DEEP-HybridDataCloud

ASSESSMENT OF AVAILABLE TECHNOLOGIES FOR SUPPORTING ACCELERATORS AND HPC, INITIAL DESIGN AND IMPLEMENTATION PLAN

DELIVERABLE: D4.1

Document identifier:	DEEP-JRA1-D4.1
Date:	29/04/2018
Activity:	WP4
Lead partner:	IISAS
Status:	FINAL
Dissemination level:	PUBLIC
Permalink:	http://hdl.handle.net/10261/164313

Abstract

This document describes the state of the art of technologies for supporting bare-metal, accelerators and HPC in cloud and proposes an initial implementation plan. Available technologies will be analyzed from different points of views: stand-alone use, integration with cloud middleware, support for accelerators and HPC platforms. Based on results of these analyses, an initial implementation plan will be proposed containing information on what features should be developed and what components should be improved in the next period of the project.

Copyright Notice

Delivery Slip

	Name	Partner/Activity	Date
From	Viet Tran	IISAS / JRA1	25/04/2018
Reviewed by	Marcin Plociennik Cristina Duma Aiftimiei Zdeněk Šustr	PSNC INFN CESNET	20/04/2018 25/04/2018 25/04/2018
Approved by	Steering Committee		30/04/2018

Document Log

Issue	Date	Comment	Author/Partner
TOC	17/01/2018	Table of Contents	Viet Tran / IISAS
0.01	06/02/2018	Writing assignment	Viet Tran / IISAS
0.99	10/04/2018	Partner contributions	WP members
1.0	19/04/2018	Version for first review	Viet Tran / IISAS
1.1	22/04/2018	Updated version according to recommendations from first review	Viet Tran / IISAS
2.0	24/04/2018	Version for second review	Viet Tran / IISAS
2.1	27/04/2018	Updated version according to recommendations from second review	Viet Tran / IISAS
3.0	29/04/2018	Final version	Viet Tran / IISAS

Table of Contents

Executive Summary

The DEEP-HybridDataCloud (Designing and Enabling E-Infrastructures for intensive data Processing in a Hybrid DataCloud) is a project approved in July 2017 within the EINFRA-21-2017 call of the Horizon 2020 framework program of the European Commission. It will develop innovative services to support intensive computing techniques that require specialized HPC hardware, such as GPUs or low-latency interconnects, to explore very large datasets.

This document describes the state of the art of technologies for supporting bare-metal, accelerators and HPC in cloud. Based on the analysis, an initial implementation plan for improving the support of accelerators and HPC in DEEP-HybridDataCloud is proposed.

There are two main approaches to reducing the overhead of virtualization layers and reaching bare-metal like performance in cloud computing: 1) to use paravirtualization technologies in hypervisors and 2) to replace hypervisors with container technologies. Although paravirtualization technologies in hypervisors can improve performance, they are still cumbersome for deployment and use. On the other hand container technologies have been rapidly improving in recent years and have proved themselves as viable alternative to application delivery and even full virtualization for cloud computing. There are many different implementations of container technologies and also cloud platforms supporting them, from traditional Cloud management frameworks like OpenStack or container-centric management environment like Kubernetes.

For what concerns support for accelerators, paravirtualization techniques have several severe limitations while GPU virtualization is still not generally available on common hardware. On the other hand, recent developments in GPU runtime support like nvidia-docker have greatly improved the usability, portability and stability of container technologies. Therefore the use of container technologies will be the approach chosen in DEEP-HybridDataCloud project. The DEEP-HybridDataCloud has selected two implementations to be further improved: **nova-lxd**, as a replacement of full Cloud hypervisors, and **udocker**, as container technology to be used in the case of HPC platforms.

HPC computing model is different from Cloud computing model in many aspects: architecture, software installation, queuing system, user access and so on. Therefore it is necessary to work out a solution that allows to efficiently use HPC resources by high level services. There are two possible approaches to making HPC resources available for PaaS services: 1) to access an HPC management system and submit batch jobs, and 2) to allow HPC nodes to be managed by a Cloud management system. These approaches are not exclusive and none of them can be chosen as the best solution, as it strongly depends on local HPC conditions and politics. The work on integration with an HPC platform will be tightly coordinated with WP5 addressing the PaaS layer.

1. Introduction

The DEEP-HybridDataCloud (Designing and Enabling E-Infrastructures for intensive data Processing in a Hybrid DataCloud) is a project approved in July 2017 within the EINFRA-21-2017 call of the Horizon 2020 framework program of the European Community. It will develop innovative services to support intensive computing techniques that require bare-metal performance, specialized accelerators such as GPUs and HPC platform in cloud, to explore very large datasets.

The objective of WP4 of the DEEP-HybridDataCloud project is to fulfill the requirements mentioned above by working as closely as possible with hardware resources, exploiting the full potential of computation performance provided by the hardware including accelerators, low-latency interconnects and HPC platforms. This workpackage will cooperate tightly with WP5 High Level Hybrid Cloud solutions where the resources provided in this workpackage will be managed and accessed via PaaS Orchestration service provided by WP5.

This document describes the state of the art of technologies for supporting bare-metal, accelerators and HPC in cloud and initial implementation plan for WP4. The bare-metal performance can be achieved either by using paravirtualization technologies or by using container technologies. The later mentioned technologies have been emerging as a viable alternative to application delivery and even replacement for traditional full virtualization for cloud computing. Available technologies will be analyzed from different views: stand-alone use, within cloud middleware, support for accelerators and HPC platforms. Based on the results of these analyses, an initial implementation plan will be proposed containing information on what features should be developed and what components should be improved in the next period of the project.

2. Available technologies for obtaining bare-metal like performance

Virtualization is one of the key technologies for enabling cloud computing. With the introduction of hardware assisted virtualization in common processors (Intel VT-x and AMD AMD-V), it is possible to create a fully simulated hardware environment (virtual machine) for executing unmodified guest operating system in complete isolation. That is the base of "Infrastructure as a service" (IaaS) cloud computing.

Full virtualization requires explicit supports in hardware components for performing efficiently. The supports are commonly available in modern processors, but not in other devices like network and graphic cards. As the results, there are high additional overheads for accessing these devices, for example network latency that is critical for high performance computing. There are two main approaches to reduce the overhead of virtualization layers and to reach bare-metal like performance in cloud computing: 1) to use paravirtualization technologies in hypervisors and 2) to replace hypervisors by container technologies.

Paravirtualization approach allows virtual machines to access the devices directly, bypassing hypervisor layers (PCI passthrough, SR-IOV) or via special modified drivers (virtio). That can reduce the overheads of virtualization but also create certain flaws in virtualization layers because of bypassing. On the other hand, container technologies rely on isolation provided by operating systems, e.g. namespaces in Linux, instead of hardware virtualization. As the results, the container technologies eliminate overheads caused by hardware virtualization, however, the containers must share the kernel with the host operating system. The container technologies have lower level of isolation than full virtualization and the containers must use the same operating system as the host. More details about the technologies and their comparison are provided in the following subsections.

2.1. Paravirtualization technologies

2.1.1. SR-IOV and PCI Passthough

PCI Passthrough is a technology that allows assigning direct access to a PCI/PCIe device to a virtual machine guest. This removes the virtualization overhead associated with copying of data from a virtual to a real device driver. Since GPUs are nowadays usually PCIe devices, PCI Passthrough technology allows the virtualization of GPU access.

PCI Passthrough is supported in both the KVM [KVMSRIOV] and Xen [XENSRIOV] hypervisors (and the derived commercial products by Red Hat and Citrix), and numerous others Level 1 and Level 2 hypervisors.

PCI Passthrough is enabled by the PCI-SIG Single Root I/O Virtualization standard [PCISIGa] (SR-IOV). This standard provides a consistent way to bypass the VM host's involvement in data movement by providing each virtual machine with its own memory space, DMA channels and interrupts. As a result, a real PCI device will appear in the device tree as several devices, each assignable to a different virtual guest. While the original PCI device is called a Physical Function (PF) and is discovered, configured and managed as a normal PCI device. This PF is also the point of management of Virtual Functions (VF) – the virtual representations of the device, with more restricted set of functionality. However, even VFs need to be supported by the hardware of the PCI device, so the number of VFs one PF can configure is limited. The relationship between a physical PCI device, its PF and VFs, the host and guests is shown in Figure 1. Unlike direct assignment of hardware to a virtual machine in PCI Passthough, using SR-IOV's VFs provides a level of flexibility and the option to reassign the device to more than one virtual machine.

Hardware support of the SR-IOV standard is called VT-d [VTD] by Intel and IOMMU by AMD. It is usually necessary to enable these standards in the BIOS of the computer on which we wish to use the SR-IOV virtualization capabilities.

On Linux, the SR-IOV standard and its configuration capabilities are supported by a kernel module. As the VFs are configured anew at each boot, the configuration of a virtual PCI device may not be persistent, for example, a NIC VF will have a different MAC address assigned after each reboot.

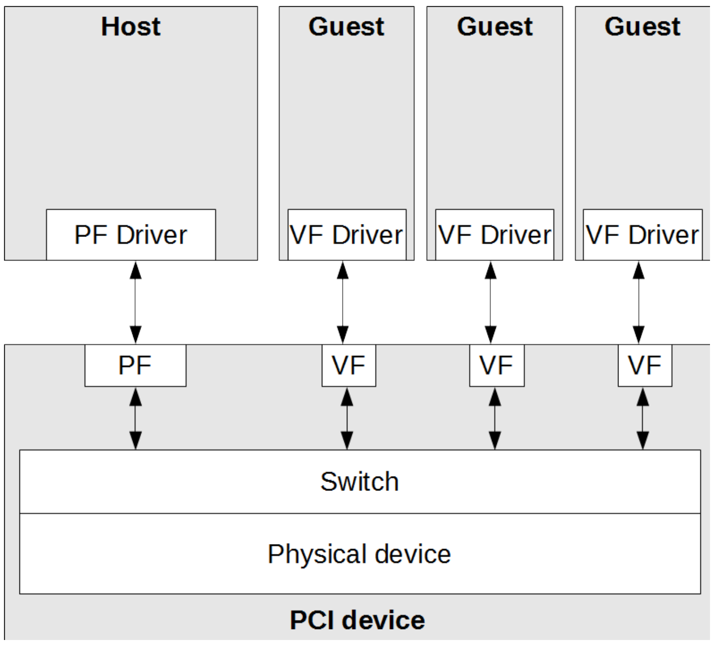


Figure 1. The illustration of SR-IOV relationships between a physical PCI device, its PF and VFs and the host and guests in a virtualization environment

2.1.2. virtio

Virtio [VIRTIO] is a paravirtualization method of virtualizing access to mainly disks (block devices) and network interfaces. It is architecturally similar to Xen's paravirtualized driver – the disk or NIC driver in the guest machine is aware of being virtualized and cooperates with the hypervisor. It is implemented as an abstraction layer over devices, providing a standardized interface through which a guest OS can access simplified versions of block devices or NIC adapters.

Virtio is essentially a set of virtual devices and their drivers. It was devised by Rusty Russell as part of the Lguest [LGUEST], a Linux virtualization hypervisor included in the Linux kernel from version 2.6.23 (October 2007) to version 4.14 (November 2017). Lguest is capable of virtualizing only 32-bit kernels, though an experimental version of Lguest64 [LGUEST64] exists.

A virtual device provided by virtio appears to the guest OS as a standard PCI device with Vendor ID of 0x1AF4 and Device ID between 0x1000 and 0x103F. The Subsystem ID identifies the type of the virtual device:

01	Network Card
02	Block Device
03	Console
04	Entropy Source
05	Memory Ballooning
06	IO Memory
07	RPMSG
08	SCSI Host

09	9P Transport
10	MAC802.11 WLAN

2.2. Container technologies

Most scientific computing facilities, such as HPC or grid infrastructures, are shared among different research disciplines, and thus the system software environment needs to be generic enough to accommodate different user and applications profiles; hence they are multi-user environments.

Because of managerial and technical constraints, such infrastructures cannot afford offering every research project a tailored environment in their machines. Therefore, the interest of exploring the applicability of containers technology on such systems is rather evident from the end-user point of view.

Researchers need then to customize their applications software to fit the computing center environment at the level of system software and batch system. Containers provide a way to pack and deploy software including all the dependencies in a way that can be executed in a seamless way, independently of the underlying Linux Operating System and environment. The main benefit of integrating the execution of containers in HPC systems would then be to provide a way to execute applications homogeneously across different resource centers [Gomes 2017].

2.2.1. Docker

Docker is a popular container platform intended both for own infrastructure and cloud. The basic building block is an image – a lightweight, standalone, executable package of a piece of software that includes everything needed to run it; i.e., code, runtime, system tools, system libraries, settings. Images are built from file system layers and share common files, thus minimizing disk usage and image size. An instantiated docker image is called a container and there can be multiple containers of the same image instantiated on the same machine.

On a single machine, containers share the host's system kernel, so they can start instantly and use fewer resources; i.e., RAM and CPU. Regardless of the environment, the software in the container will always run the same. The difference between containers and virtual machines is in their functionality. While virtual machines virtualize hardware, containers as an abstraction at the app layer virtualize the operating system. This makes containers more portable and efficient.

2.2.2. udocker

A basic user tool to execute simple docker containers in user space without requiring root privileges. Enables download and execution of docker containers by non-privileged users in Linux systems where docker is not available. It can be used to pull and execute docker containers in Linux batch systems and interactive clusters that are managed by other entities such as grid infrastructures or externally managed batch or interactive systems.

This tool is a wrapper around several tools to mimic a subset of the docker capabilities including pulling images and running containers with minimal functionality. It does not require any type of privileges nor the deployment of services by system administrators. It can be downloaded and executed entirely by the end user.

udocker is written in Python, it has a minimal set of dependencies so that can be executed in a wide range of Linux systems. It does not make use of docker nor requires its presence.

udocker "executes" the containers by simply providing a chroot like environment over the extracted container. The current implementation supports different methods to mimic chroot, enabling execution of containers without requiring privileges under a chroot like environment. udocker transparently supports several methods to execute the containers using tools and libraries such as:

- PRoot
- Fakechroot
- runC
- Singularity

2.2.3. Linux containers (LXC/LXD)

LXD is a next generation system container manager. It offers a user experience similar to virtual machines but using Linux containers instead. LXD is built on top of LXC to provide a better user experience. Under the hood, LXD uses LXC through liblxc and its Go binding to create and manage the containers.

It's an alternative to LXC's tools and distribution template system with the added features that come from being controllable over the network.

Clients, such as the command line tool provided with LXD itself, then do everything through that REST API. It means that whether you are talking to your local host or a remote server, everything works the same way.

It's image based with pre-made images available for a wide number of Linux distributions and is built around a REST API. The core of LXD is a privileged daemon that exposes a REST API over a local unix socket as well as over the network (if enabled).

Some of the biggest features of LXD are:

- Secure by design (unprivileged containers, resource restrictions and much more);
- Scalable (from containers on your laptop to thousands of compute nodes);
- Intuitive (simple, clear API and crisp command line experience);
- Image based (with a wide variety of Linux distributions published daily);
- Support for Cross-host container and image transfer (including live migration with CRIU);
- Advanced resource control (cpu, memory, network I/O, block I/O, disk usage and kernel resources);
- Device passthrough (USB, GPU, unix character and block devices, NICs, disks and paths);
- Network management (bridge creation and configuration, cross-host tunnels, …);
- Storage management (support for multiple storage backends, storage pools and storage volumes).

2.2.4. Singularity

Singularity is a container solution created by necessity for scientific and application driven workloads [Singularity]. This technology has been developed having in mind the necessities and specificities of HPC environments. Its goal is to have Linux Containers' advantages like portability, reproducibility and user freedom while maintaining traditional HPC security standards. It makes use of Kernel features like chroot, bind mounts, Linux namespaces, cgroups, etc. [Younge] [Godlove].

Like other container solutions, Singularity containers are distributed in image files. It uses a custom single image file that wraps all the necessary data to run that container. Although this technology supports Docker image files, those must be imported beforehand.

Pros

- Developed having HPC systems in mind;
- Experimental GPU support;
- Single image file;
- Docker images compatible (through import)
- Support for MPI applications (capability of running containerized, multi-node jobs using native RDMA-enabled MPI)

Cons

- Own format image files;
- Experimental GPU support;
- Requires image import procedure for non-Singularity images;
- Requires root privileges for installation;
- Requires system administrator's intervention.

2.2.5. Other available technologies

Shifter

Shifter is a platform designed to allow users to “safely and efficiently run Docker containers in an HPC environment”, developed at NERSC [Shifter]. It is released by CRAY as a part of the default installation in its CLE6.0 operating system, even though tools are available to install it on CentOS6 and CentOS7. It is currently used in production environments at a handful of sites, notably at NERSC in the US and CSCS in Switzerland.

Like Singularity, the core idea is to run user-defined containers in userspace.

Main features:

- Specifically designed to run Docker images, no image building capabilities
- Image Gateway service pulls images from DockerHub (or private repo) and converts them to single-file squashFS-based
- Simple binary on execution node starts the image
- Minimal MPI support, GPU support is currently being refactored.

Charliecloud

Charliecloud [CharlieCloud] is a lightweight tool to run unprivileged Linux containers on HPC facilities. Developed at LANL, it is a minimal system (about 1000 LoC) with an API to run containers on worker nodes. Container building is done by wrapping Docker in a sandboxed environment, and the Docker ecosystem, including DockerHub, is available. The container is then run on nodes by a separate runtime, independent of Docker.

2.3. Comparison of technologies

The brief comparison of available technologies for virtualization is provided in Table 1.

Technology	Full virtualization	Paravirtualization	OS container	Application container
Performance	Low performance for GPUs and network due to high overhead of virtualization	Better performance for specific devices supporting paravirtualization	Practically native performance	Practically native performance
Limitation	No	Hardware dependence, potential problem with suspending/migration	The same kernel between host and container (therefore same type of operating system) Possible problems when applications try to modify kernel configuration (e.g. loading driver)	The same kernel between host and container (therefore same type of operating system) Possible problems when applications try to modify kernel configuration (e.g. loading driver)
Security	Highest level of isolation	Hardware-assisted isolation (IOMMU)	Software isolation in OS kernel (namespaces)	Software isolation in OS kernel (namespaces)
Software / Privilege requirement	Pre-installed hypervisor	Pre-installed hypervisor	Installation requires root access, creating container in user space	Various: Docker requires root access for creating container, Singularity requires root access for installation but can create container in user space, udocker does not require root access at all
Usability	Virtualization of unprepared	Virtualization of environments and	Replacement of VMs	Application delivery

	environments, operating systems not supporting virtualization (as a guest)	operating systems that support the concept of paravirtualization, obtaining better performance than full virtualization		
Portability	Depending on type of hypervisor: Type 1 – most are portable across hardware Type 2 - requires support for specific OS on which the hypervisor is to be deployed	Both the host OS and the guest OS have to support paravirtualization; not very portable	Supported only on certain operating systems, the created container is not easily portable to a different host	Portability depends on the support of respective container technologies for various operating systems. Very portable across supported operating systems.
Implementation	KVM, VirutalBox, VMWare, Xen, Hyper-V	virtio, SR-IOV, Xen	LXC/LXD	Docker, udocker, Singularity

Table 1. Virtualization technology comparison

3. Supports for paravirtualization and containers in cloud middleware

3.1. OpenStack

OpenStack is a manager of cloud systems that governs large pools of compute, storage, and networking resources. Currently, it supports many hypervisors for the management of virtual machines as well as containers (see Figure 2). Most installations use only one hypervisor type. However, it is possible to schedule different hypervisors within the same installation, but multi-hypervisor OpenStack cloud requires at least one compute node for each hypervisor type.

OpenStack supports following hypervisor drivers:

- **Ironic** – Not a hypervisor in the traditional sense, this driver provisions physical hardware through pluggable sub-drivers (for example, PXE for image deployment, and IPMI for power management);

- **KVM** – Kernel-based Virtual Machine. The virtual disk format that it supports is inherited from QEMU since it uses a modified QEMU program to launch the virtual machine. The supported formats include raw images, the qcow2, and VMware formats;
- **VMware vSphere** – runs VMware-based Linux and Windows images through a connection with a vCenter server or directly with an ESXi host;
- **Xen** – XenServer, Xen Cloud Platform (XCP), used to run Linux or Windows virtual machines (a user must install the nova-compute service in a para-virtualized VM);
- **Hyper-V** – Server virtualization with Microsoft's Hyper-V, use to run Windows, Linux, and FreeBSD virtual machines. Runs nova-compute natively on the Windows virtualization platform;
- **Virtuozzo** – or its community edition OpenVZ, provides both types of virtualization: Kernel Virtual Machines and OS Containers;
- **LXD** – is supported by nova-lxd plugin, used to run Linux Containers (through libvirt);
- **Magnum** – It supports containers within Docker Swarm, Kubernetes, and Apache Mesos.

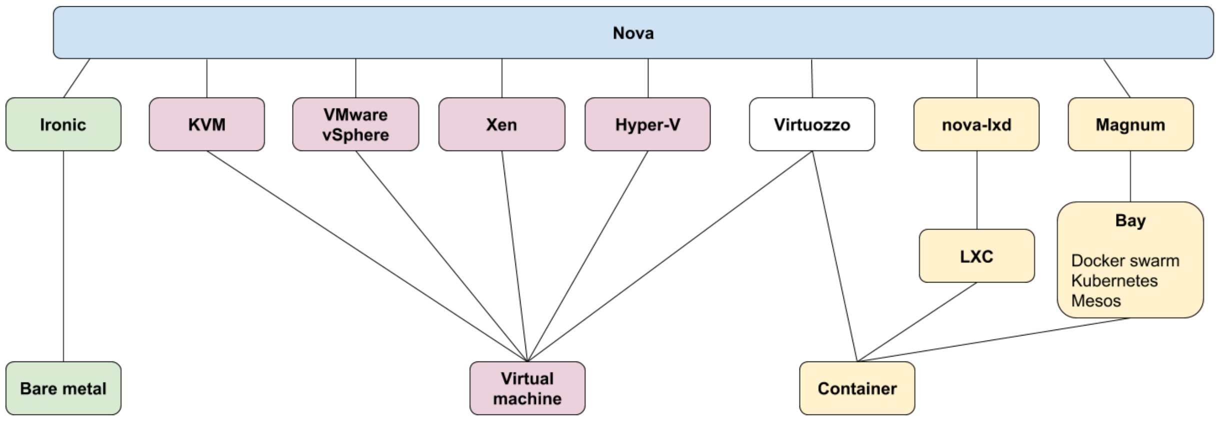


Figure 2. Overview of OpenStack resources

3.1.1. OpenStack Heat

OpenStack Heat is the orchestration service in OpenStack whose main objective is the implementation of a whole engine for managing the entire lifecycle of infrastructure and applications within an OpenStack Cloud. Orchestration not only makes it possible to reproduce a given deployment whenever needed, it also allows sharing these topologies with other users or Cloud infrastructures. This tool allows the IT administrator the automation of all the tasks involved in the process in one only step and finally, the creation of an orchestration service where the final users can launch their applications. The whole implementation of the service is based on a template where all the topology is defined by means of declarative languages, like HOT in the case of OpenStack Heat.

Heat provides an OpenStack native API and an API compatible with AWS CloudFormation. The architecture of Heat is shown in Figure 3. The user creates the template where the application is defined, and the OpenStack client sends this request to Heat for deploying the final environment in Cloud. Heat is composed by the following services:

- Heat: is the client, which communicates with the heat-api-cfn to execute AWS CloudFormation APIs;
- Heat-api: is an OpenStack REST API that processes user requests by sending them to the heat-engine;
- Heat-api-cfn: is an API compatible with AWS CloudFormation that process the requests by sending them to the heat-engine;
- Heat-engine: is the main component that provides connection with other OpenStack services (like Nova, Glance or Neutron) to deploy the final application as well as to receive events from the two other services.

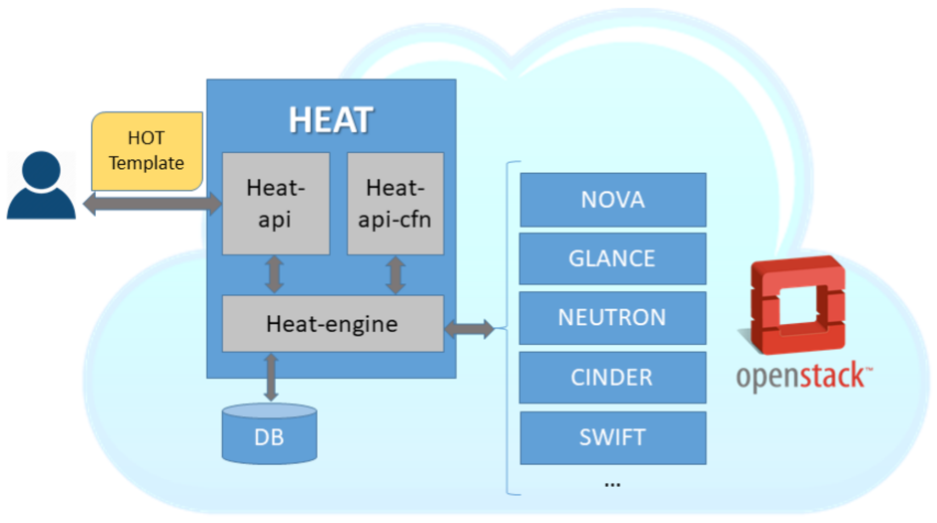


Figure 3. OpenStack Heat Architecture

3.1.2. OpenStack Magnum

OpenStack Magnum [Magnum] is technology for supporting containers in OpenStack environment. It started as an interface between Docker and OpenStack compute module (nova). OpenStack Containers Team develops it since 2014. The tool is written in Python. Currently, OpenStack Magnum supports Docker Swarm, Kubernetes, and Apache Mesos. The software is licensed under the Apache license.

Its main aim is to provide an API for container orchestration. There are two main components: API REST server and Conductor. The REST server accepts requests and outputs messages. The API REST supports 6 main objects (see Figure 4): 1) bay (cluster at newer versions) – collection of virtual machines for hosting containers, 2) bay model (cluster template at newer versions) – similar to flavor at bay level (os image, orchestration engine), 3) pod – collection of containers running on single host, 4) service – logical set of pods and access policy, 5) replication controller – manager of pods (also provides scaling and upgrade), and 6) node – nova instance.

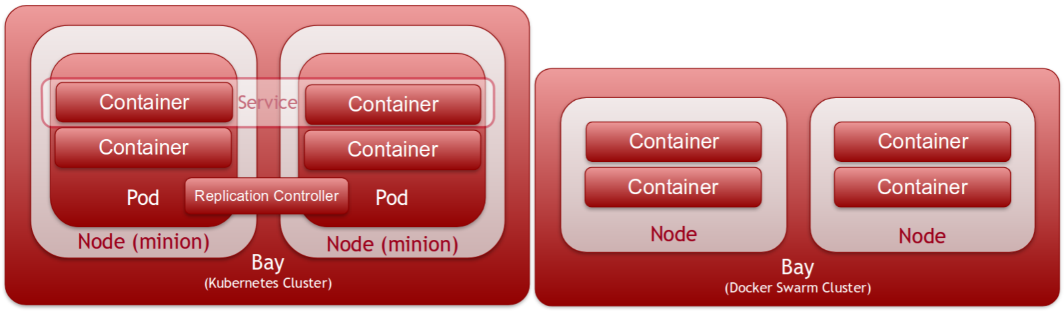


Figure 4. Magnum API resources [Otto 2015]

Conductor processes the messages and creates the outputs, which means interaction with containers and handling of container orchestration by interaction with OpenStack Heat. The multi-tenancy is provided by Keystone. The whole integration of Magnum into OpenStack environment is shown in Figure 5. It exploits: 1) Nova for instances management, 2) Neutron for configuration of a private network (inside the bay), 3) Glance for images management, and 4) Cinder for management of mounting points (inside containers).

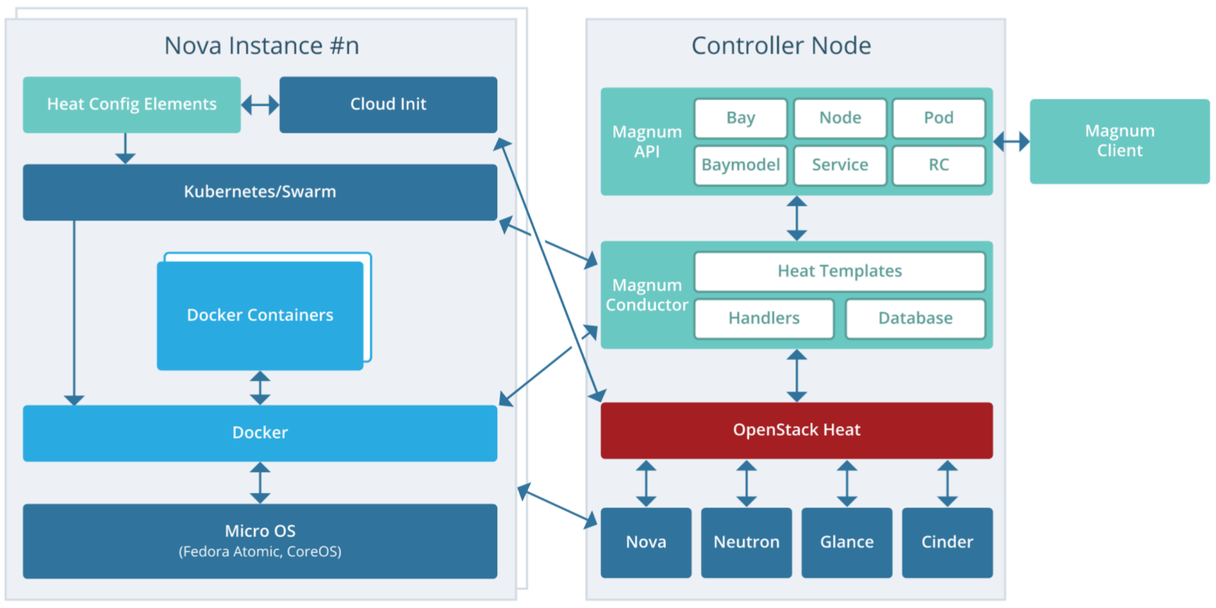


Figure 5. Magnum architecture [Cacciatore 2015]

3.1.3. OpenStack nova-lxd

OpenStack nova-lxd [nova-lxd], [nova-lxd_GitHub] is a plugin allowing managing Linux Containers (LXC) in an OpenStack cloud. Its development started in 2015. The plugin is written in Python. OpenStack nova-lxd is installed on nova servers only. It bridges nova-compute daemon with LXD, which is a daemon for LXC. It enables to use LXD as a hypervisor in OpenStack environment (alongside KVM, VMware, and Hyper-V). Thus a user is able to execute all the standard operations (like boot, reboot, stop, delete, make snapshots, do resize operations, do migration operations, and so on) on a LXC and also s/he is able to manage resources of a container (like number of CPU, size of memory, disk). The plugin implements a RESTful API on top of the LXC libraries and offers it to an OpenStack user (see Figure 6). LXC containers are managed in the same manner as KVM containers – either via Horizon or via the Nova CLI. It is released under the Apache 2.0 license.

Functionalities implemented or tested

- Life cycle control: deploy, shutdown, restart, reset, suspend and resume containers.
- Monitor hosts and containers.
- Network attachment and Floating IP association.
- Security Groups association.
- Ubuntu Xenial LXD flavored image made available.
- Image custom properties "Hypervisor_type: lxd"

Functionalities not implemented

- Native LXD Ceph support
- Live migration between compute hosts

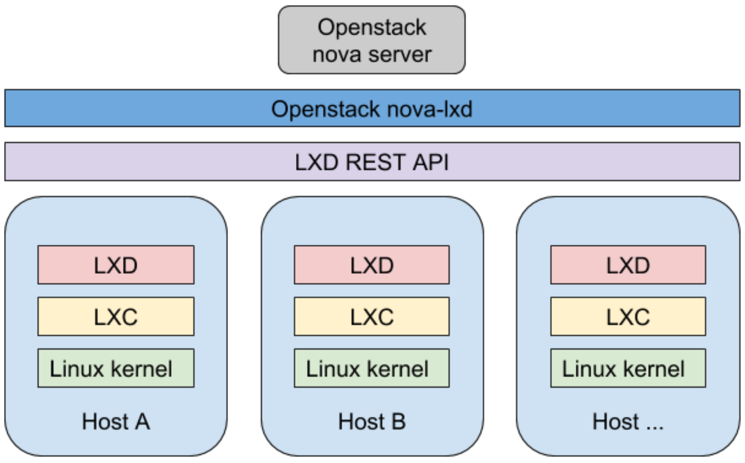


Figure 6. OpenStack nova-lxd architecture diagram. Modification of [Graber 2016]

3.2. OpenNebula

OpenNebula [OpenNebula] was first established as a research project in 2005 and the first public release of software was in March 2008, it has evolved through open-source releases and now operates as an open source project. It provides a simple but feature-rich and flexible solution for the comprehensive management of virtualized data centers to enable private, public and hybrid IaaS clouds. The last series of releases (5.4.x) has improved storage and network management and high availability.

3.2.1. OpenNebula LXDoNe

LXDoNe [LXDoNe] is an addon for OpenNebula to manage LXD Containers. It uses the pylxd API for several container tasks. Its development started in 2017. LXD is a daemon which provides a REST API to drive LXC containers, which are lightweight OS-level virtualization instances. Unlike Virtual Machines they don't share the kernel with the host and, therefore, they don't suffer from hardware emulation processing penalties.

Functionalities implemented

- Life cycle control: deploy, shutdown, restart, reset, suspend and resume containers.
- Support for Direct Attached Storage (DAS) file systems such as ext4 and btrfs.
- Support for Storage Area Networks (SAN) implemented with Ceph.
- Monitor hosts and containers.
- Limit container's resources usage: RAM and CPU.
- Support for VNC sessions.
- Deploy containers with several disks and Network Interface Cards (NICs)
- Support for dummy and VLAN network drivers.
- Full support for OpenNebula's contextualization in LXD containers (using special LXD images that will be uploaded to the market).

Functionalities not implemented

- Native LXD Ceph support: LXD's Ceph module is not cloud ready.

Installation

- Frontend setup linear and supported.
- Virtualization of the node standard for Ubuntu 16.04 (Xenial Xenus), not supported for CentOS7 (containers run only if privileged).
- Usage of pre-built image downloadable from ONE marketplace
- LXD virtualization node, bridged network and template creation in ONE supported and linear.

3.2.2. ONEDock

ONEDock [ONEDock] is a development of the INDIGO-DataCloud project, which includes a set of extensions for OpenNebula to use Docker containers as first-class entities, just as if they were lightweight Virtual Machines (VM). For that, Docker is configured to act as a hypervisor so that it behaves just as KVM or other hypervisors do in the context of OpenNebula. When OpenNebula is asked for a VM, a Docker container will be deployed instead. In the context of OpenNebula, it is managed as if it was a VM, and the user will be able to use IP addresses to access to the container. ONEDock deploys Docker containers on top of bare-metal nodes.

ONEDock provides four components that need to be integrated into the OpenNebula deployment:

- **ONEDock Datastore** that makes it possible to create a datastore that contains Docker images. It is self-managed in the sense that images are created as references that are automatically downloaded from Docker Hub;
- **ONEDock Transfer Manager** that stages the Docker images that are in a Docker datastore into the virtualization hosts;
- **ONEDock Monitoring Driver** that monitors the virtualization hosts in the context of the Docker hypervisor;
- **ONEDock Virtual Machine Manager** that carries out the tasks related to the lifecycle of the Docker containers as if they were VMs.

ONEDock seamlessly integrates the benefits of Docker containers (quick deployment, limited overhead, availability of Docker images, etc.) in a Cloud Management Platform such as OpenNebula. On the other side, it provides new features for containers that are usually reserved for VMs (e.g. enhanced IP addressing, attachment of block devices, etc.).

3.3. Kubernetes

Kubernetes is an open-source platform written in Go for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications [Kubernetes]. It can run on various platforms on local-machine, VMs on a Cloud IaaS provider, or a rack of bare metal servers. A Kubernetes cluster is made of a master node and a set of worker nodes. In a production environment, these run in a distributed setup on multiple nodes. Kubernetes has six main components that form a functioning cluster (see Figure 7): 1) API server; 2) Scheduler; 3) Controller manager; 4) kubelet; 5) kube-proxy; and 6) etcd. Each of these components can run as standard Linux processes, or they can run as Docker containers [Schroder 2017].

Kubernetes cluster consists of nodes, previously known as minions. Nodes are working machines in Kubernetes cluster. Depending on cluster nature, a node can be a virtual or physical machine. Kubernetes do not inherently create nodes. They already exist in a pool of virtual or physical machines. When such node is added into Kubernetes it is checked, whether it contains all the necessary services.

Nodes are managed by the master components running on a master node. Master components make global decisions about the cluster and include components such as kube-apiserver – horizontally scalable front-end to the Kubernetes control plane, kube-scheduler for scheduling Pods on nodes, kube-controller-manager running controllers (Node controller, Replication Controller, Endpoints

Controller and Service Account & Token Controllers). All the cluster data is stored in distributed key-value store etcd, which is also a part of the master components.

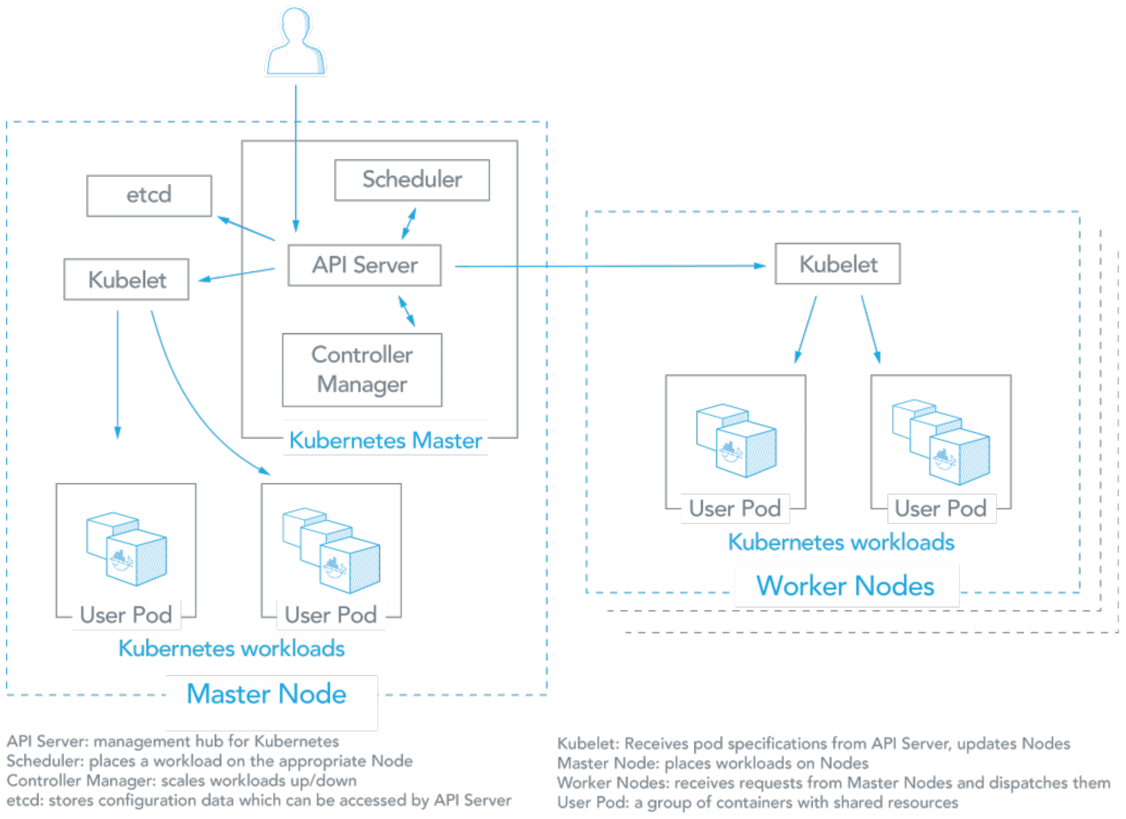

etcd
Scheduler
API Server
Kubelet
Controller Manager
Kubernetes Master
User Pod
User Pod
Kubernetes workloads
Master Node
Kubelet
User Pod
User Pod
Kubernetes workloads
Worker Nodes
API Server: management hub for Kubernetes
Scheduler: places a workload on the appropriate Node
Controller Manager: scales workloads up/down
etcd: stores configuration data which can be accessed by API Server
Kubelet: Receives pod specifications from API Server, updates Nodes
Master Node: places workloads on Nodes
Worker Nodes: receives requests from Master Nodes and dispatches them
User Pod: a group of containers with shared resources

Figure 7. Kubernetes architecture [Parthasarathy 2017]

Each node in the cluster has the services necessary to run Pods, which are the basic building blocks of Kubernetes. A Pod represents a unit of deployment; i.e., a single instance of an application on Kubernetes cluster. It is used to encapsulate a single application container or multiple tightly coupled containers working together with storage resource, unique network IP and run-time options. The most common container in Kubernetes is Docker, but there can be other runtimes integrated via Container Runtime Interface (CRI), without the need to recompile such as cri-o, rtklet, or frakti.

To scale applications horizontally in Kubernetes, Pods have to be replicated with a Controller, so there is one Pod for each replicated instance of the application.

Running a container on a laptop is relatively simple. However, connecting containers across multiple hosts, scaling them when needed, deploying applications without downtime, and service discovery among several aspects, are hard challenges. Kubernetes addresses those challenges with a set of primitives and a powerful API.

Pros

- Kubernetes is mature, production-grade container orchestration. It is purely API-driven container orchestrator, API server, scheduler, and a controller with fault-tolerance, self-discovery and scaling.

- Kubernetes is distinguished from similar container orchestration systems, such as Apache Mesos [Mesos] and Docker Swarm [DockerSwarm], by its Google heritage. Borg, the very advanced internal datacenter management system used by Google for a decade, inspired Kubernetes. Google donated Kubernetes to the Cloud Native Computing Foundation [CNCF], hosted at the Linux Foundation [LinuxFoundation] and supported by a number of big companies including Google, Cisco, Docker, IBM, and Intel. The idea is to create a reference architecture for cloud technologies that anyone can use.

Cons

- Manual installation can be complex, but flexible. There are many deployment tools for Kubernetes like kubeadm, kops, kargo, and others.

There are available several tools for monitoring Kubernetes cluster. The Web UI (Dashboard) allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself. Kubernetes is supported by Heapster monitoring platform allowing it to inspect application performance at different levels (containers, pods, services, and whole clusters). Heapster runs as a pod in the cluster and gathers usage information from all the nodes of the cluster. Collected data is pushed to a configurable backend for storage and visualization such as InfluxDB+Grafana, Google Cloud Monitoring, Kafka, Elastic search and many others. These tools differ in what kind of information they support from Heapster; i.e., monitoring metrics, events, or both. Kubernetes cluster can be created via a hosted solution on one of many cloud providers (currently counted 13 providers). Another similar option is Turnkey Cloud solution, which allows creating Kubernetes cluster on a range of Cloud IaaS providers. Turnkey Cloud solution is also available for running it on own internal, secure, cloud network via IBM Cloud Private or Kubermatic (On-Premises turnkey cloud solutions). There is kubeadm toolkit for a complete control over the cluster installation, which expects existing machines to be executed on. The type of the machines does not matter, which makes it easier to integrate kubeadm with other different systems; e.g., Ansible.

3.4. Apache Mesos + Marathon + Chronos

Apache Mesos

Apache Mesos abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to be easily built and run effectively. Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g. Hadoop, Spark, Kafka, Elasticsearch) with API's for resource management and scheduling across entire datacenter and cloud environments. Mesos aims to provide a scalable and resilient core for enabling various frameworks to efficiently share clusters. Because cluster frameworks are both highly diverse and rapidly evolving, Mesos overriding design philosophy has been to define a minimal interface that enables efficient resource sharing across frameworks, and otherwise push control of task scheduling and execution to the frameworks. Pushing control to the frameworks has two benefits. First, it allows frameworks to implement diverse approaches to various problems in the cluster (e.g., achieving data locality, dealing with faults), and to evolve these solutions independently. Second, it keeps Mesos simple and minimizes the rate of change required of the system, which makes it easier to keep Mesos scalable and robust.

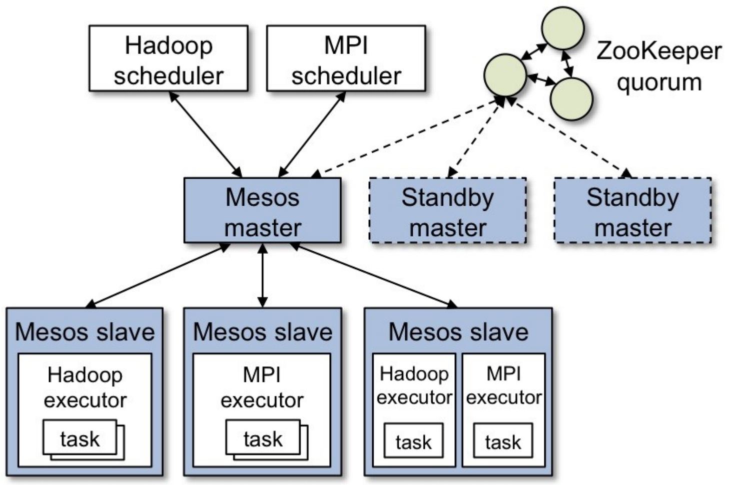


Figure 8. Mesos architecture diagram, showing two running frameworks (Hadoop and MPI) [Hindman 2011]

Mesos consists of a master process that manages slave daemons running on each cluster node, and frameworks that run tasks on these slaves. The master implements fine-grained sharing across frameworks using resource offers. Each resource offer is a list of free resources on multiple slaves. The master decides how many resources to offer to each framework according to an organizational policy, such as fair sharing or priority. To support a diverse set of inter-framework allocation policies, Mesos lets organizations define their own policies via a pluggable allocation module. Each framework running on Mesos consists of two components: a scheduler that registers with the master to be offered resources, and an executor process that is launched on slave nodes to run the framework's tasks (see Figure 8). While the master determines how many resources to offer to each framework, the framework's schedulers select which of the offered resources to use. When a framework accepts offered resources, it passes Mesos a description of the tasks it wants to launch on them. To maintain a thin interface and enable frameworks to evolve independently, Mesos does not require frameworks to specify their resource requirements or constraints. Instead, Mesos gives frameworks the ability to reject offers. A framework can reject resources that do not satisfy its constraints in order to wait for ones that do. Thus, the rejection mechanism enables frameworks to support arbitrarily complex resource constraints while keeping Mesos simple and scalable.

Example of resource offer

Figure 9 below shows an example of how a framework gets scheduled to run a task:

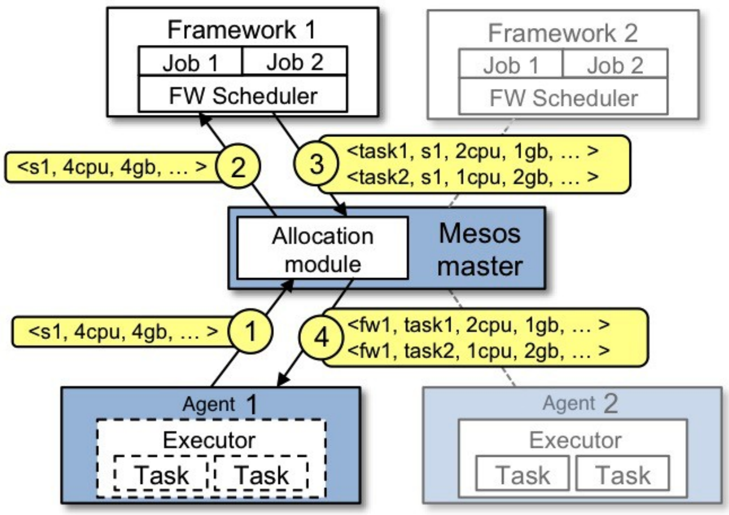


Figure 9. Resource offer example [Hindman 2011]

1. Agent 1 reports to the master that it has 4 CPUs and 4 GB of memory free. The master then invokes the allocation policy module, which tells it that framework 1 should be offered all available resources.
2. The master sends a resource offer describing what is available on agent 1 to framework 1.
3. The framework's scheduler replies to the master with information about two tasks to run on the agent, using <2 CPUs, 1 GB RAM> for the first task, and <1 CPUs, 2 GB RAM> for the second task.
4. Finally, the master sends the tasks to the agent, which allocates appropriate resources to the framework's executor, which in turn launches the two tasks (depicted with dotted-line borders in the figure). Because 1 CPU and 1 GB of RAM are still unallocated, the allocation module may now offer them to framework 2.

In addition, this resource offer process repeats when tasks finish and new resources become free.

Pros

- Support for Composing, Docker, Mesos virtualization containers
- Running several frameworks (even the same frameworks only different versions) on the same cluster
- Proven to scale, used at Apple with 75.000 nodes
- Extremely flexible, it is currently used in production by many enterprises and there are many Mesos frameworks available that can fit different needs
- Relatively old, easier to find production stories and best practices to use it

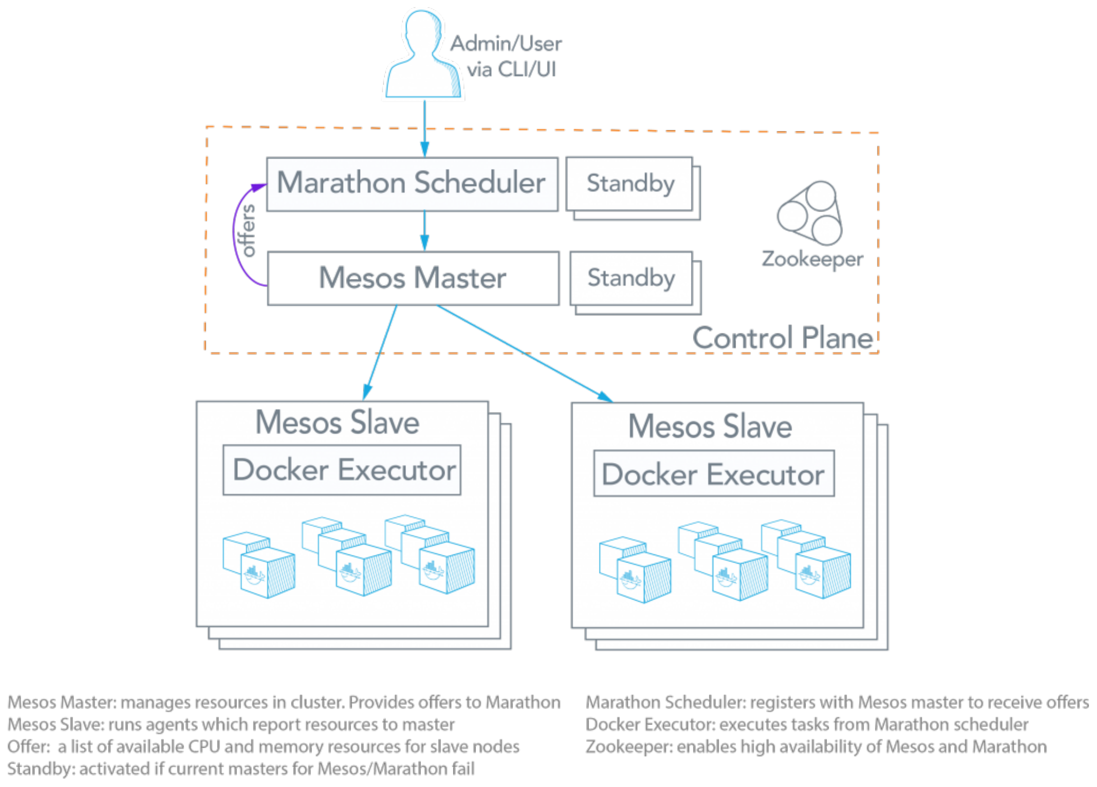
Cons

- Little bit old

- Too many languages. To have a running Mesos stack, many components are involved: Mesos (C++), Marathon (Scala), Mesos-DNS (Golang), etc. It is not so common to find developers that can be proficient in so many languages at the same time

Marathon

Marathon is a framework (or meta framework) that can launch applications and other frameworks. Marathon can also serve as a container orchestration platform, which can provide scaling and self-healing for containerized workloads. Figure 10 below shows the architecture of Mesos + Marathon.



Mesos Master: manages resources in cluster. Provides offers to Marathon
Mesos Slave: runs agents which report resources to master
Offer: a list of available CPU and memory resources for slave nodes
Standby: activated if current masters for Mesos/Marathon fail

Marathon Scheduler: registers with Mesos master to receive offers
Docker Executor: executes tasks from Marathon scheduler
Zookeeper: enables high availability of Mesos and Marathon

Figure 10. Marathon architecture

Chronos

Chronos is a replacement for cron. A distributed and fault-tolerant scheduler runs on top of Apache Mesos that can be used for job orchestration. It supports custom Mesos executors as well as the default command executor. Thus by default, Chronos executes sh (on most systems bash) scripts. Chronos can be used to interact with systems such as Hadoop (incl. EMR), even if the Mesos slaves on which execution happens do not have Hadoop installed. Chronos is also natively able to schedule jobs that run inside Docker containers. Chronos has a number of advantages over regular cron. It allows you to schedule your jobs using ISO8601 repeating interval notation, which enables more flexibility in job scheduling. Chronos also supports the definition of jobs triggered by the completion of other jobs. It supports arbitrarily long dependency chains.

Chronos is a Mesos scheduler for running schedule and dependency based jobs. Scheduled jobs are configured with ISO8601-based schedules with repeating intervals. Typically, a job is scheduled to run indefinitely, such as once per day or per hour. Dependent jobs may have multiple parents, and will be triggered once all parents have been successfully invoked at least once since the last invocation of the dependent job.

Internally, the Chronos scheduler main loop is quite simple (see Figure 11). The pattern is as follows:

1. Chronos reads all job states from the state store (ZooKeeper);
2. Jobs are registered within the scheduler and loaded into the job graph for tracking dependencies;
3. Jobs are separated into a list of those which should be run at the current time (based on the clock of the host machine), and those which should not;
4. Jobs in the list of jobs to run are queued, and will be launched as soon as a sufficient offer becomes available;
5. Chronos will sleep until the next job is scheduled to run, and begin again from step 1.

Furthermore, a dependent job will be queued for execution once all parents have successfully completed at least once since the last time it ran. After the dependent job runs, the cycle resets.

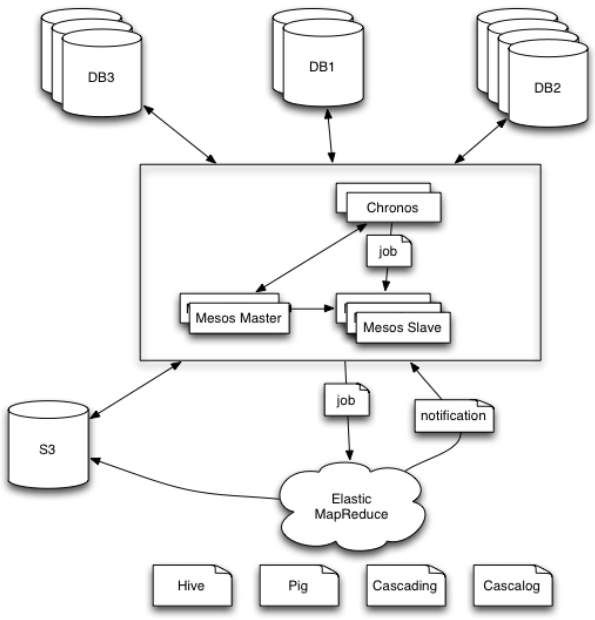


Figure 11. Chronos architecture

4. Support for Accelerated computing

4.1. Accelerators and Deep Learning

Nowadays, accelerators are designed to accelerate NNs and other Ml/DL algorithms [AI Accelerator]. They are often many core designs and generally focus on low-precision arithmetic. Early attempts of using accelerators to improve artificial intelligence applications are dated to early 1990s. Digital Signal Processors (DSP) were used to accelerate OCR software, FPGA-based accelerators were used for NNs and heterogeneous multiprocessors with specialized processors for different tasks (e.g., cell microprocessor). With the advent of general-purpose computing on GPU accelerators (GPGPU), the GPU accelerators quickly became popular also for ML/DL applications [Cano 2017] [Sze 2017]. The reason is, mainly, that the mathematical basis of NNs and image manipulation are similar.

4.1.1. Types of accelerators

A list of DL Accelerator technology providers [DLAccList] is already quite long and includes global technology companies like Google, Intel, IBM, Microsoft or NVIDIA along with many worldwide startups. NVIDIA GPU-based accelerators are among the most popular ones for both training and inference. Cloud providers provide virtualized or bare-metal servers equipped with NVIDIA Tesla accelerators and NVIDIA put significant effort to providing users with GPU-optimised containers including DL frameworks. The two recent intensive-mentioned NVIDIA architectures from 2016 and 2018 are (respectively) NVIDIA Pascal and NVIDIA Volta.

NVIDIA Pascal is the first NVIDIA architecture to integrate the revolutionary NVIDIA NVLink high-speed bidirectional interconnect [NVIDIAPascal]. This technology is designed to scale applications across multiple GPUs, delivering a 5× acceleration in interconnect bandwidth.

NVIDIA included optimizations for DL frameworks already into their Pascal architecture. Half-precision, 16-bit floating point instructions and 8-bit integer instructions along with NVLink allowed to reach significant speed-up in both NN training and DL inference.

NVIDIA Volta, which is the current architecture, goes even further [NVIDIAVolta]. It contains special ASIC “Tensor cores” designed to speed-up convolutions and matrix operations along with doubled throughput of NVLink. Equipped with 640 Tensor Cores, Volta delivers over 100 Teraflops per second (TFLOPS) of DL performance, over a 5× increase compared to prior generation NVIDIA Pascal architecture.

NVIDIA Volta uses next generation NVIDIA NVLink high-speed interconnect technology. This delivers 2× the throughput, compared to the previous generation of NVLink. This enables more advanced model and data parallel approaches for strong scaling to achieve the absolute highest application performance.

NVIDIA Volta also comes with Volta-optimized CUDA and NVIDIA Deep Learning SDK libraries like cuDNN, NCCL, and TensorRT, for existing frameworks and applications.

Although GPU accelerators are the mainstream solution for DL and they provide big speed-up comparing to CPU, some applications like mobile phones or robotic systems still need better power efficiency, mainly for DL inference. This leads to the development of AI accelerators based on application-specific integrated circuits (ASIC). The main drawback of this approach is their lack of adaptability. The most notable accelerators in this direction are:

- Google Tensor processing unit (TPU) [GoogleTPU2],
- Intel Nervana Neural Network Processor [Kloss 2017],
- Microsoft BrainWave project [Brainware],
- IBM TrueNorth neuromorphic chip [Feldman 2016],
- Accelerators in mobile phones e.g. Apple A11 Bionic chip [Dilger 2017], Huawei Kirin 970 [Boxall 2017], Samsung Exynos 9810 [SamsungExynos], MediaTek Helio P60 [MediaTek].

There are also many more other vendors as in the Table 2.

Vendor type	Vendor names
IC Vendors	Intel, Qualcomm, NVIDIA, Samsung, AMD, Apple, Xilinx, IBM, STMicroelectronics, NXP, MediaTek, HiSilicon, Rockchip
Tech Giants & HPC Vendors	Google, Amazon_AWS, Microsoft, Aliyun, Tencent Cloud, Baidu, Baidu Cloud, HUAWEI Cloud, Fujitsu
IP Vendors	ARM, Synopsys, Imagination, CEVA, Cadence, VeriSilicon
Startups in China	Cambricon, Horizon Robotics, DeePhi, Bitmain, Chipintelli, Thinkforce
Startups Worldwide	Cerebras, Wave Computing, Graphcore, PEZY, KnuEdge, Tenstorrent, ThinCI, Koniku, Adapteva, Knowm, Mythic, Kalray, BrainChip, AImotive, DeepScale, Leepmind, Krtkl, NovuMind, REM, TERADEEP, DEEP VISION, Groq, KAIST DNPU, Kneron, Vathys, Esperanto Technologies

Table 2. Growing list of accelerator vendors [Shan 2018]

4.1.2. Using accelerators in Deep Learning

CPUs are designed for more general computing workloads. GPUs in contrast are less flexible, however GPUs are designed to compute in parallel the same instructions (the extended SIMD paradigm). DNNs are structured in a very uniform manner such that at each layer of the network thousands of identical artificial neurons perform the same computation. Therefore, the structure of a DNN fits quite well with the kinds of computation that a GPU can efficiently perform [Perez 2015]. In comparison with CPUs, GPUs have many more resources and faster bandwidth to memory, which leads to faster computational speed. This issue is extremely important because training of DNNs can range from days to weeks.

Accelerated libraries

- CUDA [CUDA] is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning and graph analytics. NVIDIA CUDA Toolkit [CudaToolkit] provides a development environment for creating high performance GPU-accelerated applications.
- The NVIDIA CUDA Deep Neural Network library (cuDNN) [cuDNN] is a GPU-accelerated library of primitives for DNNs. The cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. It allows DL users to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. The cuDNN accelerates widely used deep learning frameworks, including Caffe2, MATLAB, CNTK, TensorFlow, Theano, PyTorch, etc.
- OpenCL™ (Open Computing Language) [OpenCL] is the open, royalty-free standard for cross-platform; parallel programming of diverse processors found in personal computers, servers, mobile devices and embedded platforms. OpenCL greatly improves the speed and responsiveness of a wide spectrum of applications in numerous market categories including gaming and entertainment titles, scientific and medical software, professional creative tools, vision processing, and neural network training and inferencing.
- Intel Math Kernel Library (Intel MKL) [MKL] is a library of optimized math routines for science, engineering, and financial applications. Core math functions include BLAS, LAPACK, ScaLAPACK, sparse solvers, fast Fourier transforms, and vector math. The routines in MKL are hand-optimized specifically for Intel processors.

Other libraries that can be listed here include the NVIDIA Deep Learning SDK, which is a part of the NVIDIA toolkit. It provides powerful tools and libraries for designing and deploying GPU-accelerated DL applications including libraries for DL primitives, inference, video analytics, linear algebra, sparse matrices, and multi-GPU communications. Except for cuDNN, other SDK libraries are, e.g., Deep Learning Inference Engine (TensorRT), Deep Learning for Video Analytics (DeepStream SDK), Linear Algebra (cuBLAS), Sparse Matrix Operations (cuSPARSE) and Multi-GPU Communication (NCCL).

Deep Learning frameworks and libraries

Many popular machine learning (ML) and deep learning (DL) frameworks and libraries already offer the possibility to use accelerators to speed up the learning process (Table 3). These libraries also use optimised accelerated libraries e.g. CUDA (cuDNN), OpenMP, OpenCL, etc. to improve the performance even further. The main feature of the many-core accelerators is massively parallel architecture allowing them to speed up computations that involve matrix-based operations. The GPGPU interest can be found in many other large-scale life simulation packages with dynamic progress developments.

DL frameworks and libraries	Creator	Description
Tensorflow	Google	TensorFlow [Tensorflow] and TensorFlowLite [TensorflowLite] from Google Brain is an open source software library for numerical computation using data flow graphs. TensorFlow is designed for large-scale distributed training and inference. The distributed Tensorflow architecture consists of the distributed master, worker services with kernel implementations including mathematical, array manipulation, control flow, and state management.
Keras	Google, Microsoft	Keras [Keras] is a minimalist Python library for DL that can run on top of TensorFlow, CNTK, Theano, beta version with MXNet and announced Deeplearning4j.
CNTK	Microsoft	Microsoft Cognitve Toolkit CNTK implements efficient DNNs training for speech, image, handwriting and text data [CNTK]. Its network is specified as a symbolic graph of vector operations, such as matrix add/multiply or convolution with building blocks (operations). CNTK supports FFNNs, CNNs and RNNs.
Caffe	BAIR	Caffe [Caffe] is a DL framework developed by Yangqing Jia at Berkeley Artificial Intelligence Research (BAIR) for image processing.
Caffe2		Caffe2 is a Caffe's lightweight, modular, and scalable DL framework focused on mobile developed by Yangqing Jia at and his team at Facebook [Caffe2]. It aims to provide an easy and straightforward way to experiment with DL and leverage community contributions of new models and algorithms. Caffe2 is used at production level at Facebook while development is done in PyTorch
Torch	R. Collobert, K. Kavukcuoglu, C. Farabet	Torch is a scientific computing framework with wide support for ML algorithms based on the Lua programming language [Torch]. It is aimed on large-scale learning (speech, image, and video applications), and allows supervised learning, unsupervised learning, reinforced learning, NNs, optimization, graphical models, image processing.

PyTorch	A. Paszke, S. Gross, S. Chintala, G. Chanan	PyTorch is a Python package for building DNNs and performing complex tensor computations [PyTorch]. While Torch uses Lua, PyTorch leverages the rising popularity of Python interface of the same optimized C libraries as Torch's. PyTorch uses a technique called reverse-mode auto-differentiation (i.e. dynamic computational graph), which allows to change the way a network behaves with small effort.
MXNet mxnet	APACHE SOFTWARE FOUNDATION	Apache MXNet is a DL framework that allows mixing symbolic and imperative programming to maximize efficiency and productivity [MXNet]. MXNet contains a dynamic dependency scheduler that automatically parallelises both symbolic and imperative operations on the fly.
Theano theano	Université de Montréal	Theano [Theano] is a compiler for mathematical expressions in Python to transform structures into very efficient code using NumPy and efficient native libraries like BLAS and native code to run as fast as possible on CPUs or GPUs. Theano supports extensions for multi-GPU data parallelism and has a distributed framework for training models. Theano is actively maintained – but no longer developed – by LISA group, University of Montreal, Quebec.
Chainer	Preferred Networks	Chainer is a Python-based DL framework aiming at flexibility [Chainer]. It provides automatic differentiation APIs based on the define-by-run approach, i.e., dynamic computational graphs as well as object-oriented high-level APIs to build and train NNs.
Deeplearning4J DL4J	skymind	Deeplearning4j or DL4J is an modern open-source, distributed DL library implemented in Java (JVM) aimed to the industrial Java development ecosystem and Big data processing [DL4J].
H2O / Deep Water H_2O	H_2O.ai	H2O, Sparkling Water and Deep Water are developed by H2O.ai. They are Hadoop compatible frameworks for DL over Big Data as well as for Big Data predictive analytics [H2O]. Deep Water is H2O DL with native implementation of DL models for GPU-optimized backends TensorFlow, MXNet, and Caffe.

Table 3. Deep Learning frameworks and libraries

The dependencies among accelerators/HPC, accelerated libraries and DL frameworks and libraries are depicted in Figure 12.

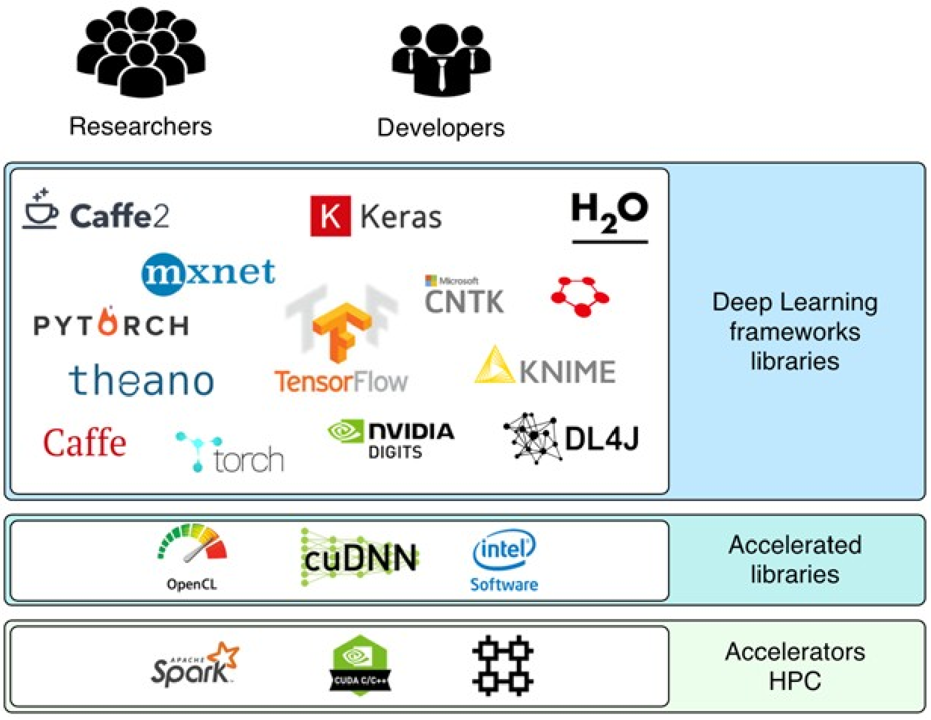


Figure 12. Deep Learning frameworks and libraries with accelerated support

More details about ML/NN/DL frameworks and libraries are available in the Deliverable D6.1 – State-of-the-art Deep Learning, Neural Networks and Machine Learning frameworks and libraries.

4.2. Support for accelerators at hypervisor/container level

In the past, the supports for accelerators at hypervisor/container level were not adequate. Technologies like PCI passthrough or device mapping are dependent on host configurations, which makes generalization and system-independent implementation difficult. For examples, PCI passthrough technique of hypervisors needs exact information about location of the PCI slot of the device; therefore, administrators must configure hypervisors for each hardware manually. Similarly, device-mapping techniques in container technologies allows containers to have access to devices, however, the software drivers for the devices must be matched between the host and the containers, preventing implementation of generic containers with supporting accelerators.

More generic solutions were implemented only recently. For example the first official version (version 1.0) of nvidia-docker was released in Jan 2017 and version 2.0 in December 2017. Similarly LXD 3.0 (released in April 2018) is the first version of LXD with GPU plugins support. The details of each approach and their comparisons are provided in the following subsections.

4.2.1. PCI passthrough and SR-IOV

PCI passthrough [PCIPASS] is a method to attach a PCI device directly to a virtual machine. This technique can be used in cases where a physical device does not need to be shared between multiple virtual guests or when it is not possible to share it. In this case, dedicating the device to one guest by giving it direct access to the device on the PCI bus means practically native performance even in a virtualized environment – a significantly faster access than in the case of device emulation. Another advantage of PCI passthrough is that the device itself does not need to be known to the hypervisor, since it does not need to be emulated. However, PCI passthrough presents some major disadvantages as well. The device cannot be shared between different machines (as it is attached exclusively to a VM) and in order to enable live migration of the guest which has direct access to a PCI device, the device's status needs to be migrated too. For this, PCI hotplug support in the device is necessary. PCI passthrough requires both hardware support (VT-d in Intel chipsets, IOMMU in AMD chipsets) and hypervisor support. The guest operating system must also support PCI passthrough to be able to fully use its capabilities.

SR-IOV stands for Single Root Input/Output Virtualization, and it is a specification that allows to expose PCI functions in an isolated way, by means of the creation of virtual functions that are similar in functionality to the functions offered by the physical resource. Using the SR-IOV technology, a PCI device will expose several virtual functions; therefore, a single physical PCI device may be assigned to several virtual guests in a transparent and configurable way. The usage of PCI passthrough and SR-IOV is the most advantageous situation, since it is possible to attach a device (like a network card) into one or several virtual machines, since the sharing is done at the SR-IOV level.

With PCI passthrough it is potentially possible to attach any PCI devices to a VM. However, some hardware vendors have imposed restrictions in their device drivers, that refuse to work if they detect that they are being loaded inside a virtualized environment. This is the case for some consumer focused NVIDIA GPU cards, whose drivers refuse to work under any virtualization platform (see [NVIDIA-PATCHES] for instance), even if the card works perfectly inside the VM. For those circumstances, it is needed to spoof the CPUID opcode so that the systems appears to be non-virtualized.

4.2.2. GPU-specific virtualization methods

Since GPU’s shaders are excellent parallel processing units, they are being used in an extensive range of applications where matrix calculations are needed – among them the already mentioned deep learning. Advent of specialized General Purpose GPUs (GPGPU) has led to the necessity to devise methods of their virtualization in cloud environments, so there can be a flexible allocation of pool of GPGPUs to a pool of virtual computation nodes. To this end, every major producer of GPUs has designed a standard for virtualization of its GPU resources. These standards rely usually on a combination of hardware and software support and apportion the GPU resources – memory, shaders, video encoders and decoders – either as physical slices or as time slots. The standards are NVIDIA GRID vGPU [VGPU], AMD MxGPU [Wong 2016] and Intel GVT [GVT].

The following table summarizes current support for these GPU virtualization technologies in the main hypervisor platforms:

	NVIDIA PCI-passthrough	NVIDIA GRID	AMD MxGPU	Intel GVT
KVM	partial/coming [NVGFORUM] [NVGDOCS]	partial/coming [NVGFORUM] [NVGDOCS]	available [GIM] [AMDDRIV]	available [GVTLINUX]
Xen	partial/coming [NVGFORUM] [NVGDOCS]	partial/coming [NVGFORUM] [NVGDOCS]	available [GIM] [AMDDRIV]	available [GVTLINUX]
VMWare	available [XENAPP]	available [NVPASS]	available [AMDDRIV]	not available

Table 4. Current support for GPU virtualization technologies in the main hypervisor platforms

NVIDIA GRID vGPU

- Physical sharing of RAM (slices)
- Time sharing of shaders (slots)
- Time sharing of video encoders and decoders (slots)
- Supports CUDA since GRID version 5.0
- Software-based management of virtualization
    - Software component for the hypervisor
    - Virtual GPU driver for the guest OS

AMD MxGPU

- Physical sharing of RAM (slices)
- Physical sharing of shaders (slices)
- No virtualization of video encoding and decoding
- Support for OpenCL
- hardware-based management of virtualization, based on SR-IOV standard

INTEL GVT

- Different modes of operation
    - GVT-d for dedicated access to the GPU by one host
    - GVT-g for time-shared access to the GPU's shaders and encoders/decoders, and slice-shared access to the RAM
    - GVT-s for sharing via a virtual driver

- Support for OpenCL
- Software-based management of GPU virtualization

4.2.3.Device mapping (passthrough) for LXD/Docker

Mellanox docker-passthrough-plugin

This network plugin allows having direct/passthrough access between the native Ethernet or InfiniBand networking device and the Docker container(s). It provides two modes of operation:

- **SR-IOV mode** – In this mode given netdev interface is used as PCIe physical function to define a network. All container instances will get one PCIe VF based network device when they are started. This mode uses PCIe SRIOV capability of the network devices. SR-IOV mode provides native access to the actual PCIe based networking device without any overheads of virtual devices. With this mode, every container can get dedicated NIC Tx and Rx queues to send or receive application data without any contention to other containers.
- **Passthrough mode** – In this mode a given netdev interface is mapped to a container. Which means that there is one network device per network, and therefore every container gets one network. In some cases, there would be need to map bonded device directly without additional layer and without consuming any extra MAC address. In such cases, this passthrough plugin driver will be equally useful.

In some sense, both modes are similar to passthrough mode of KVM or similar virtualization technologies.

With these plugin-based interfaces, there is no limitation on IP address subnet for netdevice of container and netdevice of host. Any container can have any IP address, same or different subnet as that of the host or other containers.

In the future, more settings for each such netdevice and network will be added.

In SR-IOV mode, plugin driver takes care to enable/disable sriov, assigning VF based network device to container during starting a container. This will reduce administrative overheads in dealing with sriov enablement [MELLANOX].

4.2.4. nvidia-docker runtime

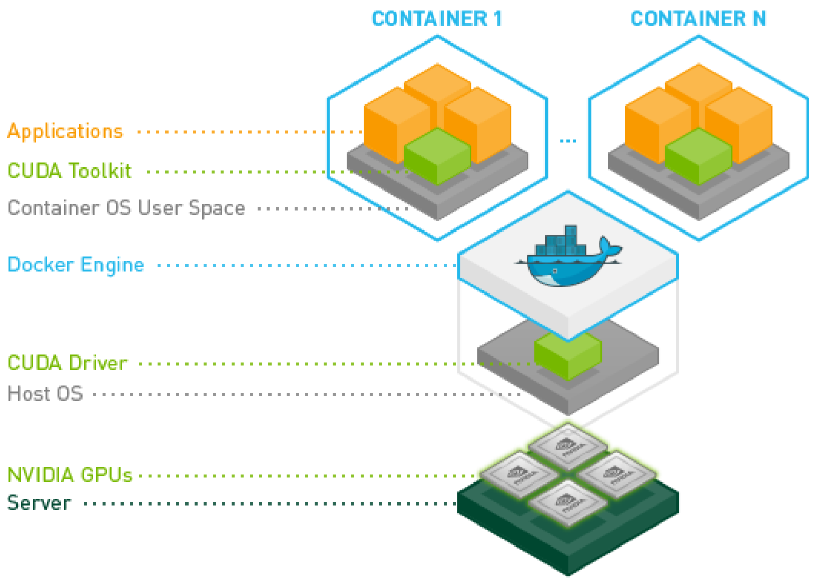


Figure 13. nvidia-docker runtime

NVIDIA proposes a solution to preserve portability advantage of Linux Containers, in this case for Docker Containers, despite the specificities and constraints of using specialized hardware.

This kind of devices requires the use of drivers. Those drivers, within a traditional Docker solution, would need to be fully installed inside the container and match the underlying device. Thus, the version of the host driver has to exactly match driver version installed in the container. This makes the Docker images unsharable and the image must be built locally for each host.

nvidia-docker acts as a runtime that allows the use of Docker images that are independent from target host (see Figure 13). Containerizing GPU applications provides several benefits [nvidia-docker], among them:

- Ease of deployment
- Isolation of individual devices
- Run across heterogeneous driver/toolkit environments
- Requires only the NVIDIA driver to be installed on the host
- Facilitate collaboration: reproducible builds, reproducible performance, reproducible results.

4.2.5. Comparison of approaches for supporting accelerators

Approach	PCI-passthrough	GPU virtualization	Device mapping	GPU runtime
Availability	Generally available on new hardware	Limited availability, only certain types of newest Intel, AMD and NVIDIA cards; support for some virtualization platforms still not sufficiently stable	Generally supported by most container technologies	Available implementation with different level of maturity
Usability	Manual configuration required, hardware dependence, difficult to generalize	If available can be used to share GPGPU resources across a virtualized environment, allocate shader units to nodes	Potential compatibility issues by conflicting version between software drivers on host and containers	Easy to use, automatic detection and including software drivers from host to container, removing problems of version conflicting of software driver in device mapping
Limitation	Cannot suspend/migrate VM, stability and security issues	Different GPU manufacturers offer different capabilities, placing limitations on usability of heterogeneous environments	Matching version of software drivers between container and host	NVIDIA driver installation needs to live on the same partition as the volume directory of the Docker plugin
Implementation	Hypervisor	KVM, VMWare	Containers	LXD 3.0, nvidia-docker

Table 5. Comparison of approaches for supporting accelerators

4.3. Support for accelerators at cloud middleware level

4.3.1. OpenStack

OpenStack provides support for GPU accelerators via PCI passthrough. On the computing node, hardware properties of GPU accelerators must be identified and included in a whitelist of nova configuration. An alias will be assigned to the accelerator on master (controller) node, then the alias is added to the properties of flavors that require accelerators. The process of configuration for supporting accelerators is manual.

One of the largest limitations of supporting accelerators via PCI passthrough in OpenStack is inability to suspend/migrate virtual machines that use accelerators. In the case of a security update or other maintaining activities that require rebooting computing nodes, site administrators must stop the running virtual machines and restart them again.

The most recent release of OpenStack (Queens, released on Feb 28, 2018) provide preliminary experimental support for GPU virtualization. The hypervisor and GPU cards must support the GPU virtualization.

4.3.2. OpenNebula

OpenNebula can keep track of PCI devices and assign them to virtual machines since Version 5.0 (2016). Still, many issues may arise in other layers (hardware, hypervisor, drivers within the virtual machine) and OpenNebula can provide only limited assistance in overcoming them.

The issue that is most difficult to tackle relates to device isolation. The GPU accelerator must be made available to the hosted virtual machine over PCI, while other PCI devices must remain, naturally, under the control of the host. This requires hardware support. Not only must the hardware support VT-d and IOMMU, but the GPU device must also be installed in a dedicated IOMMU group to achieve proper isolation. Otherwise, overly benevolent rights would have to be granted to OpenNebula users, allowing them to control all PCI hardware, which, of course, is not possible. Certainly not in multi-tenant environments.

4.3.3. Kubernetes

Kubernetes includes experimental support for managing NVIDIA GPUs spread across nodes [KubernetesGPU]. From 1.8 onward, the recommended way to consume GPUs is to use a device plugin framework. The device plugin framework enables vendors to advertise their resources to the kubelet without changing Kubernetes core code. This is done by implementing a device plugin and deploying it manually or as a DaemonSet. There are currently two device plugin implementations for NVIDIA GPUs:

- Official NVIDIA GPU device plugin
- NVIDIA GPU device plugin used by GKE/GCE

Official NVIDIA GPU device plugin

This device plugin is deployed in Kubernetes as a DaemonSet. It requires each GPU node to have NVIDIA drivers installed, Docker with installed nvidia-docker utility and default Docker runtime set as nvidia-runtime. The Docker has to be at least v1.12. Since this version, there is a support for custom container runtimes, so there can be the nvidia-runtime used. Node GPU can be shared among different containers. Multiple GPUs can be isolated between containers. There is support for NVIDIA Optimus technology as well.

Known issues and limitations:

- CUDA Multi Process Service is not supported at the moment (v1.9);

- GPU-accelerated X server inside the container is not supported;
- limiting GPU resources (e.g., bandwidth, memory, CUDA cores) per container is not supported;
- the device plugin does not support macOS, Microsoft Windows and Tegra (arm64) platforms; PowerPC64 (ppc64) is still on the way to be supported;
- exclusive access for a GPU is not supported, but it can be enforced through Kubernetes or through the driver level by setting the compute mode of the GPU.
- Docker versions must follow driver versions

4.3.4. Apache Mesos

Apache Mesos supports NVIDIA GPUs from the version 1.0.0 and the minimum required NVIDIA driver version is 340.29. To enable the GPU support in Apache Mesos cluster is straightforward. First users need to configure agent nodes in order to expose GPUs to the Apache Mesos master nodes and second enable the framework GPU capability so that the Apache Mesos master includes the GPUs in the resource offer sent to the framework. Mesos exposes GPUs as a simple SCALAR resource in the same way it always has for CPUs, memory, and disk. However, unlike CPUs, memory, and disk, only whole numbers of GPUs can be selected. At the time of this writing, NVIDIA GPU support is only available for tasks launched through the Mesos containerizer (i.e., no support exists for launching GPU-capable tasks through the Docker containerizer). That said, the Mesos containerizer now supports running docker images natively, so this limitation should not affect most users. Moreover, Apache Mesos mimics the support provided by nvidia-docker to automatically mount the proper NVIDIA drivers and tools directly into docker container. This means that anybody can easily test GPU-enabled docker containers locally and deploy them to Apache Mesos with the assurance that they will work without modification.

Agent Flags

The following isolation flags are required to enable NVIDIA GPU support on an agent.

```
--isolation="filesystem/linux,cgroups/devices,gpu/NVIDIA"
```

The filesystem/linux flag tells the agent to use Linux-specific commands to prepare the root filesystem and volumes (e.g., persistent volumes) for containers that require them. Specifically, it relies on Linux mount namespaces to prevent the mounts of a container from being propagated to the host mount table. In the case of GPUs, we require this flag to properly mount certain NVIDIA binaries (e.g., NVIDIA-smi) and libraries (e.g., libNVIDIA-ml.so) into a container when necessary.

The cgroups/devices flag tells the agent to restrict access to a specific set of devices for each task that it launches (i.e., a subset of all devices listed in /dev). When used in conjunction with the gpu/NVIDIA flag, the cgroups/devices flag allows us to grant / revoke access to specific GPUs on a per-task basis.

By default, all GPUs on an agent are automatically discovered and sent to the Mesos master as part of its resource offer. However, it may sometimes be necessary to restrict access to only a subset of the GPUs available on an agent. This is useful, for example, if a specific GPU device should be excluded because an unwanted NVIDIA graphics card is listed alongside a more powerful set of GPUs. When this is required, the following additional agent flags can be used to accomplish this:

```
--NVIDIA_gpu_devices="<list_of_gpu_ids>" --resources="gpus:<num_gpus>"
```

For the `--NVIDIA_gpu_devices` flag, a comma separated list of GPUs is needed, as determined by running NVIDIA-smi on the host where the agent is to be launched.

Framework Capabilities

Once the agent is launched with the flags above, GPU resources will be advertised to the Mesos master alongside all of the traditional resources such as CPUs, memory, and disk. However, the master will only forward offers that contain GPUs to frameworks that have explicitly enabled the GPU_RESOURCES framework capability. The choice to make frameworks explicitly opt-in to this GPU_RESOURCES capability was to keep legacy frameworks from accidentally consuming non-GPU resources on GPU-capable machines (and thus preventing your GPU jobs from running). It is not that big a deal if all nodes have GPUs, but in a mixed-node environment, it can be a big problem.

External Dependencies

Any host running a Mesos agent with NVIDIA GPU support MUST have a valid NVIDIA kernel driver installed. It is also highly recommended to install the corresponding user-level libraries and tools available as part of the NVIDIA CUDA toolkit. Many jobs that use NVIDIA GPUs rely on CUDA and not including it will severely limit the type of GPU-aware jobs you can run on Mesos. The minimum supported version of CUDA is 6.5.

5. Interaction with HPC resources using PaaS approach

5.1. Specifics of HPC systems

HPC architecture

Typical HPC systems consist of at least one head node and a set of working nodes connected to each other via a very fast network (e.g. Infiniband, Ethernet, OmniPath). Worker nodes (WN) have their own IP address pool and are not available from outside of the cluster. Depending on site policies, the outbound communication can be allowed or not. The typical way is to allow all outbound communication because many modern applications require it. Some new HPC configurations allow to provide higher isolation level by defining more strict communication rules, e.g., limit communication only to WNs assigned to a job, or to specific ports or to specific protocols or IPs. This can be defined by site administrators, or by job description.

Access to HPC

A typical access to HPC resources is to login to the head node, prepare and submit jobs from there. A short compilation of the program code and pre- or post-processing of submitted jobs might be permitted. Ssh or gsissh is used for accessing head nodes. In some cases, also graphical access to a head node is allowed via, e.g., X2Go, NX client, or VNC client. Access to worker nodes depends on the site policy and configurations. Usually it is possible to access WNs via local job scheduler only, but on some sites access via ssh is allowed to WNs on which users jobs are currently run.

It is possible to set up additional head nodes dedicated to a project or to a group of users. E.g., scientific portals allow users to define jobs visually in the portal and submit to HPC resources. Users do not need to care about the submission process, since the portal server that submits jobs directly to the job management systems, possibly by using API, does it.

Grid middleware allows to access HPC resources without using HPC head nodes. Grid middleware running on HPC sites accepts jobs from grid brokers and submits directly to the local job management system. Users can use grid job submission clients to run their jobs on different sites with different management systems. Usually grid middleware is used from grid user interface (grid head node) or from GUIs (graphical user interfaces). Examples of middleware include gLite WMS with glite-wms-client or QCG Broker with qcg-client.

Access to storage

There are different kinds of storage systems on HPC systems and various ways of accessing this storage:

- The typical way is to have a storage system shared on head nodes and worker nodes. Users can upload and download data by accessing the head node with sftp, scp or gsisftp. Two kinds of storage can be provided: low performance long-term storage, e.g., home directory filesystem, and high performance short-term storage, e.g., scratch directory filesystem. All data in long-term storage are regularly backed up while short-term storage is typically not backed up or archived. All data intensively used by computations should be copied from home directory filesystem to scratch directory filesystem at the beginning of the computation and moved back to home at the end. Some sites do not share home file system therefore users must stage in and stage out their input and output data.
- There are some specialized storage systems used by specific projects. E.g., WLCG (Worldwide LHC Computing Grid) experiments ATLAS and ALICE keep their data in DPM system [LCGDM]. Users access the storage system directly to upload/download data. Worker nodes also can access this storage and get necessary data.
- Another way of providing storage for HPC is to use logical storage, e.g., LFC (Logical File Catalog) or OneData. Users do not need to use head node to prepare their data, but interact with the storage itself. Logical storage hides the complexity of underlying storage and can provide replication to efficiently serve data. This is a user-friendly way of accessing data but may degrade performance of job processing in some conditions.

Registration

Users must be authorized to run jobs on HPC resources. Usually users need to apply for an HPC account individually on each HPC system or group of systems. This relies strongly on sites policies and procedures can be different in each site (way of application, necessary documents, assigning name id, expiration time of an account, etc.), e.g., registration may depend on the amount of asked resources (for example CPU hours). Usually permission to use HPC is granted for a specific time and with given limit of available resources. Users must periodically report the results of their computations. In the case of scientific research this is usually done by providing sites with a list of publications in which the usage of HPC infrastructure was referenced. It can be required by the end of usage period to provide a written status report.

It is possible to register to many HPC at the same time, what is common on site levels, or infrastructure levels. E.g., central PRACE portal is used to apply for computation project on HPC machines from PRACE infrastructure or PLGrid portal is used to apply for account on polish HPC machines. In both cases, information about users is added to local LDAP systems, replicated to specific machines and used in authorization process.

Another approach is used in EGI gLite middleware. Because of large number of users and sites, it was not practical to create a physical unix account for each user. Instead, pool accounts are used. They have generic names (e.g., atlas001 to atlas299) and are assigned to users when needed and cleaned when no more needed. Users register to VOs (Virtual Organizations), which authorize them to use resources dedicated to the respective VO. Certificates are used for authentication. A site authorizes a VO as a whole, while maintaining control on individual users via white and black lists.

Another possible way is to use shared accounts, e.g., to run jobs submitted by a generic project user via portal. One HPC account is used to submit jobs on behalf of users registered in the portal. This approach causes problems with job separation, job monitoring and accounting, which must be solved by the portal’s server-side software. A project representative negotiates with HPC owner for available resources and a way of settling accounts (money, publications, etc.)

Queue names

In order to submit jobs to local resource management systems (LRMS) users need to specify the queue name and job parameters. Different queues can have different limits, permissions and priorities. The names are local to HPC systems and usually are different on different sites; therefore, users must know the queue name before submitting jobs. In addition, queue names and parameters can change in time. Configuration of queues can be found by running command line programs or can be read from site documentations.

gLite middleware provides an information system, which provides information about the resources and services sites provide. LDAP with GLUE schema is used to provide information about available resources and their status. Site administrators configure queue names for each VO statically.

QCQ middleware hides queue names from users, which just submit jobs to sites. Queues configuration in kept in a Broker configurations file and needs to be updated when site configurations change.

Some job management systems support scheduling queues. This is a queue that takes a user’s job and schedules it to destination a queue depending on job parameters, e.g., time, memory size, number of cores, tags, etc. This way users do not need to care about the queues configuration, but just need to properly describe their jobs. This solution is available in the NQE management system and SLURM Workload Manager.

Queue limits

Each queue can have limits for job resources, especially time, number of cores and memory size. In addition, there are limits for the total number of jobs or total number of given jobs in the queue. Different sites have different policies for setting queue limits. The most typical limitation is the running time of jobs. Some sites have strict limits to 24 hours while most sites limit duration to no more than 7 days. Extra-long queues of more than 7 days may be available as well but have other strict limits, for instance, only one job can run at a time. Time limit can be connected with queue priority; e.g., shorter jobs can have higher priority. Depending on site policies, the priority or fairshare can be set up to promote shorter jobs, bigger jobs, or less memory bound jobs.

When users submit jobs with parameters that exceed queue limits then the submission fails.

Queue waiting time

The time jobs spend in a queue depends on many factors including: user priority, number of user jobs, total number of jobs, fairshare configuration, resources available for the queue.

It is very difficult to predict the job waiting time. Some job managers can show expected job start time, but it is very inaccurate and can be useless if a site supports urgent computing (highest priority jobs that can even preempt normal jobs).

Preemption is a form of a guarantee that the job will start (almost) immediately. In normal HPC operation, preemptive jobs are rather exceptional and are not used. Better form of providing time guarantees for jobs is to use reservations.

Reservations can be created for a user or a group of users, for a specific time or periodically. Any other users cannot use reserved resources.

Access to software

Most user jobs use applications available on HPC systems and installed by site administrators. Access to the software can be different on each HPC site, especially the path to executable and available versions. One solution to this problem is the use of modules. But even then the way of starting applications can be different, e.g., module paths can be different or can have different names. E.g., intel compiler can be available after “module add icc” or “module add tools/intel”. Some heterogeneous sites require that users are aware of architecture (“module load gaussian/amd” vs “module load gaussian/intel”) while other sites hide this from users and prepare software in the same paths but optimized for the given node.

Some projects or middleware try to unify application namespaces. E.g., PLGrid requires that all Polish HPC sites must have the same application in the same mode path, e.g. /plgrid/apps/gaussian/

EGI infrastructure used VO tags and VO software directory where VO-manager could install required software. This unifies software across sites, but requires significant effort.

QCG tries to unify access to applications by providing application templates. A user fills in a form specific for an application and QCG Broker knows how to run the application on destination node. This is configured in QCG-Computing service on each site.

The potential solution for the software problems is to use containerized jobs and software provided by user. While it solves some problems, it introduces others, especially connected to computation performance. Applications installed by administrators are optimized for the given site, especially for interconnection or CPU architecture. Using a generic version of applications instead of the onsite compiled or tuned is usually wasting of resources. This concerns also libraries, especially GPGPU, MPI and computational libraries.

Running parallel jobs

Sites can have different ways to start parallel MPI jobs. At least two different approaches were found. One approach is to use “`mpirun -np N`” from job script. The number of requested MPI processes must be given twice – in job parameters and in the mpirun commandline. The second approach is to use srun from job script and srun takes the number of mpi processes from job contexts. Therefore, job scripts should be prepared for each site individually.

Conclusions

Access to HPC is non-standardized and each site can have different configuration, rules and policies. It is not easy to automate the access to HPC resources from higher levels (e.g., PaaS environments).

5.2. Using containers in HPC

As described in 5.1 users have rather limited rights in HPC environment and typically use software pre-installed by site administrators. This can however be extended to almost any application if HPC system allows submitting containerized jobs.

Container technology, also referred to as ‚operating system virtualization‘, provides a way to pack an application with all necessary libraries or system dependencies such that it runs isolated from other applications, the underlying Linux Operating System and environment (Sec 2.2). Obvious advantages for users are the ability to run in HPC environment any custom application and mobility of application, i.e., deployment of an application homogeneously across different HPC clusters. The former also means that users can easily update the software to the latest available version, which may generally improve output value of their application. Pre-built containers of various popular software frameworks are available publicly via, e.g., Docker Hub [DockerHub], therefore allowing to fastly assess these frameworks and their features. Possible downsides of containerized applications on HPC include: a) not always an obvious way for a container to access specialized hardware and libraries, e.g. GPU, Infiniband, MPI; b) mobility supposes generalization for various HPC systems, i.e. less optimization, which may result in a partially degraded performance; c) possible security issues.

Section 2.2 reviewed major container tools available and being developed at the time of writing. The most popular one, Docker [Docker], cannot be actually used in a satisfactory way in HPC systems because of security concerns [Gomes 2017]. When Docker starts a container; processes running within the container are executed with the root id, potentially making it possible to gain privileged access in the host machine. Docker also requires Linux kernel version 3.10+ [DockerFAQ] that is not the case for RHEL6 or CentOS6 still largely used in HPC systems.

Among the tools allowing running containerized applications in HPC environment the two which get attention in recent years are **Singularity** [Singularity] and **udocker** [uDocker] (see Sec. 2.2). The main difference between them for an HPC user is that Singularity has to be installed by a system administrator while udocker is entirely a user tool. The latter allows to profit from most recent developments once a new version is available. As an example, at the time of this writing, Singularity version available in EPEL repository for RHEL7 [SingularityInEPEL] is 2.2.1, while most recent Singularity version from sources is 2.4.5 [Singularity]. Often due to HPC policies, system administrators prefer installation of software from official repositories rather than building from sources. An HPC user therefore depends on HPC administrator and on the repository maintainer to get the most recent version. As in the case of Singularity, most recent versions (2.3+) allow pulling of Docker images from the Docker Hub and significantly simplify access to GPUs. Both features do not exist in 2.2.1 and as such they are not available to many HPC users. Udocker instead does not suffer from this since a regular user can install the most recent version under his/her account. Apart from this, both Singularity (2.3+) and udocker can pull images from Docker Hub and provide access to GPUs. In tests performed on our local GPU cluster, we did not notice any statistically significant difference in terms of computational speed between tasks running on GPU by means of either Singularity or udocker.

5.3. Local schedulers

Job schedulers were designed to schedule large-scale scientific modeling and simulations on supercomputers and became a key component of the today's scalable computing infrastructures. The role of job schedulers is to orchestrate all of the work executed on the computing infrastructure that affect the effectiveness of the system. In the recent years, job workloads have diversified from long-running, synchronously parallel simulations to include short-duration; independently parallel high performance jobs, data analysis jobs and consequently different schedulers have been developed to address both job workload and computing system heterogeneity.

A list, and related features, of the most common job schedulers adopted in the scientific public-funded computing infrastructures is here presented in Table 6:

- The Portable Batch System (PBS) Scheduler [PBS] is directly descended from NQS [NQS], the first batch scheduler developed under NASA funding. PBS is a fully featured job scheduler that includes a separate queue manager, resource manager, and scheduler, and the Maui scheduler is often used in place of the native PBS scheduler. PBS continues to be developed by Altair Engineering (PBSPro) and Adaptive Computing (TORQUE/Maui/Moab Cluster Suite), which both offer open-source and commercial versions. Recently, scalability challenges have been addressed, the support to individual GPU scheduling has been added, and with the adoption of the Moab HPC Suite (the open source job scheduler that succeed the Maui Cluster scheduler), full support of array scheduling policies, dynamic priorities, extensive reservations, and fairshare capabilities has been introduced.
- Grid Engine is a full-featured, very flexible scheduler originally developed and released under the name CODINE in 1993 by Genias Software [Gentzsch 1994], later acquired and matured by Sun Microsystems and Oracle, and currently offered commercially by Univa. There are also several open-source versions, including Son of Grid Engine and Open Grid Scheduler, though further development of these offerings seems to be waning.
- IBM Platform Load Sharing Facility (LSF) is a full-featured and high performing scheduler that is very intuitive to configure and use. It was based on research and development of the Utopia job scheduler at the University of Toronto [Zhou 1993]. OpenLAVA [OpenLAVA] is an open-source derivative of LSF that has reasonable feature parity.
- HT Condor (High Throughput Condor) continues to be developed by Prof. Myron Livny's team at the University of Wisconsin [Thain 2005]. HT Condor was designed to cycle-scavenge from distributed heterogeneous desktop computers to managed clusters. It implements a single queue with many resource- and job-distinguishing options, including prioritization and fair execution. It works especially well for many smaller, single-process jobs with execution on a diverse set of computers, clusters, and supercomputers.
- The Simple Linux Utility for Resource Management (Slurm) [Yoo 2003] is an extremely scalable, full-featured scheduler with a modern multi-threaded core scheduler and a very high performing plug-in module architecture. The plug-in module architecture makes it highly configurable for a wide variety of workloads, network architectures, queuing policies, scheduling policies, etc. The Quadrics RMS scheduler inspired it. To ease transition from other schedulers, translation interfaces are provided. Slurm began development in the early 2000s at Lawrence Livermore National Laboratory to address scalability issues in job schedulers. It is entirely open source, SchedMD provides consulting services and community development leadership.

Feature	PBS	Grid Engine	LSF	HT Condor	SLURM
Parallel/array jobs	both	both	both	both	both
Cost/licensing	Commercial, Open source	Commercial, Open source	Commercial, Open source	Open source	Open source
OS support	Linux	Linux	Linux	Linux	Linux
Scheduler type	Centralized	Centralized	Centralized	Centralized	Centralized
Scalability	up to 80K	10K+	10K+ (1K+ OpenLava)	up to 40K per scheduler	100K+
GPU support	Yes (limited in the Open Source)	Yes	Yes	Yes	Yes

Table 6. List of the most common job scheduler adopted in scalable computing infrastructures and their related features

5.4. PaaS Interaction and interfaces

Exposing HPC resources to the end-users through the Platform-as-a-Service layer aims to lower barriers in accessing HPC environments.

In the last years great efforts focused on the adoption of cloud computing for running HPC workloads: nowadays almost all the cloud providers are able to provide virtual instances with accelerators and low latency interconnection network. Several benefits can come from this integration, which facilitates both workload management and interaction with remote resources.

Compared to traditional HPC environments, in cloud environments users can 1) quickly adjust their resource pools via a mechanism known as elasticity, 2) optimize resource utilization, for example using spot instances. These features help adding more flexibility to HPC applications.

Cloud technology can simplify the use of HPC resources, in particular for non-IT specialized users with no expertise in system administration and configuration of complex computing environments.

In fact, an emerging approach both in public and private clouds is to create “HPC as a service” platforms where HPC applications are executed in the cloud without requiring users to have any understanding of the underlying cloud infrastructure: the provisioning of the resources and their configuration is completely automated. An example is AWS CfnCluster that allows to quickly deploy an HPC cluster on AWS with pre-installed open source batch schedulers and MPI libraries. The user can interact with the HPC environment through the web-based EnginFrame portal that gives the user access to his applications (both batch and interactive), data, and jobs.

Another important factor is contributing to change the way HPC systems are used: the development of new applications, like deep learning, and the increasingly wide adoption of containers to support a DevOps approach for developing and running applications. These new trends are leading HPC system admins and architects to embrace new approaches: 1) run both containerized and non-containerized workloads on existing HPC clusters; 2) run HPC workloads on local or cloud-resident Kubernetes/Mesos clusters. Concerning the first approach, the support for containerized jobs and applications is being added in the main Workload Managers; moreover, the use of udocker greatly simplifies the execution of docker containers in any HPC cluster since this tool allows to run the containers in user space without requiring root privileges.

The second approach is more innovative and it is likely to become more common as the availability of new application frameworks (Artificial intelligence, Big Data, Deep-Learning, etc.) designed and implemented to run natively on these platforms (Kubernetes, Mesos) increases. Anyway, the two approaches are not exclusive and can co-exist. Indeed, the co-existence of the two environments, the traditional on-premise HPC cluster and the Kubernetes/Mesos cluster, allows to run mixed workloads and to select the most suited environment depending on the characteristics of the workload itself.

For example, tightly coupled applications can benefit from better performance running on HPC clusters (with high-performance network); on the other hand, embarrassingly parallel applications can be deployed on cloud or on container orchestration platforms like Kubernetes or Mesos, exploiting built-in features like elasticity and resilience.

As a final remark, it is worth highlighting that, from a business point of view, the hybrid cloud model is currently gaining increasing importance since it ensures sustainability for several companies with HPC workloads. With this model, it is possible to leverage the existing computing infrastructure and, depending on the peak demands, parts of the workload can be moved temporarily to the cloud.

From the analysis of the presented approaches, it is evident that exploiting hybrid/mixed environments can be a successful approach and in this case the role of a smart orchestrator/broker is crucial to select the best environment for running the users' workload.

6. Initial implementation plan

The motivation of DEEP Hybrid DataCloud project is the need to support intensive computing techniques that require specialized HPC hardware, like GPUs or low-latency interconnects, to explore very large datasets. A Hybrid Cloud approach enables the access to such resources that are not easily reachable by researchers at the scale needed in the current EU e-infrastructure. The project is structured into six different work packages, covering Networking Activities (NA) devoted to the coordination, communication and community liaison; Service Activities (SA) focused on the provisioning of services and resources for the execution of the data analysis challenges; and Joint Research Activities (JRAs), dealing with the development of new components and technologies to support data analysis. The interaction between the different work packages is present in Figure 14.

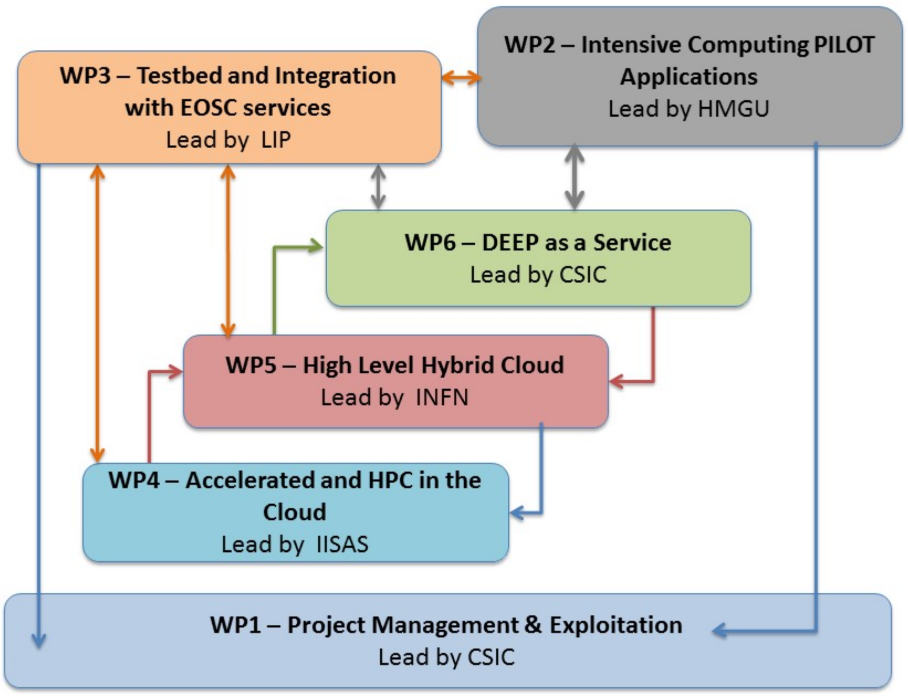

WP3 – Testbed and Integration with EOSC services
Lead by LIP
WP2 – Intensive Computing PILOT Applications
Lead by HMGU
WP6 – DEEP as a Service
Lead by CSIC
WP5 – High Level Hybrid Cloud
Lead by INFN
WP4 – Accelerated and HPC in the Cloud
Lead by IISAS
WP1 – Project Management & Exploitation
Lead by CSIC

Figure 14. DEEP-HybridDataCloud Work Packages and their relations

WP4 "Accelerated and High Performance Computing in the Cloud" is lying in the lowest layer among JRA workpackages. The research and development activities done in this workpackage are close to the hardware and infrastructure, addressing the gaps that currently exist in the support of accelerators, specialized hardware and HPC systems in general in order to exploit the full potential of computing performance provided by the hardware. On upper level, WP5 "High Level Hybrid Cloud solutions" will take care of the provisioning of the platform exploiting the outcomes from WP4 in a hybrid approach, delivering an execution platform, ensuring that applications can be spawned across several cloud infrastructures. The highest level among JRA workpackages is WP6 "DEEP as a Service" that will provide solutions to ensure that scientists have an easy way to deploy and execute their intensive compute applications based on containers (from NA2/WP2) that will be executed in an hybrid cloud platform (JRA2/WP5), exploiting the specialized hardware that their application requires (JRA1/WP4).

6.1. Task structure and coordination activities

The main research activity in WP4 is to work close to the hardware and infrastructure, addressing the gaps that currently exist in the support of accelerators, specialized hardware and HPC systems in general. It will ensure that bare-metal like performance is delivered through the adopted solution, and that the resources can be shared in multi-tenancy environments. Proper interfaces will be exposed to the upper layers, from the visualization to the cloud management framework to the platform one. On top of that, high-level access to HPC resources will be investigated, providing seamless access and data sharing from Cloud infrastructures.

This WP can be logically divided into three tasks:

- Task 4.1 – Bare-metal like performance: The aim of this task is to develop on existing middleware to interact as closely as possible with bare metal resources. That will include minimizing the overhead of software virtualization layer using para-virtualization or container technologies, providing seamless access to performance critical hardware resources (accelerators, Infiniband) from virtual machines/containers (in cooperation with Task 4.2) and ensure support from cloud middleware and PaaS level.
- Task 4.2 – Extended support for accelerated computing: This task is dedicated to provide extended support for accelerators through all software layers, to hide implementation details and provide uniform access to accelerators from PaaS and application levels. It will work closely with T4.1 for supporting accelerators in container and cloud middleware, and T4.3 for supporting accelerators on HPC platforms.
- Task 4.3 – Interaction with HPC resources with PaaS approach: The task will investigate how to integrate the HPC-like environments with High Level services, propose a way to execute hybrid, containerized applications within batch-oriented environment accessing GPU and specialized low-latency networks.

6.2. Coordination with other WPs

6.2.1. Coordination with WP 5

Activities in WP4 will aim to improve virtualization techniques in order to minimize the overhead and achieve bare-metal performances; moreover, efforts will focus on making hardware accelerators and low-latency networks first-class citizens in the cloud computing landscape. Complementarily the main goal of WP5 is to provide a uniform and transparent access to the IaaS-level resources, including HPC resources, with a hybrid approach. Therefore, WP5 and WP4 will work in close synergy in order to maximize the benefits from leveraging WP4 developments at the PaaS level: WP4 will implement the full support of containers and specialized devices in the virtualization layer and will expose to the PaaS the needed interfaces to provision the requested resources (VMs or containers). WP5 will build an abstraction layer on top of WP4 interfaces ensuring federated access to the underlying IaaS environments.

6.2.2. Coordination with WP3

The activities taking place in all tasks of WP4 have a direct connection with the two tasks of WP3. The development work occurring in WP4 is supported by services provided or organized by WP3 such as the repository for artifacts, the continuous integration system (CI), source code repositories, issue trackers and user support, etc.

The SQA, the Software release and maintenance management has been defined by WP3 after input and discussions with the other JRAs.

The organization of the testbed in WP3 is accomplished with the collaboration of WP4, specifically regarding the type of resources and how they should be integrated such as: IaaS Cloud Management Frameworks (OpenStack and OpenNebula), of GPUs, HPC clusters, docker container management frameworks (Mesos, Kubernetes).

6.3. Initial implementation plan

According to the task structure, the implementation plan can be divided into three main parts:

- Improving support for containers in cloud middleware: to use container technologies as replacement of hypervisors in traditional cloud middleware and also as the main application delivery for HPC platforms.
- Improving support for accelerators in cloud middleware: that will include improving the support for accessing accelerators at container layers, also improving the support of accelerators in configurations/drivers of cloud middleware layers.
- Interaction with HPC resources: either accessing HPC platforms via traditional batch systems or managing HPC platforms via cloud middleware.

The schedule of the tasks is in the following Table 7:

		2017		2018												2019												2020			
Tasks		11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4
Task 4.1	**Bare-metal like performance**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	Improving udocker	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						
	Improving support for nova-lxd in OpenStack	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Task 4.2	**Extended support for accelerated computing**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	Improving support GPU in udocker	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						
	Improving support Low Latency Interconnects in udocker													🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	Adding support for GPU in nova-lxd							🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
	Improving support GPU in Mesos/Kubernetes	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						
Task 6.2	**Interaction with HPC resources**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

	Support for accessing HPC from WP5	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥						
	Switching cloud management system							🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Milestones and deliverables																															
D4.1	Assessment of available technologies for supporting accelerators and HPC, initial design and implementation plan.						🟩																								
D4.2	First implementation of software platform for accessing accelerators and HPC.												🟩																		
D4.3	Final implementation of software platform for accessing accelerators and HPC																														🟩
MS4.1	Assessment of available technologies carried out			🟦																											
MS4.2	Initial design and implementation plan for software platform created						🟦																								
MS4.3	First implementation of the software platform.												🟦																		
MS4.4	First implementation of the software platform.																														🟦

Table 7. Implementation timeplan

6.3.1. Improving support for containers in cloud middleware

nova-lxd

December 2017 LXD 2.21 was released announcing as one of its new features "*a new Infiniband device type which supports physical passthrough of Infiniband devices as well as SR-IOV allocated cards*" [LXD 2.21 Release]. Since release 2.5 LXD also has support for GPU hotplug [LXD 2.5 Release]. So far, there is no evidence that this feature can be explored from OpenStack's nova-lxd perspective. Therefore, we will work towards developing further nova-lxd OpenStack component to better explore these underlying capabilities.

udocker

Aiming to maximize application's performance and portability throughout (software- and hardware-wise) heterogeneous systems, specially having in mind advanced computing systems, udocker will be further developed to improve support for GPU and other specialized devices like Infiniband. This task will mostly be centered in software development; however, it will also require testing and benchmarking on High Performance, Grid and Cloud computing resources.

6.3.2. Improving support for accelerators in cloud middleware

Improving support for accessing accelerators at hypervisor/container level

This task is focused on improving support for accelerators at hypervisor/container level. That will include automatic detection of accelerators, their software drivers and configuration of hypervisors/containers performed accordingly, simplifying the access to accelerators and enabling generalization at higher software layer (cloud) without dependency on hardware details. The work will see cooperation with 6.3.1 for supporting GPU in udocker and LXD.

Improving support for accessing accelerators at higher layers of cloud stack

This task is focused on improving support for accelerators at higher software layers like Mesos, Kubernetes and OpenStack. That will include extending software drivers of cloud middleware for supporting accelerators (nova-lxd), or to improve deployment strategy (nova-scheduler, Mesos Chronos). In addition, it will cooperate with WP5 for configuration and deployment of a GPU-enabled PaaS layer.

6.3.3. Interaction with HPC resources

As indicated in section 5, HPC model is different from Cloud model and it is necessary to work out a solution that allows efficient use of HPC resources by high level services. There are two possible approaches to making HPCS resources available to PaaS services:

- access HPC management system and submit batch jobs,
- allow HPC nodes to be managed by Cloud management system.

These approaches are not exclusive and none of them can be chosen as the best solution, they strongly depend on local HPC conditions and politics. In the short run, it is not possible to force HPC centers to change their resources management system and switch to more cloud based model, therefore all services should adapt to the specifics of HPC and to implement necessary intermediate layer for interaction with HPC management system. However for some HPC centers in the longer term it can be a suitable solution to switch to cloud based resource provisioning and follow a cluster-on-demand approach. Therefore, implementation plan for WP4.3 covers both approaches.

Support WP5 in accessing HPC resources

During the first phase, the activities will focus will be on supporting WP5 in accessing HPC resources. An interface will be prepared to access HPC resources together with its implementation for various HPC systems. This task will be done in strong cooperation with WP5 and implementation will follow requirements coming from WP5.

Switching to cloud management system.

As part of the project integration testbed, in WP3, PSNC will dedicate some resources to prepare a small cluster managed by a cloud middleware to test the cluster on demand approach. The tests can take into account both open source (e.g., Kubernetes) or commercial software (e.g., Huawei Fusion), depending on the local conditions in HPC centers.

Specific points of interest, which will be dealt with during the implementation of the project:

- separation of work environments in HPC cluster (each job/workload should have strictly separated environment to not interfere with other work);
- sharing of resources between HPC and cloud environment (especially file and object stores, integration of sync&share systems with HPC);
- management of templates of working environments (e.g., containers based templates for various classes of execution environment with support for creation, update and optimization);
- management of authentication of authorization information. Specific implementation depends on requirement from WP5 and local HPC conditions. The focus will be on using IAM as an intermediate module between HPC and PaaS services.

6.4. Risk assessments

The potential risks, their impacts and mitigation strategies are provided in Table 8.

Risk	Impact	Mitigation
Dealing with requirements from specific use cases may require certain specific skills and/or domain knowledge	Medium	Keep continuous interactions with other responsible WP representatives. Testing/validation phases could start from their requirements. If there is no sufficient technical detail, the additional communications will be established immediately.
Partners do not deliver their contributions on-time	Low	Regular meetings will be carried out among the partners, coordinated by the WP leader, with sufficient technical content to track the development progress and introduce corrective countermeasures when needed.
Limited resources of specialized hardware during development	Medium	The consortium has adequate technical capacity for increasing resources allocated to the project in case of need. The usage of specialized resources will be monitored globally within the project to identify the need.

Table 8. Risk assessments

7. Conclusion

In this document the state of the art of technologies for supporting bare-metal, accelerators and HPC in cloud has been analyzed. Based on the analysis, an initial implementation plan for improving the support of accelerators and HPC in DEEP-HybridDataCloud has been proposed.

The container technologies have been rapidly improving in recent years and have proved themselves as viable alternative to application delivery and even full virtualization for cloud computing. The technologies have gained wide supports, from traditional Cloud management frameworks to container-centric management environment and also have strong positions on HPC systems. For what concerns support for accelerators, recent developments in GPU runtime support like nvidia-docker have greatly improved the usability, portability and stability of container technologies.

Therefore the use of container technologies will be the approach chosen in DEEP-HybridDataCloud project. The DEEP-HybridDataCloud has selected two implementations to be further improved: nova-lxd, as a replacement of full Cloud hypervisors, and udocker, as container technology to be used in the case of HPC platforms. The first release of the implementations is planed at Month 12 of DEEP-HybridDataCloud project (Milestone MS4.3) and will be reported in Deliverable D4.2 “First implementation of software platform for accessing accelerators and HPC”.

HPC computing model is different from Cloud computing model in many aspects: architecture, software installation, queuing system, user access and so on. Therefore it is necessary to work out a solution that allows to efficiently use HPC resources by high level services. The work on integration with an HPC platform will be tightly coordinated with WP5 addressing the PaaS layer and the first implementation will be released at Month 12 and reported in D4.2 together with other tasks.

8. List of Figures

9. List of tables

10. Acronyms

API	Application Programming Interface

ASIC	Application-Specific Integrated Circuit
AWS	Amazon Web Services
CNCF	Cloud Native Computing Foundation
CNTK	(Microsoft) Cognitive Toolkit
CRI	Container Runtime Interface
cuDNN	CUDA Deep Neural Network
DAS	Direct Attached Storage
DL	Deep Learning
DNN	Deep Neural Networks
FGPA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
GVT	(Intel) Intel Graphics Device
HPV	High-Performance Virtualization
IC	Integrated Circuit
IOMMU	Input–Output Memory Management Unit
IP	Semiconductor Intellectual Property Core e.g. IP vendor
LFC	Logical File Catalog
LSF	Load Sharing Facility
LXC	Linux Containers
LXD	open source container management extension for LXC
MKL	(Intel) Math Kernel Library
ML	Machine Learning
MPI	Message Passing Interface
NIC	Network Interface Cards
NN	Neural Network
PaaS	Platform as a Service
PBS	Portable Batch System
PCI	Peripheral Component Interconnect

PF	Physical Function
REST	REpresentational State Transfer
SAN	Storage Area Networks
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SLURM	Simple Linux Utility for Resource Management
SQA	Software Quality Assurance
SR-IOV	Single Root I/O Virtualization
TPU	(Google) Tensor Processing Unit
VF	Virtual Function
VNC	Virtual Network Computing
VT-d	Intel Virtualization Technology for Directed I/O
WLCG	Worldwide LHC Computing Grid

11. References and links

11.1. References

[Boxall 2017] Boxall, A.: Huawei Kirin 970: Everything you need to know, https://www.digitaltrends.com/mobile/huawei-kirin-970-ai-news/, accessed Sep 2017

[Cacciatore 2015] Cacciatore, K., Czarkowski, P., Dake, S., Garbutt, J., Hemphill, B., Jainschigg, J., Moruga, A., Otto, A., Peters, C. and Whitaker, B.E., 2015. Exploring Opportunities: Containers and OpenStack. OpenStack White Paper, 19

[Cano 2017] Cano, A.: 2017, A survey on graphic processing unit computing for large scale data mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery

[Dilger 2017] Dilger, D.E.: Inside iPhone 8: Apple's A11 Bionic introduces 5 new custom silicon engines, http://appleinsider.com/articles/17/09/23/inside-iphone-8-apples-a11-bionic-introduces-5-new-custom-silicon-engines, accessed Sep 2017

[Feldman 2016] Feldman, M.: IBM Finds Killer App for TrueNorth Neuromorphic Chip,https://www.top500.org/news/ibm-finds-killer-app-for-truenorth-neuromorphic-chip/, accessed Sep 2016

[Gentzsch 1994] Gentzsch, W.: 1994. Codine, Computing in Distributed Networked Environments, User's Guide and Reference Manual. Genias Software GmbH, Erzgebirgstr. 2B, D-93073 Neutraubling, Germany

[Gomes 2017] Gomes, J., Campos, I.: Researchers Advance User-Level Container Solution for HPC, https://www.hpcwire.com/2017/12/18/researchers-advance-user-level-container-solution-hpc/, accessed Mar 2018

[Graber 2016] Graber, S.: On the way to safe containers, Linux Security Summit 2016 Toronto, Canada, https://events.static.linuxfound.org/sites/events/files/slides/Linux%20Security%20Summit%202016-%20On%20the%20way%20to%20safe%20containers_0.pdf

[Hindman 2011] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A., Katz, R., Shenker, S., Stoica, I.: 2011, NSDI 2011, Mesos, A Platform for Fine-Grained Resource Sharing in the Data Center

[Kloss 2017] Kloss, C.: Intel® Nervana™ Neural Network Processor: Architecture Update, https://ai.intel.com/intel-nervana-neural-network-processor-architecture-update/, accessed Dec 2017

[Otto 2015] Adrian Otto: Magnum making containers a first class resource in OpenStack, https://www.OpenStack.org/assets/vancouver-summit/slidedecks/Adrian-Otto-Magnum-Making-Containers-a-First-Class-Resource-in-OpenStack.pdf#page=1&zoom=auto,-54,540, accessed Feb 2018

[Parthasarathy 2017] Parthasarathy, A., June 2017, Kubernetes vs Docker Swarm, https://platform9.com/blog/kubernetes-docker-swarm-compared/

[Perez 2015] Perez C.E.: Why are GPUs well-suited to deep learning? https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning, accessed Apr 2018

[Sato 2017] Sato, K.: An in-depth look at Google's first Tensor Processing Unit (TPU), May 2017, https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[Shan 2018] Shan, T.: A list of ICs and IPs for AI, Machine Learning and Deep Learning, Jan 2018, https://github.com/basicmi/Deep-Learning-Processor-List

[Sze 2017] Sze, V., Chen, Y.H., Yang, T.J. and Emer, J.: 2017. Efficient processing of deep neural networks: A tutorial and survey. arXiv preprint arXiv:1703.09039

[Schroder 2017] Schroder C: What Makes Up a Kubernetes Cluster? Apr. 2017, https://www.linux.com/news/learn/chapter/intro-to-kubernetes/2017/4/what-makes-kubernetes-cluster

[Thain 2005] Thain, D., Tannenbaum, T., Livny, M.: "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005

[Wong 2016] Wong, T.: WhitePaper - AMD MULTIUSER GPU: HARDWARE-ENABLED GPU VIRTUALIZATION FOR A TRUE WORKSTATION EXPERIENCE https://www.amd.com/Documents/Multiuser-GPU-White-Paper.pdf, accessed Apr 2018

[Yoo 2003] Yoo, A. B., Jette, M. A., Grondona, M., 2003. Slurm: Simple Linux utility for resource management. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp. 44-60

[Zhou 1993] Zhou, S., Zheng, X., Wang, J., Delisle, P., 1993. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. Software: Practice and Experience 23 (12), 1305–1336

11.2. Links

[Accelerator] What is an accelerator: https://www.techopedia.com/definition/31677/accelerator, accessed Feb 2018

[AI Accelerator] AI accelerator: https://en.wikipedia.org/wiki/AI_accelerator, accessed Feb 2018

[AMDDRIV] https://pro.radeon.com/en/solutions/vdi/, accessed Apr 2018

[Brainware] Microsoft unveils Project Brainwave for real-time AI, Aug 2017: https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/, accessed Apr 2018

[Caffe] Deep learning framework by Berkeley Artificial Intelligence Research (BAIR): http://caffe.berkeleyvision.org/, accessed Feb 2018

[Caffe2] A New Lightweight, Modular, and Scalable Deep Learning Framework: https://caffe2.ai/, accessed Feb 2018

[Chainer] A Powerful, Flexible, and Intuitive Framework for Neural Networks: https://chainer.org/index.html, accessed Feb 2018

[CharlieCloud] Unprivileged Containers for User-Defined Software Stacks in HPC: https://hpc.github.io/charliecloud, accessed Feb 2018

[CNCF] Cloud Native Computing Foundation: https://www.cncf.io/, accessed Feb 2018

[CNTK] Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit: https://docs.microsoft.com/en-us/cognitive-toolkit/, accessed Feb 2018

[Cuda] CUDA Zone - NVIDIA Development: https://developer.NVIDIA.com/cuda-zone, accessed Feb 2018

[cuDNN] NVIDIA cuDNN - GPU Accelerated Deep Learning: https://developer.NVIDIA.com/cudnn, accessed Feb 2018

[CudaToolkit] NVIDIA CUDA Toolkit: https://developer.NVIDIA.com/cuda-toolkit, accessed Feb 2018

[DIGITS] The NVIDIA Deep Learning GPU Training System: https://developer.NVIDIA.com/digits, accessed Feb 2018

[DLAccList] Deep Learning Processor List: https://github.com/basicmi/Deep-Learning-Processor-List, accessed Feb 2018

[Docker] Virtualization technology: https://www.docker.com/, accessed Feb 2018

[DockerFAQ] Docker FAQ - What platforms does Docker run on?: https://docs.docker.com/engine/faq/#what-platforms-does-docker-run-on, accessed Apr 2018

[DockerHub] Docker Hub - Dev-test pipeline automation: https://hub.docker.com/, accessed Apr 2018

[DockerSwarm] Swarm a Docker-native clustering system: https://github.com/docker/swarm, accessed Apr 2018

[DL4J] Deeplearning4j - The first commercial-grade, open-source, distributed deep-learning library written for Java and Scala, integrated with Hadoop and Spark: https://deeplearning4j.org/, accessed Feb 2018

[GIM] https://github.com/GPUOpen-LibrariesAndSDKs/MxGPU-Virtualization, accessed Apr 2018

[Godlove] Containers for HPC, analytics, machine learning, reproducible and trusted computing: http://users.ugent.be/~kehoste/eum18/Singularity_Dave_Godlove.pdf, accessed Apr 2018

[GVT] Intel® Graphics Virtualization Technology (Intel® GVT): https://01.org/igvt-g, accessed Apr 2018

[GVTLINUX] https://github.com/intel/gvt-linux/wiki/GVTg_Setup_Guide, accessed Apr 2018

[H2O] H2O.ai Fast Scalable Machine Learning: http://h2o.ai/, accessed Feb 2018

[HPV SRV-IO] High-Performance Virtualization - SR-IOV and Amazon's C3 Instances: https://glennklockwood.blogspot.pt/2013/12/high-performance-virtualization-sr-iov.html, accessed Mar 2018

[Keras] High-level neural networks API: https://keras.io/, accessed Feb 2018

[Kubernetes] Production-Grade Container Orchestration: https://kubernetes.io/, accessed Feb 2018

[KubernetesGPU] Schedule GPUs: https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/, accessed Feb 2018

[KVMSRIOV] How to assign devices with VT-d in KVM: https://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM, accessed Apr 2018

[LCGDM] LcgDM - Data Management Servers: http://lcgdm.web.cern.ch/, accessed Apr 2018

[LGUEST] Lguest - The Simple x86 Hypervisor: http://lguest.ozlabs.org/, accessed Apr 2018

[LGUEST64] https://github.com/IxLabs/lguest64, accessed Apr 2018

[LinuxFoundation] LinuxFoundation.org | Enable Open Source Ecosystems: http://www.linuxfoundation.org/, accessed Feb 2018

[LXD 2.21 Release] LXD 2.21 Release: https://discuss.linuxcontainers.org/t/lxd-2-21-has-been-released/953, accessed Mar 2018

[LXD 2.5 Release] LXD 2.5 release announcement: https://linuxcontainers.org/lxd/news/#lxd-25-release-announcement, accessed Mar 2018

[LXDoNe] https://github.com/OpenNebula/addon-lxdone, accessed Apr 2018

[MediaTek] MediaTek Powers the Future of Mobile with New Helio P60 Chipset, Bringing Big Core Power & AI Experiences to Consumers: https://www.mediatek.com/news-events/press-releases/mediatek-powers-the-future-of-mobile-with-new-helio-p60-chipset-bringing-big-core-power-ai-experiences-to-consumers, accessed Mar 2018

[MELLANOX] docker-passthrough-plugin: https://github.com/Mellanox/docker-passthrough-plugin, accessed Mar 2018

[nova-lxd] Linux Containers - nova-lxd webpage: https://linuxcontainers.org/lxd/getting-started-OpenStack/, accessed Feb 2018

[nova-lxd_GitHub] OpenStack - nova-lxd on GitHub: https://github.com/OpenStack/nova-lxd, accessed Feb 2018

[NQS] The Network Queueing System: http://gnqs.sourceforge.net/docs/papers/mnqs_papers/original_cosmic_nqs_paper.htm, accessed Feb 2018

[NVGDOCS] http://docs.NVIDIA.com/grid/4.5/grid-vgpu-release-notes-red-hat-el-kvm/index.html, accessed Apr 2018

[NVGFORUM] https://gridforums.NVIDIA.com/default/topic/501/NVIDIA-virtual-gpu-technology/NVIDIA-grid-vgpu-on-kvm-hypervisors/, accessed Apr 2018

[NVIDIA-PATCHES] https://patchwork.kernel.org/patch/9230579/, accessed Apr 2018

[NVIDIAAC] NVIDIA Accelerated Computing: https://developer.NVIDIA.com/computeworks, accessed Feb 2018

[nvidia-docker] Docker Engine Utility for NVIDIA GPUs: https://github.com/NVIDIA/nvidia-docker, accessed Mar 2018

[NVIDIAPascal] NVIDIA PASCAL ARCHITECTURE - Infinite Compute for Infinite Opportunities: https://www.NVIDIA.com/en-us/data-center/pascal-gpu-architecture/, accessed Apr 2018

[NVIDIAVolta] NVIDIA VOLTA - The New GPU Architecture Designed to Bring AI to Every Industry: https://www.NVIDIA.com/en-us/data-center/volta-gpu-architecture/, accessed Apr 2018

[NVPASS] Dedicated GPU Technology for Virtual Desktops: http://www.NVIDIA.com/object/dedicated-gpus.html, accessed Apr 2018

[Magnum] https://docs.OpenStack.org/magnum/, accessed Feb 2018

[Mesos] Apache Mesos - open-source project to manage computer clusters: http://mesos.apache.org/, accessed Feb 2018

[MKL] Intel Math Kernel Library: https://software.intel.com/en-us/intel-mkl/, accessed Feb 2018

[MXNet] Apache MXNet - A flexible and efficient library for deep learning: https://mxnet.apache.org/, accessed Feb 2018

[ONEDock] Docker support for OpenNebula https://github.com/indigo-dc/onedock, accessed Apr 2018

[OpenCL] Open Computing Language - The Khronos Group Inc.: https://www.khronos.org/opencl/; https://developer.NVIDIA.com/opencl, accessed Feb 2018

[OpenLAVA] OpenLava: http://www.openlava.org/, accessed Feb 2018

[OpenMP] OpenMP - API specification for parallel programming: http://www.openmp.org, accessed Feb 2018

[Open MPI] Open MPI - Open Source High Performance Computing: https://www.open-mpi.org/, accessed Feb 2018

[OpenNebula] OpenNebula – Flexible Enterprise Cloud Made Simple: https://opennebula.org/, accessed Feb 2018

[PBS] Portable Batch System Scheduler: http://www.pbspro.org/, accessed Feb 2018

[PCIPASS] Linux virtualization and PCI passthrough: https://www.ibm.com/developerworks/library/l-pci-passthrough/l-pci-passthrough-pdf.pdf, accessed Apr 2018

[PCISIGa] PCI-SIG SR-IOV Primer: https://www.intel.sg/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf, accessed Apr 2018

[SR-IOV LXC] Single-Root Input/Output Virtualization (SR-IOV) with Linux* Containers: https://software.intel.com/en-us/articles/single-root-inputoutput-virtualization-sr-iov-with-linux-containers, accessed Mar 2018

[PyTorch] Deep learning framework that puts Python first: http://pytorch.org/, accessed Feb 2018

[SamsungExynos] Samsung Exynos - Exynos 9 Series 9810 Processor: http://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9810/, accessed Mar 2018

[Singularity] Singularity Official Webpage: http://singularity.lbl.gov/, accessed Mar 2018

[SingularityInEPEL] Singularity RPM in Fedora Packages, https://apps.fedoraproject.org/packages/singularity/,accessed Apr 2018

[Shifter] Shifter - Containers for HPC: https://github.com/NERSC/shifter, accessed Feb 2018

[TensorFlow] TensorFlow - An open-source software library for Machine Intelligence: https://www.tensorflow.org/, accessed Feb 2018

[TensorflowLite] TensorFlow Lite: https://www.tensorflow.org/mobile/, accessed Feb 2018

[Theano] Theano: http://deeplearning.net/software/theano/, accessed Apr 2018

[Torch] Torch - scientific computing framework for LUAJIT http://torch.ch/, accessed Apr 2018

[uDocker] udocker - abasic user tool to execute simple docker containers in user space: https://github.com/indigo-dc/udocker, accessed Apr 2018

[VGPU] NVIDIA GRID Virtual GPU User Guide: http://images.NVIDIA.com/content/grid/pdf/GRID-vGPU-User-Guide.pdf, accessed Apr 2018

[VIRTIO] https://wiki.osdev.org/Virtio, accessed Apr 2018

[VTD] Understanding VT-d: Intel Virtualization Technology for Directed I/O: https://software.intel.com/en-us/blogs/2009/06/25/understanding-vt-d-intel-virtualization-technology-for-directed-io, accessed Apr 2018

[Younge] A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds: http://ieeexplore.ieee.org/document/8241093/, accessed Mar 2018

[XENAPP] http://www.NVIDIA.com/object/xenapp.html, accessed Apr 2018

[XENSRIOV] Xen PCI Passthrough: https://wiki.xen.org/wiki/Xen_PCI_Passthrough, accessed Apr 2018