Bare-metal, Virtual Machines and Containers in OpenStack

Charalampos Gavriil Kominos
Department of Computer Science
Uppsala University
Uppsala, Sweden
Email: h'lastname'@gmail.com

Nicolas Seyvet and Konstantinos Vandikas
Management and Operations of Complex Systems
Ericsson Research
Kista, Sweden
Email: firstname.lastname@ericsson.com

***Abstract*—Cloud computing is an on-demand access model for computing resources most notably embodied by the OpenStack project. As of release Liberty, OpenStack supports provisioning Bare-metal, Virtual machine (VM) and container based hosts. These different hosts incur different overheads. Consequently, the main goal of this paper is to empirically quantify that overhead through a series of experiments.**

The following drivers are leveraged in this process: Ironic for Bare-metal or Metal as a Service (MaaS), nova-compute for VM-based hosts, and nova-docker for Docker based containers.

We make use of a private-cloud in order to compare the different options. This cloud is then used to compare the different hosts in terms of performance (CPU, networking, disk I/O and RAM) by using various open-source benchmarking tools. We also measure boot-up times. The output of these benchmarks is collected and results are compared.

In this paper we discuss our learnings as well as the different configurations and fine-tuning that we implemented. As a result, we provide a set of recommendations based on the advantages and disadvantages of each host in present and future cloud deployments.

I. Introduction

Cloud computing brings an entire new set of value propositions to enterprise computing environments by offering benefits such as application scalability, operational flexibility, improved economies of scale, resource efficiency, agility improvement and more. The National Institute of Standards and Technology [1] defines Cloud as *a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Cloud computing is, at this date, based on four (4) main deployment models [1]:

- Private cloud: The cloud infrastructure is provisioned for exclusive use by a single organization.
- Community cloud: The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns.
- Public cloud: The cloud infrastructure is provisioned for open use by the general public.
- Hybrid cloud: Which is a combination of the public and private models.

Generally it is assumed that the origins of Cloud computing can be traced back to the 1960s when applications had to share system resources provided by mainframe computers. IBM then invested a lot of time and effort in developing robust time-sharing solutions to improve efficiency between users and expensive shared computer resources. Today, the best way to improve resource utilization, and at the same time simplify data center management, is through virtualization.

Virtualization refers to the act of creating a virtual (rather than actual) version of something. It abstracts the physical hardware formed by clusters of servers into large aggregated pools of logical resources. This, in turn, can then be divided and offered back to users in the form of VMs. Today, Infrastructure as a Service (IaaS) is largely synonymous with VMs.

Within IaaS, the OpenStack project is a well-known open source software. OpenStack begun in 2010 as a joint project between Rackspace Hosting and Anso Labs (contracting for NASA). Its first release, code name "Austin", is launched in July 2010. Since then, hundreds of companies have pledged support to the project, and, at the time when this paper is written, there have been thirteen additional releases, the latest being "Mitaka". The OpenStack mission is to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of both public and private clouds regardless of size, by being simple to implement and massively scalable [2].

When deciding how to deploy an application in a private cloud using OpenStack a developer can provision three different hosts:

- Bare-metal: No virtualization, the hardware is fully dedicated, delivered on demand.
- Virtual Machine: Traditional virtualization where the machine appears as a self-contained computer, boots a standard OS kernel, and runs an unmodified application process on top of a Hypervisor layer (see Figure 1).
- Containers: A light-weight approach to isolating resources, where applications share a common kernel.

OpenStack supports these hosts through three different drivers/projects:

- Ironic [3]: To provision bare-metal machines via PXE and Intelligent Platform Management Interface (IPMI)

978-1-5090-3672-1/17/$31.00 ©2017 IEEE

aka Lights Out Management (LOM).
- Nova-compute [4]: To provision VMs using KVM [5] as a hypervisor within an OpenStack compute host.
- Nova-docker [6]: To provision Docker containers within an OpenStack compute host.

The key focus of this paper is to evaluate the different hosts that can be provisioned by aforementioned drivers in an empirical manner; by comparing their performance in terms of CPU, Network, Disk I/O, memory and boot-up times.

In this research space we encounter similar work done by Felter et. al. [7]. The main difference between our work and theirs is that we place our emphasis on OpenStack and on different kinds of hosts that can be provisioned as opposed to performing a straight-cut comparison of hypervisors vs containers. Moreover, in the context of OpenStack there is similar work done by Boden [8]. Our work improves on that by extending that comparison with Ironic and a higher version of Docker, both of which were not available at the time when that comparison took place.

II. Background and related work

In this paper, the OpenStack project is used as an IaaS provider. OpenStack supports three fundamentally different types of computing resources that can be requested by a user: bare-metal, VM and container based. These different options are illustrated in Figure 1.

Ironic (bare-metal) allocates the whole server hardware to the load. Consequently, applications can run natively on the host and fully utilize the underlying hardware. However, this is a single tenant option as unused hardware resources cannot be shared or re-used by others within the data center. As a result bare-metal tends to decrease overall utilization rates, and is often not a cost-effective option. Additionally, from an Ironic project perspective, special extensions are need to allow access to hardware resources. Implementation of these extensions limits the list of supported hardware and leads to increase the cost of developing and maintaining the OpenStack Ironic driver.

The other two types of hosts addressed in this paper overcome the aforementioned limitations by introducing a level of abstraction by means of a hypervisor and the container engine respectively. This addition comes at the expense of performance.

The hypervisor spawns VMs, and presents the guest operating system (OS) with a list of ideal virtual resources through which to manage all interactions with the physical resources. As a result, it achieves load isolation and hardware independence from the guest OS perspective.

There are two types of hypervisors:
- Type one hypervisor or bare-metal hypervisor: VMs lie directly on top of the hypervisor (e.g. ESXi, Xen, KVM)
- Type two hypervisor: VMs lie on top of a physical host which is running its own operating system (e.g. VirtualBox).

Nova-compute uses KVM by default; a type one hypervisor [9] and more specifically the KVM-QEMU pair. Nova-libvirt ([10] and [11]) driver manages the communication between OpenStack and KVM.

While VMs excel at isolation, data exchanges between guests and hypervisors are slower. KVM uses hardware virtualization features in the processors to limit overheads, and supports paravirtual devices via Virtio [12], a virtualization standard for network and disk device drivers where only the guest's device driver "knows" it is running in a virtual environment, and cooperates with the hypervisor. Both features tend to reduce virtualization overheads.

Rather than running a full OS in a VM, containers achieve similar process isolation by modifying a shared OS (operating-system-level virtualization). This eliminates the need for a Guest OS; applications run in their own isolated user space; a sandbox which is manifested via Linux kernel features such as namespaces and cgroups. The entirety of these user spaces is managed by the container engine which gains access to physical resources through the host operating system. The elimination of Guest OS in containers enables them to be more lightweight and faster than VMs thus significantly reducing boot time.

The nova-docker driver spawns Docker [13] containers. Docker containers wrap a piece of software in a complete file-system that contains everything needed to run an application: Code, runtime, system tools, system libraries anything that can be installed on a server.

The remaining sections of the paper are structured as follows: The Methodology section describes our setup and the different configurations we have implemented in order to provision the individual hosts. In addition, it describes the different assumptions we made and the expectations we had for each test. The Evaluation section presents and analyzes the different tests that have been performed for each kind of host for the following resources: CPU, Network, Memory, disk I/O and booting times. Finally, the paper finishes off with the conclusions that can be derived from this work and a set of further steps that can be taken as part of Future Work.

III. Methodology

In this section we describe the hardware setup, software tools and configuration choices that have been made in our evaluation. Before we do that we start by giving more specific definitions to a set of terms that are used extensively throughout this paper.

A. Terminology

Within the scope of this paper, the following terms are used:
- A Compute host or host is a node managed by OpenStack to instantiate a VM, or a container. It runs various OpenStack services such as the Neutron agent, ceilometer, a nova agent, Open vSwitch [14] (OVS), a messaging client (RabbitMQ), etc.
- An Ironic host is a bare-metal node managed by OpenStack. The physical server is entirely allocated to the load and does not run any OpenStack services.

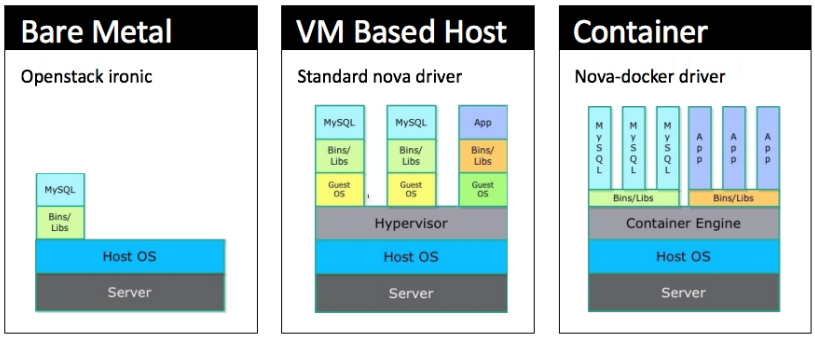

Fig. 1. Different hosts in OpenStack

- A docker container within a compute host is referred to as Docker.
- A VM within a compute host is referred to as VM.

B. Hardware

Our setup consists of five servers located in the same rack. Each physical server is a Hewlett Packard G6 blade equipped with two Intel(R) Xeon(R) CPU E5540, 72 GB of RAM and two 160 GB 10000 rpm hard disks in RAID0 configuration. The servers are connected via two NICs to a 1000/1000 network through an Top of Rack (ToR) Extreme Networks X450a-48t Switch. One of the servers is dedicated to be the OpenStack controller and the four others are compute/ironic hosts.

Note that during the evaluation phase, a particular server is singled out to avoid variations. This is the server under test. In cases where communication to a remote VM or container is required, then a different server connected to the ToR is used in order to limit the number of network hops to one. To ensure like-with-like is tested, the system under test is always the same server. There is no networking traffic outside of the tests as the cluster is exclusive and free-standing.

Networking in OpenStack uses Neutron and VLANs. The PXE network (NIC1, untagged) is required for Fuel to control and deploy the physical servers. The Public network (NIC1, 1200) is used to distribute public IPs to VMs, containers and hosts. Storage (NIC1, 1062) and management (NIC1, 1061) are separate internal networks used by OpenStack management. The ”private” tags (NIC2, 1063-1080) are used for isolating each virtual network. The bare-metal network (NIC2, 1082) is reserved for Ironic provisioned instances. Standard tuning is applied for 1Gbps network.

Since the Ironic conductor achieves bare-metal provisioning through Lights Out Management (LOM) protocol, the servers are communicating via the Integrated Lights Out (ILO) interface. ILO is HP's version of LOM. LOM allows a system administrator to monitor and manage servers and other network-attached equipment by remote control regardless of whether the machine is powered on, or if an operating system is installed or functional.

C. IaaS software

We used the OpenStack Liberty release, and Docker v1.11. Those were the latest available versions at the time.

D. Benchmarking tools

Our evaluation focuses on CPU, memory, disk I/O, network (latency and bandwidth) and boot-up times. To stress each resource, the following open-source tools have been selected:

- **CPU:** the PXZ program has been selected. [15] This tool, which is widely available in Ubuntu and Fedora, implements the classic Lempel-Ziv-Markov compression algorithm [16]. PXZ is a parallel loss-less compression tool and can be easily configured to run in any number of cores. As input for PXZ a 1GB Wikipedia data dump has been selected [17]. The file is fed into the PXZ algorithm while varying the number of allocated cores. The wall time that PXZ takes to compress the file is then recorded.
- **Network:** Network throughput and latency were tested using two different tools:
  - Nuttcp [18] is used to measure network throughput. This tool works in a client-server fashion. The system to be tested acts as client and an external system in the cloud acts as a server. Nuttcp traffic is generated between the client and server. The throughput is a variable and tests are made between 650 and 1100 Mbps. We measure both inbound and outbound throughput.
  - Netperf [19] is used to measure network latency. It also works in a client-server model although in this case the machine to be tested acts as a server. The client is then used to generate a packet of fixed length (200 bytes). When that packet reaches the server, the server sends back a response of fixed length (200 bytes). Therefore only one packet is in flight at any point in time. We measure how many packets are sent within a fixed period of time.
- **Memory:** In the tested system four memory levels are available (L1 cache, L2 cache, L3 cache, RAM). In order to test all these resources the open-source bandwidth [20] tool was selected. In this tool memory chunks of different

sizes are copied into memory. The tool starts from 256 bit and continues until the chunks are 512 MB in size.

- **Disk I/O:** In order to test system disk I/O two different tools were used:
  - SysBench [21].
  - Linux binary copy (dd) utility.

  Disk IO tools operate by copying and reading from and to big chunks of data. For such benchmarking, it is important to prevent RAM involvement (buffer caches at OS level). To achieve that:
  - VMs and containers are limited to 8 GB of RAM.
  - For Ironic and Host experiments, the physical server is stripped to two DIMMs.
  - The benchmark files will have a size of 16 GB (i.e. twice the size of the RAM).

Tool	Version	Resource
ParallelPXZ	4.999.9beta	CPU
Nuttcp	6.1.2	Network
Netperf	2.6.0	Network
Bandwidth	1.3.1	Memory
SysBench	0.4.12	Disk
dd	8.25	Disk

TABLE I
BENCHMARKING TOOLS

Table I summarizes the different tools used in our evaluation.

E. Configuration choices

For I/O tests instead of using the nova-docker driver, we spawned a standard Ubuntu 14.04 container directly from the CLI. The reason is that in a standard container, disk access is not to the device but to a copy-on-write file-system (Advanced multi-layered Unification FileSystem, AUFS). Since our intent is not to compare AUFS vs VM block based storage, but to compare how different hosts affect the access to a resource, this approach permits mounting a data volume to access the local Compute host disk and thus to bypass AUFS. Similarly the VM is using the "mount host-file" option to access the local host hard drive. One of the caveats of using Docker directly was security issues with App Armor.

For CPU measurements of KVM VMs, the CPU allocation model is set to "host-passthrough" [22], With this mode, the CPU is visible to the guest VM is exactly the same as the host physical machine CPU. By default, nova-compute does not optimize KVM. Consequently, CPU pinning is not enabled.

IV. EVALUATION

The different experiments for each resource performed in our evaluation are repeated ten times. Mean and standard deviation are then computed. The standard deviation is displayed as error bar in the diagrams. When the error bar is not visible, assume the deviation to be less than 1%.

The criteria for this evaluation are CPU (and CPU contention), networking latency/bandwidth, memory, boot-up time and disk I/O. These have been chosen since these are the usual required resources when provisioning a node and as such impact the user's experience when deploying an application in Openstack.

The results for each experiment are presented and analyzed in the following subsections.

A. CPU

For CPU tests, four different OpenStack VMs are created respectively with 1,2,4,8 vCPUs. We also leverage **PXZ**'s ability configure the number of cores. For example a VM with 2 vCPUs is compared with a bare-metal machine where **PXZ** is limited to 2 cores. The results for the different configurations are shown in Tables II and III.

Resource	1 vCPU			2 vCPU		
	time (s)	std	%	time (s)	std	%
Bare-metal	670.8	6.9	-2%	360.4	4.3	-7%
Compute host	683.8	2.8	0%	388.9	3.8	0%
Docker	685.2	3.9	0%	387.0	7.6	0%
VM	709.6	7.8	+3%	393.6	3.4	+1%

TABLE II
1-2 VCPU PERFORMANCE

Resource	4 vCPU			8 vCPU		
	time (s)	std	%	time (s)	std	%
Bare-metal	192.0	0.8	-9%	109.5	1.0	-16%
Compute host	211.3	2.1	0%	131.4	2.1	0%
Container	211.7	2.5	0%	131.2	1.4	0%
VM	223.9	1.2	+5%	141.1	1.0	+7%

TABLE III
4-8 VCPU PERFORMANCE

From this evaluation the following observations can be made:

- Bare-metal offers best performance.
- As highlighted by the difference between bare-metal and Compute hosts, OpenStack services consume a non-negligible part of the CPU.
- The worst performer is the VM. This is in part due to the hypervisor as its main task is to schedule time in the physical CPU for the guests processes. For example, when the hypervisor requests 4 vCPU cores for a limited amount of time, it must wait until these resources become available.
- The Docker container performs on par with the compute host.
- Surprisingly, when the number of vCPUs allocated to a VM increases, the overall performance degrades. I.e. while an improvement in overall computation time is measured, the difference with running on bare-metal increases. The number of allocated vCPUs by the VM is a major factor when trying to approximate hypervisor overhead. Our tests indicate a performance drop of about 2%, 5% and 7% when allocating respectively 1, 2, 4 and 8 vCPUs to a VM. The more CPU cycles that the hypervisor schedules for, the more the performance drops percentage-wise. KVM CPU pinning could have improved the VM performance significantly as indicated by [8].

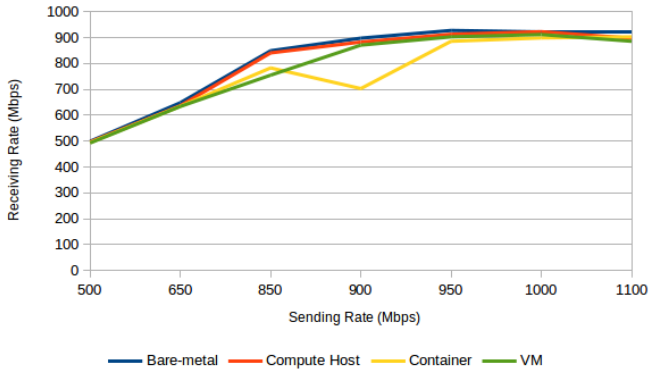

Fig. 2. Network Bandwidth

B. CPU contention

Contention is the situation where different processes compete for a single resource. To assess this we created up to 4 VMs and Containers with 4 vCPUs/CPUs on a single server and then ran PXZ.

# of instances	VM	Container	% diff
1	216s	203s	6.0%
2	252s	236s	6.3%
3	367s	339s	7.6%
4	483s	445s	7.9%

TABLE IV
CPU CONTENTION: PXZ EXECUTION TIME IN SECONDS

As shown in Table IV, running within Docker containers is slightly faster than running within a hypervisor. As contention increases, the delta between containers and VMs also increases in favor of containers.

C. Networking

1) Bandwidth: **Nuttcp** [18] uses a client/server approach where the client is the system under test, and the server another node within the cluster. It is configured to generate traffic at different rates ranging from 500 to 1100 Mbps.

As expected, in our measurements, see Figure 2, the link saturates just below 1000 Mbps. This value is the expected maximum rate of the network(switch and NIC) when considering packet headers.

The following observations can be made:

- Bare-metal based host is closer to maximum throughput than other options,
- All other options are quite close to bare-metal except the Docker case.
- The Docker container based host did not perform as well in nearly every case despite several re-configuration attempts.

A partial explanation to above results is that the OpenStack internal communication being based on TCP as a protocol, a small part of the network traffic in all cases except Ironic is used by the internal messaging system.

In the Docker case, when taken out of the box a 50% drop in bandwidth was initially observed. It is only after lowering the Maximum Transmission Unit (MTU) within the container from 1500 to 1420 that a more reasonable performance was achieved. In our setup, both nova-docker and Neutron cooperate to provide networking within the container. During the instantiation, the container is assigned a networking namespace belonging the host that is connected to the tap port provided by Neutron (see Figure 3). In our experiments, we observed that the tap ports received traffic with an MTU set to 1420 while the nova-docker daemon expected 1500. This may also be a consequences of issues within Docker 1.11([24]).

This tuning of the container was only possible from within the host, and a normal OpenStack nova-docker user would have been unable to manipulate the stack to reach such a result which is a significant drawback. Further research and experimentation is advised with newer Docker releases.

Resource	TCP	Reverse TCP	UDP	Reverse UDP
Bare-metal	923	929	941	931
Compute host	923	–	937	—
Container	899	327	932	939
VM	912	931	939	939

TABLE V
TCP & UDP MAXIMUM PERFORMANCE

Table V displays the findings for maximum throughput achieved in TCP and UDP traffic in both directions. Reverse TCP and UDP traffic were not measured successfully in the time set and were omitted.

2) Latency: **Netperf** [19] was used for testing the system's latency. For this tool, the machine that is to be tested acts as a server and another machine outside the cloud acts as the client. In order to make a good approximation of system latency we performed a fixed time test (30 sec). In this time frame, the tool calculates how many packets were sent and received. Therefore we can approximate the time it takes each packet to reach its destination.

We use formula 1 from [25] to approximate system latency.

$$\frac{\frac{1}{TransferRate(Req/s)} * \frac{10^6 \mu s}{1s}}{2} \quad (1)$$

Setup	UDP (req/s)	UDP std	UDP latency (μs)	TCP (req/s)	TCP std	TCP latency (μs)
Bare-metal	1917	100	260	1944	102	257
Compute host	2016	59	250	1992	87	250
Container	1673	67	298	1724	93	289
VM	1380	64	362	1430	43	349

TABLE VI
MEASURED TCP & UDP LATENCY

It can be observed from Table VI that while both Ironic and the Compute host show similar performance results, the extra networking layers added by Neutron (Figure 3) introduce some 40 μs latency both for Docker and VM. In the VM's case, there is an extra 60 μs inferred by the hypervisor bringing the total to 100 μs when compared to a bare-metal option.

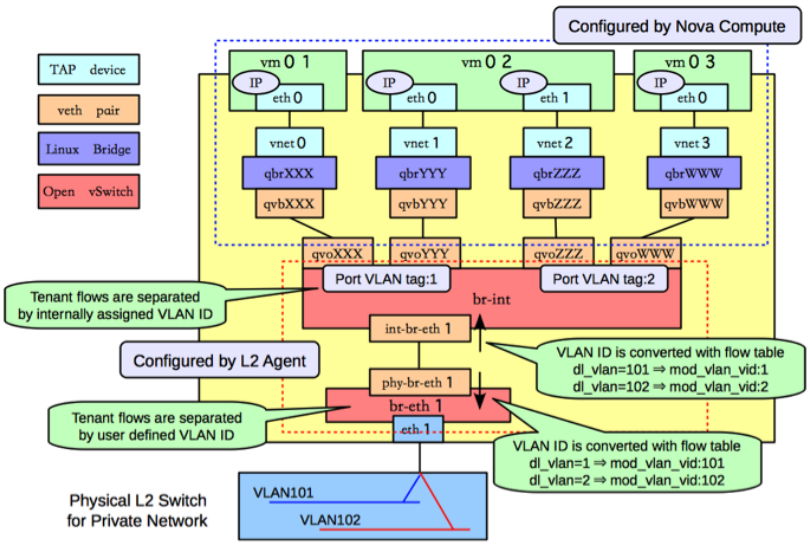

Fig. 3. Compute node networking using Neutron (VLAN) in Liberty release [23]

D. Memory

The **Bandwidth** tool [20] uses a random data batch of variable size (from 256bit to 512 MB) to stress system memory. It is used to test the random read and write speeds which are typical access types for applications.

The Intel Xeon E5440 CPU used in our setup has L1 (data/instruction) cache of 32KB, L2 of 256KB, and L3 of 8192KB. The different memory levels are clearly visible in Figures (4, 5, 6). And, as expected, each level outperforms the previous by a considerable factor.

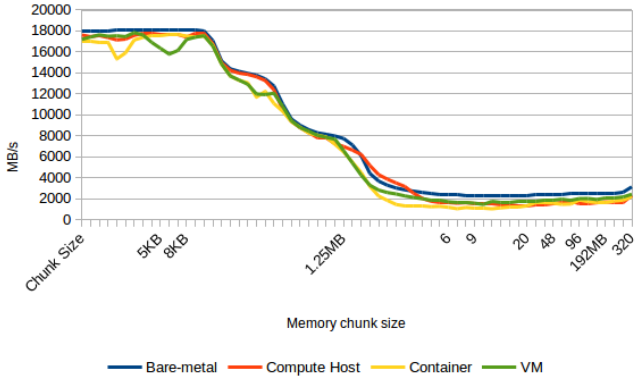

Fig. 4. Memory access patterns: Random read

Figure 6 illustrates our data set partitioned in four data-clusters (256b, 48KB, 768KB, 512MB), one for each memory level. From this figure we observe:

- Read performance does not show significant fluctuations regardless of the system tested.
- RAM write performance in the VM shows a performance drop of approximately 30%.

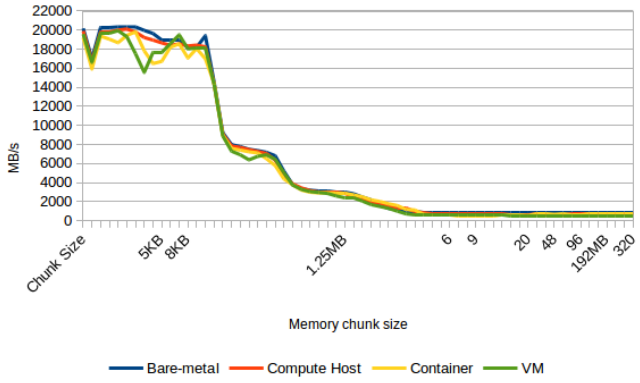

Fig. 5. Memory access patterns: Random writes

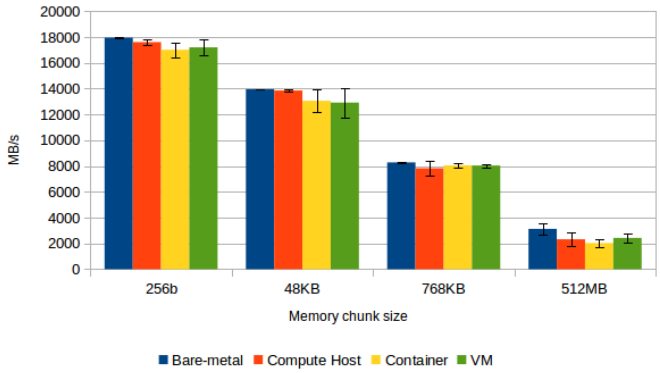

Fig. 6. Random Read example

- Bare-metal has faster memory access than Compute host. The OpenStack services are adding an overhead of at least 5% for all tests.

E. Disk I/O

Two different tools are used to stress disk I/O and measure first number of requests/s, then MB/s.

1) SysBench: We use **Sysbench** [21] to create a 16GB set of files and an internal 70%/30% distribution of read/write to stress the I/O. Since Sysbench cannot force data to always be stored on disk, the test length is extended to minimize the effects of data caching in RAM. As such, we measured I/O requests over a 20 minute period.

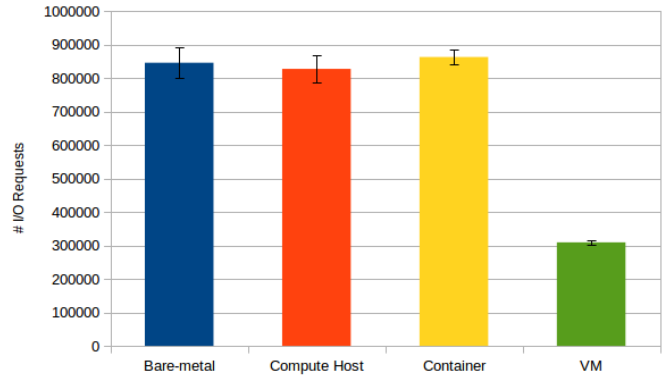

Fig. 7. File Test I/O

As indicated by Figure 7, While bare-metal, compute and Docker achieve similar behavior (within the standard deviation), the VM handles only 35% of I/O requests.

2) Binary Copy: A simple binary copy with the Linux's **dd** is done to create a 4.3GB file filed with zeros. The results are compiled in Table VII.

	s	MB/s	std
Bare-metal	28.0	148	2.5
Compute host	28.7	149	1.4
Container	31.2	138	1.9
VM	82.0	104	2.4

TABLE VII
BINARY COPY PERFORMANCE

Bare-metal and compute host exhibit similar performance. While Container experiences a 6% performance degradation and VM performance drops by 30%.

It seems that in cases of full load, the KVM/QEMU hypervisor slows down the data transfer considerably. We speculate that with further KVM tuning higher Input/Output(IO) speeds can be achieved. However, as mentioned in [7] because each request must go through QEMU, the system will require more IO Operations per second (IOPS) and CPU instructions per second for the same absolute performance. This leaves fewer resources available for useful application work.

F. Boot-up time

To measure boot-up times, the OpenStack logs are used. An instance is said to be available when it reaches the "login" stage. Although this method is not overly accurate, it is a good enough estimation of the time required for a resource to become usable.

	Boot-up time (s)
Bare-metal	600
Container	5
VM	105

TABLE VIII
BOOT-UP TIMES

Depending on the job execution length, the boot-up time for a computing resource to become available to a user may or may not be relevant. If the job is going to run for days or months then the above values are not important. But for short jobs ($<$ 1-2 h), then the differences are significant enough to favor container based setups.

While the Boden report [8] indicates boot-up times of 5s and 3s respectively for VMs and containers, we did not see any optimization that would justify such improvements. We assume that the authors used the time taken to allocate a resource, not the time where it becomes usable. While a VM instantiation is quick, it does not imply that it is capable of handling application load. For our measurements, we assume that a computing resource is ready when it can be **ssh**-ed in and not just pinged or visible in the horizon interface.

Note that the measured time includes transferring the image from the Glance repository to the server, This impacts Bare-Metal worse than VMs or Container as it is a mandatory step during provisioning while other options can use host based caching mechanisms on the compute.

V. Conclusions

The main goal of this paper is to offer an evaluation of different hosts that can be provisioned in OpenStack. Within this scope, we have evaluated bare-metal hosts, VM based hosts and Docker container based hosts in terms of CPU, Networking, Memory, Disk I/O and boot-up times. We also evaluated the impacts of running OpenStack services on a compute host.

As expected, bare-metal outperforms all other options in our tests. As such it is recommended for high intensity workloads. Deploying bare-metal via Ironic introduces no overhead and applications can run native on the hardware. However, the absence of overhead also means a lack of flexibility, and an overall lower utilization rate as no resources are shared. For example, there is no network isolation as all bare-metal instances share the same network address range. Note also that while the Ironic project is a recent addition to OpenStack we encountered no issues using it during our experimentation.

On the opposite-end in our evaluation we have VM based hosts. We observe that VM overhead varies and depends on both the workload and the quantity of assigned virtual resources. More specifically, our CPU tests indicate a strong correlation between the number of resources requested (vCPU) and the measured overhead where performance degrades from 1-2% (negligible) to 8-10% (important). Beyond CPU, VM based hosts exhibit degraded performance when it comes to Memory and Disk I/O. VM networking performance is on par with the other alternatives.

Docker container based hosts have the fastest boot-time and an overall performance on par with bare-metal with the one exception being networking bandwidth. This is a strong indication that the nova-docker project is viable and allows OpenStack to combine the speed of Containers with the flexibility and isolation guarantees of OpenStack's Neutron based networking. There remains some restrictions to deploy nova-docker at scale. First there is this issue with the MTU setting in the container, then nova-docker specifies a hard-coded space limit of 26GB per node for running containers. These two limitations make traditional VM-based hosts easier to setup and maintain over the long term.

While security is out of scope in this paper, there is no best options. It varies mostly based on the size and quality of the surface of attack [26]. Certain benefits are generally attributed to containers as they are both smaller and more ephemeral by nature and thus reduce potential security risks associated with the applications.

Finally, a surprising result is the non-negligible CPU (2-7%) and networking (40 μs) overheads introduced by the OpenStack services running on all the compute nodes of the cluster.

VI. Future Work

Overall, OpenStack can be setup to control and deploy different options from bare-metal, to containers, and VMs. Such an environment presents an interesting solution to a complete data center deployment.

In the scope of Future work it would be interesting to repeat this evaluation and enable certain features and optimization that may yield better performance such as KVM CPU pinning with regards to CPU performance for VM-based hosts. In addition it is important to further investigate the issues behind the poor networking performance in Docker despite our efforts for configuring the MTU. Another interesting dimension to investigate is that of the AUFS available in Docker which could also yield better Disk I/O performance in certain scenarios since it is based on copy-on-write.

Moreover, there are efforts to integrate the advantages provided by combining bare-metal and container technology. One such project named Kolla [27] provides production-ready containers and deployment tools for operating OpenStack. Kolla uses containers to instantiate and manage the deployment of OpenStack.

Overall, we observe a considerable pressure to move away from pure VM based clouds to more flexible computing environments that make better use of available hardware resources, and improve the revenues/server ratio for cloud providers.

In parallel, cloud computing continues to evolve with new approaches to application deployment. Two such newly emerging options are serverless computing and unikernels. Serverless computing completely removes the need to care for servers (including virtual ones) by adding another layer of abstraction atop the cloud infrastructure. Running an application becomes only about purposefully built code, and a number of operations to perform i.e. cycles. Unikernels re-use the concept of minimalistic operating systems introduced by Containers. A unikernel is single-user, single-process, specialized operating systems containing the full application stack. While deployed on top an hypervisor, they benefit from being light and ultra compact and as a result compete in performance with containers.

Acknowledgment

The authors would like to thank the following people from the IT group that helped us setup the hardware and network, particularly Patrik Usher, Christopher Lange and Rajive Yadav, as well as all the OpenStack developers in in the IRC channels #fuel, #nova-docker, #openstack-ironic, #openstack-neutron.

References

[1] "Final version of nist cloud computing definition published." [Online]. Available: https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published

[2] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[3] "Ironic project: Openstack bare metal provisioning program." [Online]. Available: https://wiki.openstack.org/wiki/Ironic

[4] "Openstack nova." [Online]. Available: https://wiki.openstack.org/wiki/Nova

[5] "Kvm." [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[6] "Docker driver for openstack nova." [Online]. Available: https://github.com/openstack/nova-docker

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[8] "Passive benchmarking with docker lxc, kvm & openstack v 2.0," April 2014. [Online]. Available: http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack

[9] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[10] "Libvirt: The virtualization api." [Online]. Available: http://libvirt.org/

[11] "Under the hood with nova, libvirt and kvm," OpenStack Summit, 2014. [Online]. Available: https://www.openstack.org/assets/presentation-media/OSSummitAtlanta2014-NovaLibvirtKVM2.pdf

[12] "Paravirtualized drivers for kvm/linux." [Online]. Available: http://www.linux-kvm.org/page/Virtio

[13] "Docker: Build, ship, run." [Online]. Available: https://www.docker.com/

[14] "Openvswitch." [Online]. Available: http://openvswitch.org/

[15] "Devstack project." [Online]. Available: http://docs.openstack.org/developer/devstack/

[16] "Lempelzivmarkov chain algorithm." [Online]. Available: https://en.wikipedia.org/wiki/LempelZivMarkov_chain_algorithm

[17] "Hutter." [Online]. Available: https://cs.fit.edu/~mmahoney/compression/textdata.html

[18] "Nuttcp." [Online]. Available: http://www.nuttcp.net/

[19] "Netperf." [Online]. Available: http://www.netperf.org/netperf/

[20] "Bandwidth: a memory bandwidth benchmark." [Online]. Available: https://zsmith.co/bandwidth.html

[21] "Sysbench." [Online]. Available: https://github.com/akopytov/sysbench

[22] "Hostpassthrough." [Online]. Available: https://libvirt.or/formatdomain.html\#elementsCPUAllocation

[23] "Welcome to openstack documentation." [Online]. Available: http://docs.openstack.org/liberty/

[24] "containers in docker 1.11 does not get same mtu as host." [Online]. Available: https://github.com/docker/docker/issues/22297

[25] "Ibmnetperf." [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSQPD3_2.4.0/com.ibm.wllm.doc/runnetpe

[26] G. Hulme, "Containers and security q&a: Putting a lid on risk." [Online]. Available: http://containerjournal.com/2016/04/13/containers-security-qa-putting-lid-risk/

[27] "Welcome to kollas documentation!" [Online]. Available: http://docs.openstack.org/developer/kolla/