With this proviso, possible CMS supported cases are:
• **P** HPC storage can be remotely accessible and provide enough quota: the ideal scenario. CMS treats it as a Grid SE, exploiting data pre-placement, and jobs can stageout locally or remotely.

• **C** HPC has storage but it is not remotely accessible: a service for moving data is needed and this would be an edge service. A Cache/proxy (e.g. XCache) can be used. Stability in production and at scale would need to be proven. If a delay-tolerant networking were made available, it might cover well enough this case.

• **C** HPC has no storage: the solution could be the CMS “Any Data, Anytime, Anywhere” approach and remote stageout. Depending on network connectivity and scale of operations this might lead to a degradation of CPU efficiency and success rate. Not requiring storage at the HPC centre has advantages too: simpler workflows and data management, less infrastructure to maintain and keep working, stateless workflow management.

**Integration into larger CMS Computing Infrastructure**

The ability to run HTCondor pilots at HPC centers doesn’t mean that CMS can run workflows at the site. There are larger integration issues to be considered. CMS can setup the HPC center as a new site and access its resources directly through the HTCondor but that might not be the optimal way to proceed. The HPC might have restrictions or preferences in working together with a CMS site, either for technical or administrative reasons: in this case CMS can treat the HPC as an extension of that CMS site. Job routing then can either be automatic, where the HPC resources just show up as extra resources on the CMS site, or rule based, where the HPC resources can take jobs sent to the CMS site but there are rules in place to decide which jobs are forwarded to the HPC. This latter approach is what the HEPCloud project at Fermilab favours.

Another example of integration into larger CMS infrastructures is what INFN is experimenting with CINECA. The followed approach there is to have a dedicated network connection to the CNAF Tier1 which also allow to access the CNAF storage as local at CINECA.

**Worker nodes without Internet access**

*Under this condition CMS is currently not able to provide a technical solution to use the resources at all.*

The major obstacle derives from HTCondor, because worker nodes need to communicate externally to register the allocated resources to the central manager of the HTCondor pool, in order to be matched to requests, i.e. the payload jobs. This communication usually occurs via network. The absence of network access would therefore make these resources unusable, or at least unsuitable for the late binding paradigm that CMS employs.

However, R&D activities led by the HTCondor developers and the PIC teams are ongoing to resolve this issue. An approach is being adopted which is based on the existence of an external access point to the storage file system shared by the HPC worker nodes. The communication between the HTCondor pool's central manager and the daemon running on the node would
therefore proceed via file exchange. This is a promising approach, although still in the prototype phase. All other requirements could have a solution although it would be mandatory for HPC to allow running of at least few edge services such as Squids and XRootD proxy. From a technical perspective and deployment model this could mean just a single node. Finally, some kind of containerisation layer need to be supported as well for the rest of runtime environment requirements.

Appendix

Minimal set of requirements

In the assumption of outbound connectivity on worker nodes, we can realistically distinguish two sets of requirements depending on workflows taking the level of I/O as discriminant. This has already demonstrated by a NERSC integration activity which took place in CMS.

Low I/O workflows
In this case the requirements would be:
1. Submission: ssh in to head node
2. Runtime:
   a. Singularity (or any other lightweight virtualisation at some cost for CMS)
   b. Remote Squid servers
   c. Access to CMS condition data from remote squids
3. Data Management: remote data access and remote stageout

Edge services in this case are not needed.

High I/O workflows
1. Submission: ssh in to head node
2. Runtime:
   a. Singularity (or any other lightweight virtualisation at some cost for CMS)
   b. Local Squid servers
3. Data Management: local storage and data managed centrally by CMS
   a. A Xrootd proxy cache would be enough

Edge services are required in such scenario. Minimally this could be one single edge machine embedding all the services
<table>
<thead>
<tr>
<th>Category</th>
<th>Explanation</th>
<th>CMS standard solution</th>
<th>CMS preferred solution for HPC</th>
<th>CMS fallback workable solution (full utilizability)</th>
<th>CMS fallback solution (for a fraction of workflows)</th>
<th>CMS no-go scenario</th>
<th>Possible CMS devels to solve the no-go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Base system architecture</td>
<td>x86_64</td>
<td>x86_64</td>
<td>x86_64 + accelerators (with partial utilization)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Currently, OpenPower, ARM, … they could be used but at the price of physics validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory per Thread/core</td>
<td>Memory available to each thread / process</td>
<td>2 GB/Thread</td>
<td>2 GB/Thread</td>
<td>Down to 0.5 GB/thread needs heavy multithreading, at the expenses of CPU efficiency</td>
<td>GEN and SIM workflows need less than 2 GB/Thread (0.5GB/Thread would be a limit) in order to run efficiently</td>
<td>Less than 0.5 GB/thread</td>
<td>QEMU? Recompiling + physics validation?</td>
</tr>
<tr>
<td>I/O</td>
<td>I/O demand per process</td>
<td>5 MB/s/core</td>
<td>5MB/s/core</td>
<td></td>
<td>GEN and SIM workflows are mostly CPU bound still ok with 0.1 MB/s/core</td>
<td>Less than 0.1 MB/s/core</td>
<td></td>
</tr>
<tr>
<td>Local Scratch space</td>
<td>Local space per production job</td>
<td>20 GB/Thread</td>
<td>20 GB/thread local</td>
<td>Less than 20 GB/thread ok if a shared high performance FS is available on all the machines Large multithreading lowers 20 GB/thread requirement to ~ 10</td>
<td>Some CMS workflows run for hours without creating huge local disk areas (GEN, SIM)</td>
<td>No sizeable local space and no shared usable FS</td>
<td></td>
</tr>
<tr>
<td>Outgoing networking</td>
<td>Needed on WNs in order to access remote data, conditions, and to speak to the CMS Global Pool</td>
<td>Full outgoing connectivity</td>
<td>Full outgoing connectivity</td>
<td>Connectivity to only a subset of the IP ranges (for example, to CERN, and to a close xrootd proxy cache) And to everywhere we have condor services?</td>
<td>NAT with a very limited bandwidth via an edge service</td>
<td>No outgoing connectivity from the compute nodes and no NAT available</td>
<td>Edge service running Harvester or HTCondor? Prepare a single container to be deployed at the edge and doing: NAT for Condor, Squid, Xroot proxy cache, …?</td>
</tr>
<tr>
<td>Computing Element</td>
<td>Launch local batch pilots jobs</td>
<td>Present locally</td>
<td>Present locally</td>
<td>It can be not local, if a proper network</td>
<td>It can be substituted by</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Local batch queue</strong></td>
<td><strong>Distributes the pilots to the local compute nodes, upon submission via the CE</strong></td>
<td><strong>Any supported (HTCondor, LSF, SGE, Slurm, …)</strong></td>
<td><strong>Autostart from /etc/rc.local is always possible. It gets very close to the vacuum model</strong></td>
<td><strong>Harvester can help packing jobs into multinode jobs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td><strong>The base Linux system</strong></td>
<td><strong>A Linux derivative of CentOS7</strong></td>
<td><strong>If singularity is present, “almost anything” new enough (64 bit)</strong></td>
<td><strong>Virtualization is an umbrella solution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>A Linux derivative of CentOS7</strong></td>
<td><strong>A single CVMFS server with NFS export should work.</strong></td>
<td><strong>Having containers containing full CMSSW releases is possible, but only a few releases. It means a site could support just a specific workflow. In principle, we could support also “by hand” installation on a local shared area, but again it would support just specific workflows</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Virtualization</strong></td>
<td><strong>Used to ship middleware and specific packages w/o the intervention of local sysadmins</strong></td>
<td><strong>Singularity</strong></td>
<td><strong>We can certainly use full virtualization (OpenStack, etc), but they need someone to launch them.</strong></td>
<td><strong>No virtualization and a strange operating system</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Singularity requested by CMS from the start of 2018</strong></td>
<td><strong>Available on every compute node, served via a local squid</strong></td>
<td><strong>Docker should also be possible; with some effort in principle also udocker</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CVMFS</strong></td>
<td><strong>Used to distribute SW and needed middleware (grid etc)</strong></td>
<td><strong>Available on every compute node, served via a local squid</strong></td>
<td><strong>A single CVMFS server with NFS export should work.</strong></td>
<td><strong>No CVMFS, no CVMFS in NFS mode, no virtualization , no large local shared area</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Available on every compute node, served via a local squid</strong></td>
<td><strong>A single CVMFS server with NFS export should work.</strong></td>
<td><strong>Having containers containing full CMSSW releases is possible, but only a few releases. It means a site could support just a specific workflow. In principle, we could support also “by hand” installation on a local shared area, but again it would support just specific workflows</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Length of jobs (on the local batch system)</strong></td>
<td><strong>Time a slot can be held by CMS processes</strong></td>
<td><strong>48 hours</strong></td>
<td><strong>48 hours (or more)</strong></td>
<td><strong>Slots available for less than 1 hour</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>We can adapt to smaller length, but it needs specific jobs in principle. In practice if the tuning is 8 h as of now, even 24 should be</strong></td>
<td><strong>48 hours</strong></td>
<td><strong>If the slots need to be very short (say 1 h), we can send specific workflows, but the overall</strong></td>
<td><strong>Event service</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Under GWMS factory control, DMZ is set; A remote example is BOSCO. E.g. OSG hosted CE solution. VACUUM pilot creations, which is problematically not aware of CMS job pressure.
<table>
<thead>
<tr>
<th>Access control / local users</th>
<th>Local users (uid/gid) which are mapped to our workflows</th>
<th>One for the pilot, one for singularity. Add to this 1-2 for SAM and debugging. The pilot will launch processes taking credentials from outside (proxies etc)</th>
<th>One for the pilot, one for singularity. Add to this 1-2 for SAM and debugging</th>
<th>We can live with a single user (the pilot one), most probably</th>
<th>Sites which do not want to accept external users at all</th>
</tr>
</thead>
</table>

good enough. Just, the CPU efficiency can go down due to the inability to fill the latest hours of a slot. Throughput will not be stellar.