Optimizing OpenStack Nova for Scientific Workloads

Belmiro Moreira[1,*], *Spyridon* Trigazis[1], and *Theodoros* Tsioutsias[1]

[1]European Organization for Nuclear Research (CERN), CH-1211 Geneva, Switzerland

Abstract. The CERN OpenStack cloud provides over 300,000 CPU cores to run data processing analyses for the Large Hadron Collider (LHC) experiments. To deliver these services, with high performance and reliable service levels, while at the same time ensuring a continuous high resource utilization has been one of the major challenges for the CERN cloud engineering team. Several optimizations like NUMA-aware scheduling and huge pages, have been deployed to improve scientific workloads performance, but the CERN Cloud team continues to explore new possibilities like preemptible instances and containers on bare-metal. In this paper we will dive into the concept and implementation challenges of preemptible instances and containers on bare-metal for scientific workloads. We will also explore how they can improve scientific workloads throughput and infrastructure resource utilization. We will present the ongoing collaboration with the Square Kilometer Array (SKA) community to develop the necessary upstream enhancement to further improve OpenStack Nova to support large-scale scientific workloads.

1 Introduction

During the last 5 years, CERN has been running a large cloud infrastructure that provides an IaaS service to the CERN community. The cloud infrastructure brought several advantages when compared to the previous model, the manual allocation of physical resources. It improved the user responsiveness with a self-service kiosk for virtual resources, enabled cloud interfaces like OpenStack APIs and EC2 API, improved efficiency over the entire lifetime of the resources [1]. However, in order to have enough capacity to process the data from the next generation experiments we will need to reduce all possible overheads and increase the resource utilization.

The High Luminosity Large Hadron Collider (HL-LHC) program will require several times the current compute capacity. At same time the Square Kilometre Array (SKA) will face similar challenges to process the huge amounts of data from its high sensitive radio telescopes. Because of that, CERN and SKA are collaborating [2] to understand how they can improve the efficiency of the current infrastructures. In this paper we will focus on the CERN cloud infrastructure use-case.

In this paper we present two different approaches to improve the efficiency of cloud infrastructures, based in the CPU utilization. The first is to avoid virtualization and instead use containers on bare-metal provisioned by containers orchestrators. The second is to use preemptible instances in order to run worloads in unused resources. Preemptible instances

*Corresponding author: belmiro.moreira@cern.ch are virtual machines that are created using tenant[1] resources that have been allocated but not instanciated. Depending in the cloud orchestration tool this usually means quota allocated but not used. Preemptible instances are deleted as soon a tenant tries to create virtual machines within its quota.

The paper is organized as follows: Section 2 presents a summary of the CERN cloud architecture including a brief description of the OpenStack project. Section 3 describes the virtualization overhead considering the virtual machine size and the optimizations done in order to reduce this overhead. On Section 4 we introduce containers on bare-metal. We describe how workloads can have a performance similar to bare-metal eliminating the virtualization overhead and how OpenStack Magnum can deploy clusters directly on bare-metal. Section 5 introduces the concept of preemptible instances, the proposed architecture design for the OpenStack project, and how it can improve the resource utilization of a large infrastructure.

2 CERN cloud

The CERN cloud infrastructure is based on the OpenStack project [3]. The OpenStack project is an open source solution that allows "to build a platform that is easy to use, simple to implement, interoperable between deployments, works well at all scales, and meets the needs of users and operators of both public and private clouds" [4] that was initially made available by NASA and Rackspace.

The CERN cloud implements several OpenStack projects to offer different services including the easy deployment of virtual machines, containers, block storage and shared file systems.

The infrastructure is built on top of more than 10,000 compute servers (300,000 cores) and it runs in two CERN data centers (Geneva and Budapest), configured as single region. Users only have one endpoint to access the CERN cloud.

The CERN cloud infrastructure is organized using Cells [5]. Cells allow to split the infrastructure into smaller failure domains that are easier to manage, scale and recover in case of an incident. Currently CERN Cloud has more than 70 cells available. Cells are designed to increase the infrastructure availability, not application availability. For application availability CERN Cloud offers 5 availability zones, 3 in the Geneva data centre and 2 in the Budapest data centre.

The OpenStack control plane is deployed in virtual machines that run inside the same cloud infrastructure. Physical servers are used to bootstrap the configuration. Each virtual machine only runs the services of a particular OpenStack project (such as Nova or Glance). The databases of each service are stored in different MySQL instances. This enables independent upgrades of the different OpenStack projects.

CERN cloud infrastructure is today used by more than 3,000 users with 4,500 tenants that host more than 35,000 VMs, 400 container clusters and 6,000 block storage volumes.

3 Virtualization overhead

CERN cloud infrastructure has more than 80 percent of the resources dedicated for the analysis of the LHC data using the batch service. The batch service runs on top of the cloud infrastructure and uses virtual machines to run the processing jobs. Usually these virtual machines are very large with more than 16 vCPUs and 32 GB of RAM.

We observed that if the number of vCPUs in the virtual machines increases, the virtualization overhead would also increase [6]. The virtualization overhead was measured comparing

[1] OpenStack tenants represent the base unit of ownership in OpenStack

the difference between the results of the benchmark HEP SPEC 06 executed in bare-metal nodes and in virtual machines.

To reduce the virtualization overhead we have enabled several techniques in the compute nodes hosting the virtual machines, like KSM (Kernel Same-page Merging), EPT (Extended Page Tables), NUMA (Non-Uniform Memory Access) with vCPU pinning, and huge pages. With these features enabled we reduced the virtualization overhead to less than 4 percent in large virtual machines [6] [7].

Considering all the advantages and flexibility of a virtualization environment, the virtualization overhead that we experience is not a large penalty. However, in a large infrastructure, the virtualization overhead can represent the lost of a significant compute capacity. To overcome this problem we investigated how we can reduce or even eliminate the virtualization overhead without losing the flexibility of a shared, consolidated and self-service environment.

4 How to remove virtualization overhead

In many cases a user or group runs an application using the microservices architecture which breaks monolithic applications in many inter-connected components. An example of this architecture may include REST API servers, databases, etc. A usual pattern of deploying such an application is to run each component in a separate virtual machine. This choice would add the overhead of virtualizing a machine for each component. Instead of using a virtual machine per component, operators can leverage containers to achieve operational agility and good process isolation. In cases where operators control the hardware as well as the virtual machines, removing the virtualization overhead gives good a performance improvement.

4.1 Containers

Containers are a form of operating system virtualization leveraging features of the Linux kernel. A container is an isolated process in the operating system whose level of isolation is defined using control groups (cgroups), namespaces, kernel capabilities, seccomp, SElinux and AppArmor:

- cgroups limit the access to resources like CPU, memory, disk I/O, network, etc.
- namespaces partition kernel resources like mounts (mount), process id tree (pid), networks (net), inter-process communication (ipc), UTS for different host and domain names (uts), user namespaces (user)
- capabilities limit the privileges of a process which runs as the root user
- seccomp profiles filter the system call a process can do in the system
- SElinux and AppArmor enforce mandatory access control (MAC) to processes

4.2 Containers on bare-metal

One of the OpenStack services of CERN's cloud is Magnum. Magnum allows users to create container clusters with an API call via the command line or web interface. Users can select different cluster types among Kubernetes [8], Docker Swarm and Mesos, with Kubernetes being the most popular one. Magnum orchestrates Nova instances in a cluster setting the selected orchestration engine. Then users can retrieve the TLS certificates from the Magnum API and interact with the orchestration engine's API directly. Using Magnum, users do not need to manage machines, instead they can only care about the total capacity of their clusters.

Applications are described by Kubernetes, Docker Compose or Marathon manifests, then it is up to the orchestrator to make sure that the application is running. Since applications run in clusters of machines without being tied to specific machines, the storage requirements are different to traditional applications. Container orchestration engines like Kubernetes were initially designed for stateless applications, like websites where the state is either static or in a remote database. Stateful application need to store their state in remote storage like NFS or object storage like Amazon S3. The container service is integrated with CVMFS [9] for software distribution, EOS [10] for access to petabytes of LHC data, OpenStack Cinder for block storage and CephFS via OpenStack Manila for shared filesystems.

The initial implementation of Magnum allowed only virtual machines as compute instances but recently with the adoption and evolution of the OpenStack Ironic project, it is possible to use physical machines provisioned by Ironic. Users can select a Nova flavor which corresponds to physical machines and run their cluster in physical compute instances. The design and workflow for managing applications can be the same as virtual machines, however the underlying resources are usually more powerful since the cluster runs on servers with many cores and a lot of RAM comparatively to small or medium size VMs. Clusters made up of many physical machines, are a good fit for batch systems when the resources are used only from a single team and the additional isolation from VMs is not required. In this scenario, users can use all the computing capacity of the hardware. Finally, distributed storage systems that run on dedicated hardware do not have strong isolation requirements between running process, so then can benefit from the operational agility that containers provide, without the virtualization overhead.

In late 2017, OpenStack Ironic became available to CERN users and in late 2018 users could use it with OpenStack Magnum to create container clusters. Additionally, during 2018 OpenStack Manila became available to users and a CSI plugin [11] was implemented to allow users to run stateful applications easily using shared storage. The Container Storage Interface (CSI) [12] is an interface in the container ecosystem which can be implemented by storage system providers. In the CERN cloud, users can use csi-cvfms and csi-cephfs for access to cvmfs and cephfs respectively.

5 Resource utilization

Either in a public or a private cloud, optimizing the utilization of the infrastructure is really important for cloud operators. To achieve this, idling resources have to be eliminated. Often, quotas per user or groups of users, are used to ensure the fair sharing of the infrastructure. But quotas are hard limits and their extensive use leads to dedicating resources to workloads even if they are not used. Resources in idle state can occur, showing lower cloud utilization than the full potential of the acquired equipment while the users' requirements are growing.

5.1 Preemptible instances

The concept of preemptible instances can be the solution to the problem described above. Instances of this type can be created using idle resources in the system, even after a user's quota is exhausted. The main difference with the normal instances is that preemptible instances will be terminated in case higher priority workloads need the resources they consume. Overcommitting resources gives operators the ability to provide a more "elastic" capacity and at the same time, it enables the handling of increased demand for resources, with the same infrastructure just by maximizing the cloud utilization.

5.2 Current Work

Currently, OpenStack does not support preemptible instances. Adding this functionality would mirror the AWS Spot Market and the Google Preemptible Instances. The prerequisites for providing preemptible instances with OpenStack can be summed up to the following:

- Tagging instances as preemptible,
  in order to be able to distinguish from normal instances, preemptible instances need to be tagged at creation time with an immutable property.
- Access to preemptible instances,
  operators should be able to control which users/tenants are allowed to spawn and use preemptible instances.
- Control preemptible resources (optional),
  operators should have the ability to limit the resources each user can use for creating preemptible instances to avoid misuse of the functionality. This is optional and it depends on the financial model in each use case. For example, it would be useful to control the preemptible resources in private clouds. But in the case of public clouds, where users pay for what the resources they use, this is not relevant.

According to Nova team's suggestion and in order to identify the changes needed to provide preemptible instances with OpenStack, there is an ongoing effort, at CERN, to introduce a prototype orchestrator for preemptible instances. To minimize the changes in OpenStack Nova, the orchestrator is designed as a stand-alone OpenStack service.

To focus on the functionality of the orchestrator, it was decided to use dedicated tenants (OpenStack tenants) for hosting preemptible instances addressing the prerequisites in the following way:

- Tagging instances as preemptible,
  instances will inherit this property from the tenant to which they belong.
- Access to preemptible instances,
  by using preemptible tenants, operators can control which users will be able to spawn preemptible instances.
- Control preemptible resources (optional),
  if this applies to the use case, the operator is able to enforce quotas on the preemptible tenants to limit the resources each user can use for preemptible instances.

5.3 Service workflow

As it can be seen in Figure 1, the orchestrator, codenamed Aardvark [13], is triggered as soon as an instance fails to be built due to the lack of resources. This is done by consuming notifications emitted by the OpenStack Nova and specifically the Nova scheduler. Instances which cannot be created for this reason are set to the PENDING state until the resource cleanup is finished.

At that point, Aardvark applies the configured strategy (strict/random) to decide which preemptible instances have to be deleted in order to free up the resources needed for the pending instance. According to the result of the cleanup procedure, Aardvark, using the Nova API, either rebuilds the instance or it resets its state to ERROR.

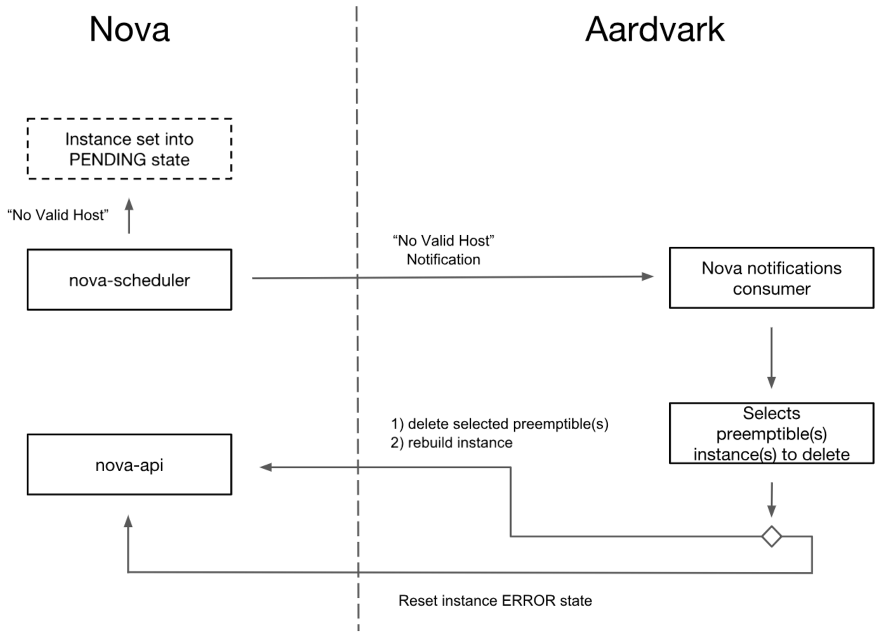
Nova
Aardvark
Instance set into PENDING state
"No Valid Host"
nova-scheduler
"No Valid Host" Notification
Nova notifications consumer
Selects preemptible(s) instance(s) to delete
1) delete selected preemptible(s)
2) rebuild instance
nova-api
Reset instance ERROR state

Figure 1. The Aardvark service workflow

5.4 OpenStack Nova changes

Since this functionality is not supported in the current releases of OpenStack and in order to enable Aardvark, there are two changes needed in Nova:

- The PENDING instance state has to be added in Nova [14].
- Currently it is not allowed to rebuild instances that failed while scheduling, but the change is already proposed upstream [15].

5.5 Future Work

Aardvark will be deployed in CERN cloud during the second trimester of 2019. This will allow for further verification of the functionality at scale and provide with feedback from users. Furthermore, the authors plan to introduce Aardvark to the OpenStack Community as a new OpenStack project to identify and include more use cases.

6 Conclusion

The High Luminosity Large Hadron Collider (HL-LHC) and the Square Kilometre Array (SKA) will produce unprecedented amounts of data that needs to be stored and analysed. This will require large compute and data infrastructures. In these environments, small inefficiencies can be translated in the underutilization of a large number of resources delaying the scientific results. Because the similar challenges, CERN and SKA are working together to design and build a flexible and efficient compute infrastructure.

In this paper we discussed how to overcome the overhead of a virtualized infrastructure and to leverage unused resources with preemptible instances.

These solutions are being evaluated for the next compute infrastrctures of CERN and SKA.

Containers can provide enough application isolation without the overhead of virtualization technologies. Running the workloads in containers on bare-metal allows to have compute workloads performance close to the bare-metal node [7]. Initials tests have been done with OpenStack Magnum to deploy large Kubernetes and Docker Swarm clusters deployed directly in the bare-metal nodes. OpenStack Magnum requires very few changes to support deploying Kubernetes and Docker Swarm clusters in bare-metal nodes. Also, using this container cluster orchestrator we keep the flexibility of the cloud infrastructure to rapidly change workloads.

To improve the resource utilization we are implementing preemptible instances in the OpenStack project. This allows to consume all available resources without blocking the execution of workloads with an available quota. All this work has been done in collaboration with the upstream OpenStack community.

References

[1] P. Andrade, T. Bell, J. van Eldik, G. McCance, B. Panzer-Steindel, M.C. dos Santos, S. Traylen, , U. Schwickerath, Journal of Physics: Conference Series **396**, 042002 (2012)

[2] *Cern courier*, https://cerncourier.com/ska-and-cern-co-operate-on-extreme-computing/

[3] *Openstack project*, https://www.openstack.org

[4] *Openstack mission*, https://wiki.openstack.org/wiki/Main_Page

[5] T. Bell et al., J. Phys. Conf. Ser. **664**, 022003 (2015)

[6] *Optimisations of the compute resources in the cern cloud service*, https://indico.cern.ch/event/384358/contributions/909247/

[7] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, *An updated performance comparison of virtual machines and linux containers*, in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)* (IEEE, 2015), pp. 171–172

[8] *Kubernetes project*, https://kubernetes.io

[9] *Cern-vm filesystem*, https://cernvm.cern.ch/portal/filesystem

[10] *Eos filesystem*, http://eos-docs.web.cern.ch/eos-docs/

[11] *Container storage and cephfs (part 2): Auto provisioning manila shares*, https://techblog.web.cern.ch/techblog/post/container-storage-manila-provisioner-part2/

[12] *Container storage interface (csi)*, https://github.com/container-storage-interface/spec/blob/37e74064635d27c8e33537c863b37ccb1182d4f8/spec.md

[13] *Aardvark repository*, https://gitlab.cern.ch/ttsiouts/aardvark

[14] *Pending instance state specification*, https://review.openstack.org/#/c/554212/

[15] *Rebuild instances specification*, https://review.openstack.org/#/c/554218/