Dynamic provisioning of a HEP computing infrastructure on a shared hybrid HPC system

2016 J. Phys.: Conf. Ser. 762 012012

(http://iopscience.iop.org/1742-6596/762/1/012012)

Dynamic provisioning of a HEP computing infrastructure on a shared hybrid HPC system

Konrad Meier[2], Georg Fleig[1], Thomas Hauth[1], Michael Janczyk[2], Günter Quast[1], Dirk von Suchodoletz[2], Bernd Wiebelt[2]

[1]Karlsruhe Institute of Technology, Institut für Experimentelle Kernphysik,
Wolfgang-Gaede-Str. 1, 76131 Karlsruhe, Germany
[2]Albert-Ludwigs-Universität Freiburg, Professur für Kommunikationssysteme,
Hermann-Herder-Str. 10, 79104 Freiburg im Breisgau, Germany

E-mail: konrad.meier@rz.uni-freiburg.de

Abstract

Experiments in high-energy physics (HEP) rely on elaborate hardware, software and computing systems to sustain the high data rates necessary to study rare physics processes. The Institut fr Experimentelle Kernphysik (EKP) at KIT is a member of the CMS and Belle II experiments, located at the LHC and the Super-KEKB accelerators, respectively. These detectors share the requirement, that enormous amounts of measurement data must be processed and analyzed and a comparable amount of simulated events is required to compare experimental results with theoretical predictions.

Classical HEP computing centers are dedicated sites which support multiple experiments and have the required software pre-installed. Nowadays, funding agencies encourage research groups to participate in shared HPC cluster models, where scientist from different domains use the same hardware to increase synergies. This shared usage proves to be challenging for HEP groups, due to their specialized software setup which includes a custom OS (often Scientific Linux), libraries and applications.

To overcome this hurdle, the EKP and data center team of the University of Freiburg have developed a system to enable the HEP use case on a shared HPC cluster. To achieve this, an OpenStack-based virtualization layer is installed on top of a bare-metal cluster. While other user groups can run their batch jobs via the Moab workload manager directly on bare-metal, HEP users can request virtual machines with a specialized machine image which contains a dedicated operating system and software stack. In contrast to similar installations, in this hybrid setup, no static partitioning of the cluster into a physical and virtualized segment is required.

As a unique feature, the placement of the virtual machine on the cluster nodes is scheduled by Moab and the job lifetime is coupled to the lifetime of the virtual machine. This allows for a seamless integration with the jobs sent by other user groups and honors the fairshare policies of the cluster. The developed thin integration layer between OpenStack and Moab can be adapted to other batch servers and virtualization systems, making the concept also applicable for other cluster operators.

This contribution will report on the concept and implementation of an OpenStack-virtualized cluster used for HEP workflows. While the full cluster will be installed in spring 2016, a test-bed setup with 800 cores has been used to study the overall system performance and dedicated HEP jobs were run in a virtualized environment over many weeks. Furthermore, the dynamic integration of the virtualized worker nodes, depending on the workload at the institute's computing system, will be described.

1. Introduction

Current and upcoming experiments in High Energy Physics (HEP) require large amounts of processing and storage capacity to handle the recorded data and to provide sufficient simulation and analysis capabilities to their scientists. The Institute of Experimental Particle Physics (EKP) is actively involved in the Compact Muon Solenoid (CMS) experiment at the LHC collider in Geneva (Switzerland) and the Belle II experiment at the Super-KEKB collider in Tsukuba (Japan). The CMS experiment resumed operation, after improvements to the LHC accelerator in 2014 with an increased center-of-mass-energy of $\sqrt{s} = 13\,\text{TeV}$, more concurrent collisions and an increase in factor three of the stored measurement data. These factors make the event reconstruction more challenging, and more computing resources are required to process and analyze the recorded data in a timely fashion. The Belle II particle detector is being built at the moment (02/2016) and is expected to start taking data in 2018. The factor 40 increase in luminosity, compared to the predecessor experiment Belle, also poses a challenge to the available computing resources. In conjunction with the recorded collision data, both experiments require an equal amount of Monte-Carlo simulation events to compare the observations with theoretical models.

The classical model, used with great success during the last decades, is to provide these computing resources via data centers dedicated to HEP computing and running a HEP-specific software stack. Due the independent nature of HEP computing workloads, they can be easily partitioned in many small jobs and workflows can be evenly distributed across different data centers and there is no need for a concurrent execution as no inter-process communication (IPC) is required. Therefore, HEP jobs can be considered to belong to the High-throughput Computing (HTC) class, which describes a loosely coupled computing workflow. Logically, dedicated HEP data centers are tailored to suit the HTC workloads and have batch systems which can manage small jobs efficiently.

In contrast, many other user groups in academia require computing clusters that efficiently serve the High-performance Computing (HPC) class of workloads, which can be characterized as tightly-coupled processing across as significant part of the cluster, which needs low latency interconnection between nodes.

Furthermore, the software environment typically used on HPC clusters is very different from the typical HEP experiment configuration. HPC-focused data-centers use an operating system best suitable to support the installed hardware, for example the IPC components. The cluster operator also provides compilers, software libraries and even whole applications suites for its users. In contrast, most HEP experiments require a specific operating system and centrally provide the experiment software, including compilers and libraries, which have to be used on all sites in order to guarantee the software portability and consistency of the results across all data centers. Furthermore, HEP experiment software is updated in regular intervals and new software releases must be available in a matter of hours on every site. This release mode cannot be properly supported if software must be installed by local experts.

These conditions make it challenging to employ classical HPC-based centers to fulfill the computation needs of HEP experiments. Yet, HPC data centers can provide a significant addition to the computing budget of HEP experiments and their usage is therefore highly desired.

In the following, a new approach to integrating the HEP software setup and workflows, based on virtualization technology, will be outlined and its implementation in a shared cluster system will be described.

2. Hybrid HPC Setup with OpenStack

To allow classic HPC computing (bare metal) and virtual machines on the same cluster the HPC operating modell is extended by a virtualization layer. In order to run virtual machines on a compute node, a virtualization hypervisor is installed on every compute node. Users that require a HPC system for computation can use the middle layer with direct access to hardware to run computation jobs. Users that require a special software environment and do not need direct access to hardware can use the top layer that provides Virtual Research Environments.

To manage virtual machines (network, images, life-cycle) OpenStack [1] is used as a virtualization management framework. OpenStack is chosen for its modular structure. Components required for the cluster virtualization, such as network, image handling or a web interface, can be used while others can be ignored without interference with the overall system. This allows to build a virtualization environment for the cluster that meets the specific requirements.

The challenge in this setup is to interconnect the virtualization environment and the classic HPC environment to allow running classic bare metal compute jobs and VMs on the same cluster at the same time without interference. In order to achieve this, it is necessary to integrate the scheduling of virtual machines into the already existing HPC scheduler. A single resource management is handling resources used by bare metal computation and VMs. The advantage is that, it is not necessary to assign a compute nodes to run bare metal jobs or VMs. Thus a static partitioning of the cluster between virtualization and classic HPC nodes is not required.

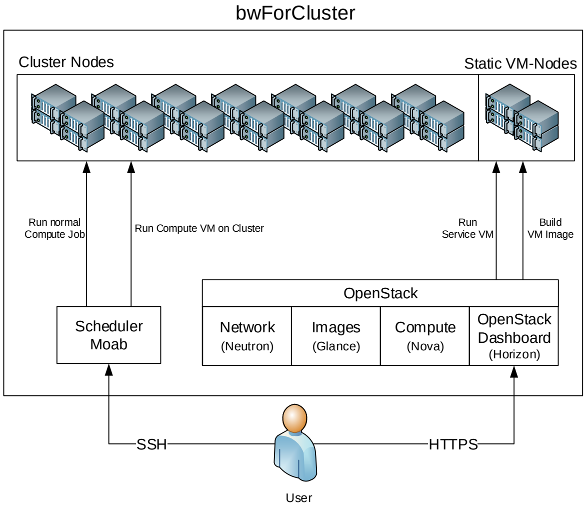

Figure 1. Cluster Structure

The workflow from the user's perspective is given in figure 1. If the user has a prebuild VM image he can directly upload the image to the cluster using the OpenStack Dashboard. If an image is not present, the user is able to use the cluster environment to run a VM on a static VM-Node (management server) in order to install the required software. The image of that VM can then be used as a new image for new VMs running on the cluster.

To run a compute VM on the cluster the user connects to the login server of the cluster and submits a compute job to the scheduler that includes a request for a virtual machine.

2.1. Integration of OpenStack

In a traditional HPC cluster the resource management is done via a scheduler. A user submits a job to the scheduler and the scheduler starts compute jobs on dedicated resources if they are available. Since users in the hybrid cluster are able to submit jobs to the scheduler for bare metal computation, it is required that the scheduler is also responsible for the resources allocated by virtual machines. If virtual machines are started on the cluster without the knowledge of the scheduler, resources could get assigned twice. For CPU resources this would slow down the computation. For memory resources it is even worse if more memory is allocated than physically available and the running jobs are canceled.

To solve the problem it is necessary that the scheduler is aware of the resources that virtual machines are using. A script was developed to submit a request for a virtual machine as a normal cluster job. In this way, the scheduler handles a request for a new virtual machine like any other cluster job and is not required to have knowledge of the virtualization environment. This makes the overall hybrid HPC cluster concept flexible and independent of the actual scheduler used.

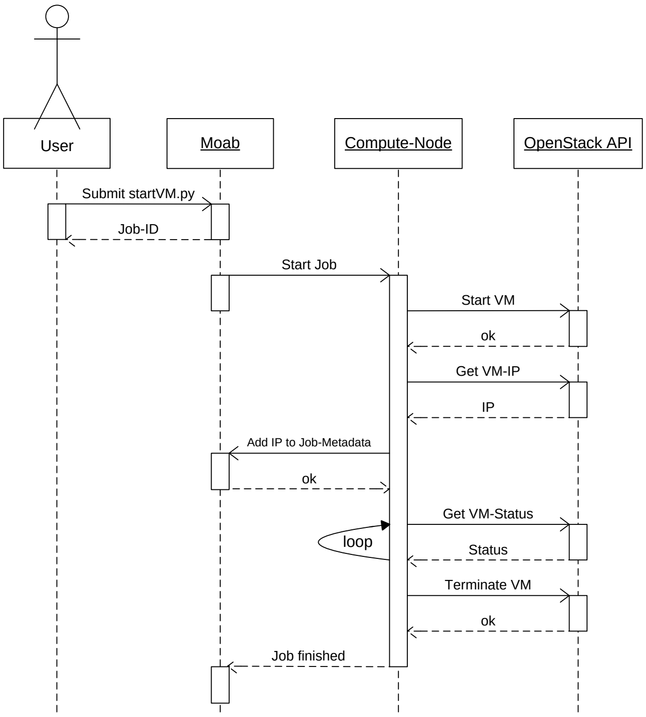

Figure 2. Workflow for starting VMs on the Cluster

The workflow of the script is given in figure 2. The script is submitted by the user as a normal cluster job to the scheduler. By doing so, the user is able to specify a wall time, number of CPU cores and amount of memory. These values are read by the job script and mapped to a matching OpenStack flavor. This way it is possible for the user to define the size of the virtual machine by the time he submits the job to the scheduler and the scheduler is aware of the resources used by the virtual machine.

After submission the scheduler assigns a free compute node to the job and starts the script on this node. The script reads the job metadata (requested wall time, CPU cores, memory) and requests OpenStack to start a virtual machine on the node the script is executed on.

If the job is then executed by the scheduler on the compute node, the script reads the environment variables HOSTNAME, PBS_NP, PBS_JOBID. The host name is required by OpenStack to start the VM on the correct compute node. The job ID is used to generate the VM name. Thus, it is possible to find a compute job for every VM name in the resource manager. PBS_NP corresponds to the requested number of CPUs and is mapped to an OpenStack-Flavor[1]. To start the virtual machine, the following information is passed to the OpenStack API: flavor, generated VM name, compute node hostname and the ID of the image.

Once the VM has started successfully, information of the VM network configuration is polled. If an IP address is successfully assigned to the VM the address is added to the metadata of the compute job. This allows the user to get the IP of a VM by requesting the job status from the scheduler. The IP can be used to login to the VM for monitoring and debugging purposes.

The status of the VM is monitored by the startVM-script. If the status is ACTIVE the script waits 30 seconds before checking the status again. In case of the status ERROR the VM is deleted. The status SHUTDOWN indicates that the VM was shutdown and all computations within the VM are finished. In this case the VM is deleted and the script exits, to signal the scheduler that the resources are free again. The shutdown of the VM can thus be used to signal the system that the VM is no longer needed. If a VM is not shutdown and reaches the maximum wall time, the job is canceled by the scheduler. For this a SIGTERM signal is sent to the job. This single is captured by the startVM script and the VM is deleted. The same procedure is triggered if the job is canceled by the user.

2.2. Testbed Setup

The described hybrid HPC cluster is developed and tested as testbed cluster with 1248 Cores at the computing center at the University of Freiburg. The final cluster, named bwForCluster, with 15000 cores will be installed in spring 2016.

[1] Describes a predefined set of resources of CPU cores, RAM and Storage.

To test the system concept, tests were performed over many weeks. The following section describes the setup and discusses first results.

3. Integration of an HPC Cloud into HEP Workflows

In this section we switch from the provider perspective to the user perspective. The access to the new cloud-based resources should be seamless and hassle-free for HEP users so that no adaptations of the existing workflows are required. Additionally, it should be possible to request resources on-demand in a dynamic manner. To achieve this, a flexible batch system as well as a central cloud manager are necessary.

3.1. A Flexible Batch System: HTCondor

HTCondor is a modular open-source workload management system using a client-server architecture [2]. Two important features make it the batch system of choice for the dynamic integration of cloud resources: It easily handles the addition and removal of worker nodes in the computing pool and it enables the access to resources in a private network or behind a firewall. Both are typical scenarios when using cloud resources for virtualized worker nodes.

3.2. A Cloud Meta-Scheduler: ROCED

A central component is required which dynamically manages virtual machines depending on the demand for computing resources. For this purpose, the cloud meta-scheduler ROCED (**R**eponsive **O**n-demand **C**loud **E**nabled **D**eployment) has been developed at the EKP since 2010 [3]. ROCED is written in a modular structure, the interfaces to batch systems and cloud sites are implemented as modules. This makes ROCED independent of a specific user group or workflow. For the described use case, the HTCondor and a special HPC-cloud module were implemented. The ROCED source code is available on GitHub [4].

3.3. Virtual Machine Management Challenges

Encapsulating VMs in user jobs increases the complexity of managing these machines. The underlying resources are shared with other communities, which means that a VM request might not get processed immediately when the cluster is under heavy load. In addition, the run time of VMs is limited to the maximum wall time allowed for user jobs on the cluster, which is 4 days. As a result, static booking of VMs for several weeks is not possible. These conditions needed to be taken into account when provisioning resources.

3.4. Integration of a Remote HPC Cloud

The following list describes the interaction of all components, illustrated in Figure 3:

(i) HEP user submits computing job to HTCondor, specifying a requirement to send the job to a cloud resource.

(ii) ROCED continuously monitors the demand for computing resources.

(iii) If required, a batch job containing a VM request (presently 4 vCPUs, 8 GB memory, but flexible) is sent to the job scheduler of the bwForCluster.

(iv) The startVM script is scheduled on the cluster and requests an OpenStack-based VM on the allocated worker node.

(v) After booting, the VM integrates in the physics institute's HTCondor batch system and acts as an additional worker node.

(vi) HEP jobs get scheduled on this virtualized HEP worker node. The VM can be reused for multiple user jobs, it shuts down when it is idle for more than 5 minutes. In addition, the job slots are set to drain mode when the underlying batch job is close to the wall time limit.

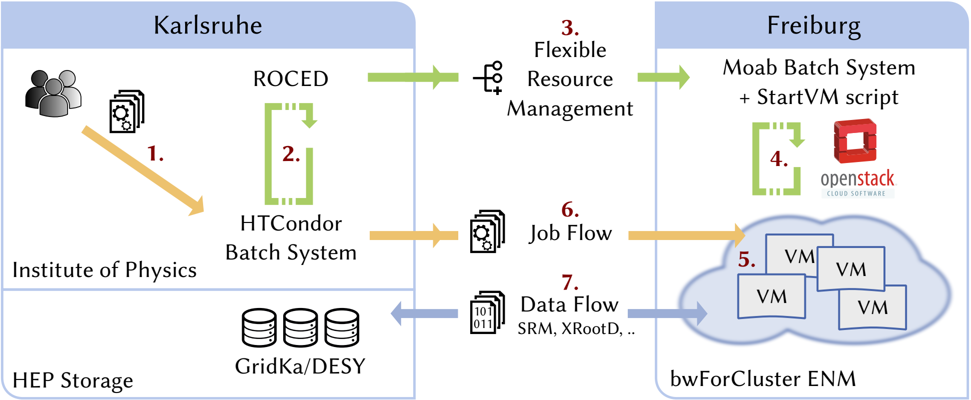

Figure 3. Interplay of tools and technologies for accessing cloud resources at the bwForCluster ENM. Accessing virtualized HEP worker nodes is completely transparent to the end user. No adaptations to the existing workflows of job submission are required.

(vii) Data can be read from and written to any HEP storage via common HEP protocols such as XRootD [5] and SRM [6].

3.5. Building the Virtual Machine Image

The virtual machine disk image containing the HEP experiment software stack is created in an automated manner. This is done by using the OZ toolkit [7] to install the SLC 6.7 operating system [8], as well as CernVM-FS [9] to access Grid UI and CMS software and the HTCondor client to integrate the VM at boot-time. Additional scripts for monitoring, auto-shutdown and auto-drain were implemented. The templates for the images can easily be shared between various cloud sites to ensure the exact same software environment on each site.

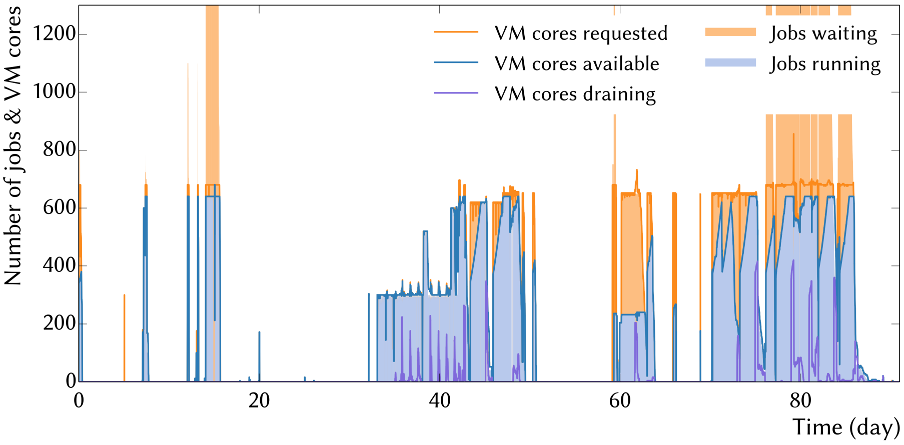

Figure 4. Allocation of virtualized HEP worker nodes at the bwForCluster. Up to 640 CPU cores were used. The discontinuous behavior originates from the many different HEP workflows and submission scenarios. For example testing the workflow with a few jobs and then submitting the whole analysis later or increasing the size of a Monte-Carlo simulated sample by adding more computing jobs after a first batch is finished.

4. Stability and Long-Term Evaluation

The described setup has been successfully in production use for over 5 months: in total 412 000 h of CPU time have been used. Various kinds of HEP workflows such as Monte-Carlo event generation and detector simulation and skimming of datasets were processed. The complete usage overview is shown in Figure 4. The discontinuous usage of resources is typical for HEP users and is one of the reasons why sharing computing power between multiple communities to buffer peak loads is necessary.

5. Conclusion

The hybrid approach presented in this paper combines the advantages of classical HPC data centers, namely complete software environment and batch system, with the flexibility of virtualization to allow groups of scientists, such as HEP users, to deploy a software setup tailored to their needs. The dynamic nature of integration of such a hybrid center into HEP workflows guarantees that only the required resources are booked and freed for other user groups in times of low demand. The booking of virtual machines on a shared cluster is not limited to the used Moab or OpenStack system and can also be applied to other clusters using different software setups.

[1] OpenStack website `https://www.openstack.org` (25/02/2016)
[2] HTCondor website `http://research.cs.wisc.edu/htcondor` (17/02/2016)
[3] Hauth T, Quast G, Kunze M, Bge V, Scheurer A and Baun C 2011 *Journal of Physics: Conference Series* **331** 062034 URL `http://stacks.iop.org/1742-6596/331/i=6/a=062034`
[4] Erli G, Fleig G, Hauth T and Riedel S Roced cloud meta-scheduler project website `https://github.com/roced-scheduler/ROCED` (17/02/2016)
[5] XRootD website `http://xrootd.org` (28/02/2016)
[6] SRM website `https://sdm.lbl.gov/srm-wg/index.html` (28/02/2016)
[7] OZ website `https://github.com/clalancette/oz` (17/02/2016)
[8] ScientificLinux Cern website `https://linux.web.cern.ch/linux/` (25/02/2016)
[9] CVMFS website `http://cernvm.cern.ch/portal/filesystem` (25/02/2016)