UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

KAIO KASSIANO MOURA OLIVEIRA

ASSESSING THE USE OF THE BEEFS DISTRIBUTED FILE SYSTEM ON THE PERFORMANCE OF OPENSTACK

CAMPINA GRANDE - PB

2019

KAIO KASSIANO MOURA OLIVEIRA

ASSESSING THE USE OF THE BEEFS DISTRIBUTED FILE SYSTEM ON THE PERFORMANCE OF OPENSTACK

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Francisco Vilar Brasileiro.

CAMPINA GRANDE - PB

2019

O48a Oliveira, Kaio Kassiano Moura.
Assessing the use of the BeeFS distributed file system on the performance of OpenStack. / Kaio Kassiano Moura Oliveira. – 2019.

12 f.

Orientadores: Prof. Dr. Francisco Vilar Brasileiro.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. OpenStack. 2. Distributed file storage. 3. Ceph. 4. BeeFS. 5. OpenStack cloud. I. Brasileiro, Francisco Vilar. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

KAIO KASSIANO MOURA OLIVEIRA

ASSESSING THE USE OF THE BEEFS DISTRIBUTED FILE SYSTEM ON THE PERFORMANCE OF OPENSTACK

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Francisco Vilar Brasileiro
Orientador – UASC/CEEI/UFCG

Professora Dra. Lívia Maria Rodrigues Sampaio Campos
Examinadora – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 02 de julho de 2019.

CAMPINA GRANDE - PB

Assessing the use of the BeeFS distributed file system on the performance of OpenStack

Kaio Kassiano Moura Oliveira
kaio.kassiano.oliveira@ccc.ufcg.edu.br
Universidade Federal de Campina Grande

ABSTRACT

OpenStack is one of the most used open source solutions to create private clouds for researching, testing and business purposes. As a cloud orchestrator, service aggregator, and IaaS provider, OpenStack controls pools of computing, storage and network resources in a data center. OpenStack services that provide some kind of storage (*e.g.*, block, image, blob) rely on an underlying file system to provide the service and also for performance matters. This work presents the integration of the Beehive File System (BeeFS) as a distributed file system back-end storage for an OpenStack cloud, and a performance assessment considering Nova, a fundamental service in charge of managing virtual machines in the cloud. Experimental benchmarking results show that, in average, Nova performs 5 times faster using Ceph - the *de facto* distributed storage technology used in OpenStack - then when using BeeFS as its storage back-end. Although BeeFS reduces costs of creating distributed storage infrastructure, its integration with OpenStack is not complete which causes significant impact in the cloud applications performance.

KEYWORDS

OpenStack, distributed file storage, Ceph, BeeFS

1 ACKNOWLEDGEMENTS

I would like to dedicate this work to my family who helped me through this journey: without them, my victory and success would never be possible. To my mother Celia Moura, who always provided everything for me; I will be forever thankful for your support and unconditional love. To my father, Josue Rufino, who guided and taught me to be a decent and reasonable person; I will always miss you - may you rest in peace. To my love Rayane Dantas, who believed and encouraged me when times were difficult; I will always be thankful for your love and your company.

To Ph.D. Francisco "Fubica" Vilar Brasileiro, who guided me through the development of this work; thank you for your patience and suggestions.

To my colleagues Sérgio Duarte, Lucas Cavalcante, Guilherme Steinmüller, Joab Silva, Paulo Feitosa, Benardi Nunes in Distributed Systems Laboratory; thank for all your help and support. To my friends and Ph.Ds. Thiago Emannuel and Matheus Gaudêncio; thank you for your assistance, input, and advice. To Ph.D. Andrey Brito; thank you for believing in me and giving me the opportunity to work with you.

2 INTRODUCTION

OpenStack is an open source, distributed, scalable, and flexible service aggregator as well a cloud orchestrator. It controls resources of computing, storage, and network on a data center, providing a cloud Infrastructure-as-a-Service (IaaS) environment. OpenStack coordinates multiple services, and each service is responsible for managing and providing a given functionality. For instance, Nova is the compute service and the main component of OpenStack, and it is responsible for launch VMs (Virtual Machines) and manage the compute nodes in which VMs will be launched, along with appropriate VM placement heuristics; Neutron is the service responsible for managing the OpenStack network; Keystone manages users, projects, access control. There are 50+ existing projects [8], and many of them are not obligatory to be used on OpenStack.

Some OpenStack services that provides some kind of storage (*e.g.*, Nova - disk, Glance - volume, Cinder - image) require a file system back-end. For instance, when Nova runs a virtual machine, it can use many approaches storage technologies, such as distributed file systems or object storage. Depending on user's needs, one or many of these storage technologies could be deployed on the cloud. The performance of such services depends on the back-end architecture and characteristics, which means a given distributed storage solution could determine the overall performance of OpenStack. A few examples of common back-ends used are NFS, Gluster, LVM, and Ceph, the latter being the most used and the state-of-practice as distributed storage for OpenStack [3]. In this work we focus on evaluating the performance of OpenStack Nova when launching VMs to run a well-known performance workload, the FIO benchmark [1]. Nova will be configured to use two different storage solutions: Ceph (the state-of-practice in terms of distributed storage for OpenStack), and BeeFS (a cheap and naturally-scalable distributed file system, created and developed at the Distributed Systems Laboratory [20]).

Ceph is a distributed, fault-tolerant, reliable and massively scalable storage technology. First proposed in 2007 [22], Ceph focuses on implementing object storage using an algorithm

called CRUSH to place and organize data in a unified storage system, called RADOS. Ceph architecture uses a monitor entity to manage cluster consistency and coherence. Also, monitors maintain a master map of the RADOS cluster, which includes, for instance, how many and where are the cluster monitors and the cluster OSDs. OSDs are the object storage daemons and they are responsible for storing the actual data (*i.e.*, objects). A Ceph client interacts with the monitor to obtain a recent cluster map, and using the CRUSH algorithm, it can compute the location of any object across the RADOS cluster, and then interacts with the OSD to retrieve/access the object. Monitors manage security and access control of clients to objects.

Our main purpose will be the introduction of the BeeFS file system as another back-end storage technology for OpenStack, particularly for OpenStack Nova. BeeFS is a distributed file system that relies on the harnessing of free disk space of machines to provide distributed storage. Following a hybrid architecture, BeeFS consists of a centralised node, the queen-bee, that is responsible for metadata and file replication management. Data is actually stored on data servers, the honeycombs, that collaboratively store all the file system data. BeeFS clients, the honeybees, retrieve metadata and file localisation from the queen-bee, and then interact directly with honeycombs to fetch/send data. BeeFS is fully POSIX-compliant, and the coupling between the user level application and the Linux kernel file system modules is done via FUSE [13, 20]. A single node can act as both client and data server at the same time, which indicates a possible performance leverage.

We believe that the BeeFS, a cheap and naturally scalable storage technology, can actually serve as storage back-end for OpenStack. Specially when using data servers running on the same nodes as where the clients are, performance should be compared to a state-of-practice distributed storage technology.

However, our hypotheses have shown wrong. In average, results are that Nova+Ceph performs 5 times faster than Nova+BeeFS for FIO benchmark execution. Also, the time for VM initialisation **should** perform better on BeeFS than on Ceph, because BeeFS does not pre-allocate the VM entire disk as Ceph does [19]. This shows that when Nova is aware of the file system back-end it implies on enormous performance gain, as Nova knows how Ceph operates and uses proper drives for communicating with Ceph cluster [5].

The rest of this document is organised as follows: Section II presents a short summary of related work; Section III is an overview of OpenStack as cloud provider orchestrator; Section IV provides a brief discussion of centralised versus distributed storage and presents the state-of-practice distributed storage technology used in OpenStack; Section V an overview of BeeFS; Section VI the methodology to integrate and evaluate BeeFS as back-end storage for OpenStack; Section VII provides performance results and discussion; finally Section VIII presents the conclusion with the final work remarks.

3 RELATED WORK

Virtual Machine disk performance using different back-ends were analysed by Johari et al [14]. They found better performance execution for GlusterFS when compared to Ceph, using mixed read/write workloads with block sizes ranging from 16K to 256K. Specially for writing or rewriting small size of files between 4K and 8K, Ceph seems to be the reasonable choice for most applications.

Other performance comparison works focuses on isolating the storage technologies and file systems, removing the cloud and hypervisor layer (*e.g.*, QEMU) and measuring performance only without any further integration [6]. In this work, we wanted to understand how OpenStack performance could change with different storage technologies as its back-end.

4 OPENSTACK IN A NUTSHELL

OpenStack is an Infrastructure-as-a-Service (IaaS) provider solution[1] and a cloud computing platform. Initially developed by NASA and Rackspace on 2010 [10], its code was open sourced right after the development of the first version of Nova, a "Python-based cloud computing fabric controller". OpenStack consists of various projects that can control pools of computing, storage, and network resources in a data center. Managed by a consortium that includes about 600 IT companies (such as Red Hat, Cisco, Dell EMC, VEXXHOST, Rackspace, and SUSE), OpenStack is still under active development. Its main characteristics are the following:

- *Open*: as an open source technology, the entire source code can be modified and adapted by anyone who needs to. OpenStack is licensed under the Apache 2 license.
- *Scalable*: developed following a distributed architecture, OpenStack can scale up to hundreds of thousands of physical and virtual machines [15].
- *Flexible*: supports many existing virtualisation technologies, such as KVM, QEMU, Xen, Microsoft Hyper-V, and LXC [9].

OpenStack services offer interfaces (APIs) that ease integration between services. An OpenStack deployer (*e.g.*, cloud

[1]IaaS is a provisioning model in which an organization outsources physical components of a data center, such as storage, hardware, servers, and networking components. A service provider owns the equipment and is responsible for housing, operating and maintaining it. The client typically pays on a per-use basis.

operator) can install some or several OpenStack services, depending on what she needs. Typical OpenStack deployments normally comprise: one or more infrastructure nodes, where all APIs are running; one or more compute nodes, where OpenStack can launch Virtual Machines (VMs), containers or even functions; and one or more storage nodes.

Common OpenStack services and projects

Figure 1 provides an overview of a basic OpenStack service architecture including components characterized as core components [7], which should exist on any OpenStack deployment.

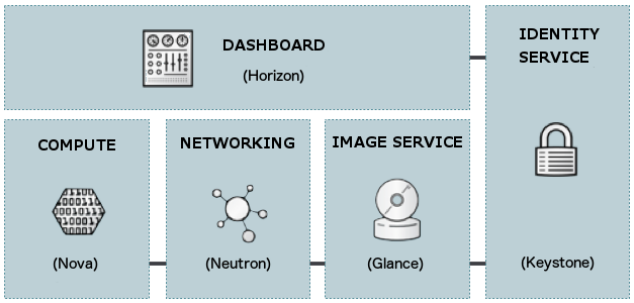

Figure 1: OpenStack map

The description of the core components are the following:

- **OpenStack Compute (Nova)** is the main component of OpenStack. It manages the computing service. Nova creates an abstraction layer for virtualizing hardware of physical machines (*e.g.,* compute nodes) resources such as CPU, RAM, network adapters, and hard drives. Nova provides management functions to launch, resize, suspend, stop, and reboot VMs.
- **OpenStack Networking (Neutron)** provides "network connectivity as a service". Neutron includes the capability to manage LANs and VLANs, Dynamic Host Configuration Protocol (DHCP), and Internet Protocol versions 4 and 6. Neutron users and operators can create networks, subnets, and routers to configure their internal topology. Floating IP addresses allow users to assign (and reassign) fixed IP addresses to the VMs.
- **OpenStack Image Service (Glance)** provides support for VM images. Images are the system disks to be used when launching new VMs. Glance also provides capabilities for creating VM snapshots and backups, which can be used as templates to launch new VMs. Images are provided by users as private or public images in a variety of formats, such as QCOW2, RAW, VDI, RAW, VDI, and VHD.
- **OpenStack Identity Mangement (Keystone)** manages domains, projects, users, and a list of services they can access. It exposes a central authentication mechanism across OpenStack components. Keystone can integrate with many other directory services, for instance, LDAP and OAuth. Administrators can create projects and users, assign them to administrative domains, and also configure access policies that apply to users and services. Users and components can retrieve a listing of which services they can access.
- **OpenStack Dashboard (Horizon)** provides a web-based user interface to OpenStack. Administrators and users can interact with almost all OpenStack services through Horizon. For instance, users can use Horizon to request the creation of VMs and images; cloud operators can create networks and routers; and so on.

The deployment of some OpenStack services require a file system back-end. This is the case for the storage services (*e.g.*, Cinder, Glance and Swift), but also to services like Nova, which require to instantiate disks to store the images of the VMs it manages. The back-end file system used is typically a shared file system or a distributed storage system that provides storage for the services. The next section describes the trade-offs between using centralized and distributed storage and also presents the state-of-practice of storage back-end and distributed storage used in OpenStack.

5 DISTRIBUTED STORAGE ON OPENSTACK

As an instance of a distributed system, OpenStack must cater to aspects like security, high availability, and fault tolerance. Also, as an a IaaS provider, it needs to orchestrate at least three resources: compute, network, and storage. Standard virtualized environments and several cloud environments often use centralized storage. In this scenario, each compute node accesses the centralized storage over the network using a protocol such as iSCSI or NFS. This is a common architecture that has been used for many years. The main problem here is the potential bottleneck of the centralized architecture if the system scale to a large number of clients. Of course, centralized storage is not the only choice available. In a local storage architecture, data is stored on hard disks within the compute nodes. This is relatively cheap as it requires no specific hardware, and failure occurrences are far less catastrophic. There is also no bottleneck of I/O in the network. Whatever architecture is chosen, the VM and the compute node become inevitably connected: the VM cannot run on another compute node because its virtual disk is stored within the source compute node. This presents several placement issues, and also prevents key operations such as VM live migration from working efficiently (*i.e.*, when live migrating, a VM disk would need to be copied through the network from a compute node to another).

Distributed storage intends to offer benefits of centralized storage with the cost and scalability of local storage. It is basically made up of a combination of many individual centralized storages, consisting of one or any number of physical disks, stored as objects. In this scenario, VM disks are split up into segments, and each segment is stored several times among the storage disks. A copy of a segment is called a replica. This model is designed to be fault-tolerant, as data is replicated across the storage cluster, and VM disks are not stored in the compute nodes themselves. Therefore, VMs can be live migrated between compute nodes at a feasible time. Moreover, VMs can be evacuated from a compute node in case of failure.

Ceph is an open source distributed, reliable, fault-tolerant, and massively scalable storage technology used in a variety of cloud orchestration systems [4]. It implements object storage on a distributed cluster, and provides interfaces for object-, block- and file-based storage under a unified system, called RADOS (Reliable Autonomic Distributed Object Store). Ceph is designed to run on commodity hardware[2]. It uses an algorithm called CRUSH (Controlled Replication Under Scalable Hashing) to ensure that data is evenly distributed across the Ceph cluster and all cluster object store devices (OSDs) can receive, as well as provide, data quickly. Ceph has three main components: a client, a cluster of OSDs, and a cluster of Monitors. Clients interact with Ceph through RESTful APIs, POSIX interfaces, and using `librados` which enables clients to create their own interface to Ceph. OSDs collectively store all data. Monitors manage the namespace (file names and directories) and regulate consistency, coherence, and security. Other metadata are stored in the OSDs. Figure 2 illustrates this behavior.

Ceph has been integrated with the Linux kernel, KVM and included by default in many GNU/Linux distributions. This increased overall interest in Ceph, which ended up in Ceph beeing the preferred and the state-of-practice solution for corporate and production OpenStack clouds [3].

Although it would be great if there was a one-size-fits-all storage architecture which is perfect for all use cases, there is not. Ceph also has some drawbacks to be considered. For instance, there is no active data deduplication [23], which leads to unnecessary storage consumption. Before BlueStore, Ceph had no copy-on-write (COW) functionality for RBD (RADOS Block Device) [21], which caused overhead when creating large block devices (*e.g.*, 200 GB in size) as the entirety of RAW data is needed to be pre-allocated and placed across the cluster. Another concern is that OSDs performance are highly affected if OSDs' physical disks are not formatted

[2]Commodity hardware is a device or component that is relatively inexpensive, widely available and more or less interchangeable with other hardware of its type.

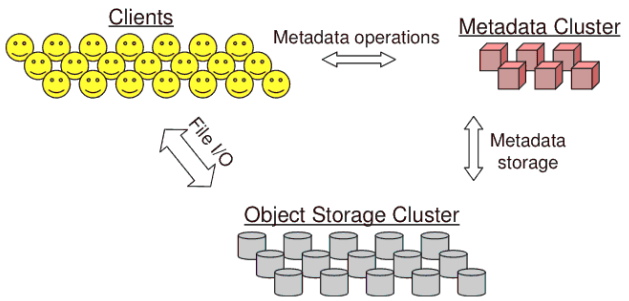

Figure 2: Ceph architecture. Clients perform file I/O by communicating directly with OSDs.

as XFS [16]. Also, topology changes on the cluster (*e.g.*, the addition of new OSDs) leads to intense data movement to maintain CRUSH contracts, which can cause overhead on the network [24]. Ceph also requires monitors deployed in an odd number to obtain a quorum and reduce the likelihood of data loss occurrences.

Storage systems involve trade-offs that we can think of as responses to their particular requirements and circumstances. For instance, some Ceph cons will not appear on other storage technologies. On the next section, we present BeeFS, a cheap, dynamic, naturally scalable, and distributed storage solution.

6 THE BEEFS FILE SYSTEM

BeeFS is "a distributed file system that essentially harnesses on the free disk space of machines deployed in a corporation to provide distributed storage" [20]. Following a hybrid architecture[3] with a centralised node, the *queen-bee*, to store and maintain metadata and file replication, distributed storage nodes, the *honeycombs*, collaboratively store data, which reduces overload on the central server. BeeFS "provides a global file namespace and location-transparent access to files which are stored in the disks of the participating machines" [20]. BeeFS count on the usage of commodity hardware, which enables honeycombs deployment on any desktop or rack- and blade-server with disk space to spare, and eases the scalable growth of the filesystem. The clients, called the *honeybees*, are served by the queen-bee and the honeycombs. A node can run both a client and a server component at the same time. BeeFS is fully POSIX-compliant, meaning it offers support for applications compatibility. Furthermore, the hybrid architecture facilitates the system design and enables simple administration on the entire file system. Unlike Ceph, BeeFS does not rely on a particular format of the storage servers disks: it can store data on top of any POSIX file system. The

[3]Hybrid architecture, in this case, relates to an implementation that mixes aspects of client-server and peer-to-peer systems.

coupling between the user level application and the Linux kernel file system modules is done via FUSE [13, 20].

Architecture overview

A typical BeeFS deployment comprises a single queen-bee, running on a dedicated machine, handling naming, metadata and replica management, providing global file namespace, access control, resource discovery and placement coordination services [20]; one or more honeycomb servers storing the data. The queen-bee does not store anything but metadata. The honeybees contact the queen-bee to obtain the location of honeycomb servers. Then, the honeybees are able to send/fetch data directly to/from the appropriate honeycomb server [20]. A data placement strategy tries to keep data as close as possible from its users [20], meaning nodes acting as a data server and client may read/write files stored in the local data server, which indicate a performance gain. Figure 3 presents the aforementioned design [20].

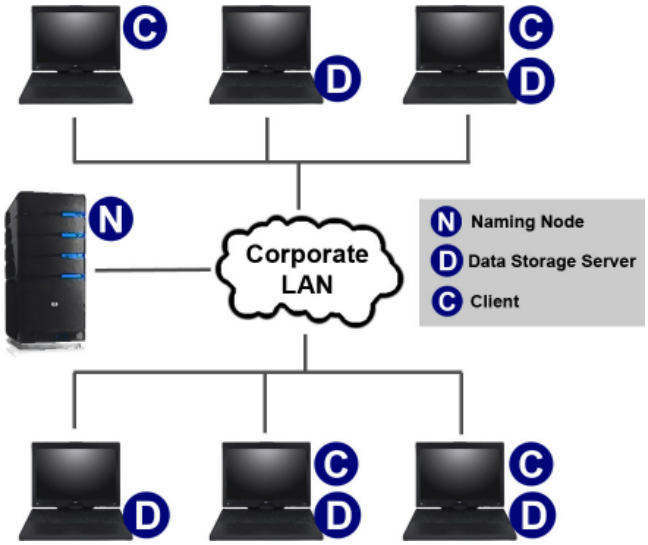

Figure 3: BeeFS Components overview.

BeeFS does not cache data on the client [20], but only on the honeycombs. Also, BeeFS maintains a single copy of a file at any time [20], which creates a strong consistency model. On the other hand, the absence of data-caching on clients may lead to to significant performance issues. This could be mitigated with a proper data placement mechanism forcing that only a tiny fraction of data access is performed in remote honeycombs [20].

When it comes to fault tolerance, BeeFS employs a replication model that focuses on the non-blocking model, which means honeybees perform blocking operations on the primary honeycomb and eventually the data replication/updates are persisted on the replicas. In that way, data is always consistent on the primary honeycomb and eventually consistent on the replicas. BeeFS does not implement erasure coding.

7 METHODOLOGY

We want to evaluate the performance of OpenStack Nova when using two different distributed storage solutions as its back-end: Ceph and BeeFS. Different solutions mean that Nova interacts differently with each one. For instance, when using Ceph as a back-end, the hypervisor (QEMU, libvirt) has its own RBD driver implementation to communicate with RADOS [5]. This architecture is shown in Figure 4 [18].

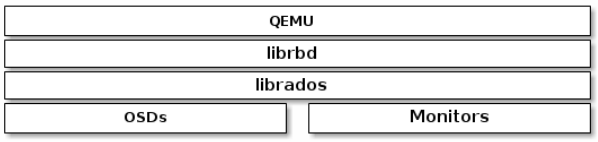

Figure 4: QEMU communication with RADOS.

One can say QEMU then is optimized for Ceph in comparison to QEMU interaction with BeeFS. On BeeFS, every QEMU operation on a VM virtual disk is captured by FUSE [20] and then forwarded to BeeFS honeybee component running on the client (Figure 5), which can lead to several performance penalties for QEMU operation. This process causes QEMU to be fooled into thinking that it is manipulating a local QCOW2 virtual disk, which is not true. QEMU, then, is not even aware of BeeFS existence: it keeps using its QCOW2 driver to manipulate a file stored the local file-system, when the file is actually stored across BeeFS honeycombs.

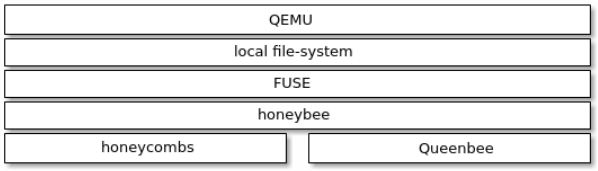

Figure 5: QEMU unaware communication with BeeFS.

System configuration and setup

For the experiment, we have used four physical machines HPE ProLiant BL460c Gen9 (Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 16GB RAM) as a modules of a Blade server providing 10Gb/s network interfaces through its backplane. All servers running Ubuntu server 16.04, and the network not isolated.

We deployed a small OpenStack cloud comprised of **one infrastructure node** and **one compute node**. OpenStack was deployed on the Rocky (stable/rocky) tag, the 18th release of OpenStack. The deployment was made through OpenStack-Ansible (OSA) [12].

Experiment design

To evaluate Nova performance with BeeFS (proposed) and Ceph (the state-of-practice), we defined two different scenarios: the **baseline** and the **optimized** scenarios.

Baseline scenario. The baseline is comprised of an unfeasible configuration for production environments, regarding performance and fault tolerance. This scenario uses three nodes, described as follows:

- Ceph:
(1) Ceph monitor
(2) Ceph OSD 1, comprised of 1 SAS HDD for Journal and 1 SAS HDD for data
(3) Ceph client

- BeeFS:
(1) Queen-bee server
(2) Honeycomb server
(3) A honeybee - client

Optimized scenario. The optimized scenario is comprised of a more feasible, although small, distributed storage deployment for production. Unlike the baseline, this scenario uses four nodes for Ceph and three nodes for BeeFS:

- Ceph:
(1) Ceph monitor;
(2) Ceph OSD 1, comprised of 1 SAS SSD for Journal and 1 SAS HDD for data;
(3) Ceph OSD 2, comprised of 1 SAS SSD for Journal and 1 SAS HDD for data;
(4) Ceph client.

- BeeFS:
(1) Queen-bee server;
(2) Honeycomb server - for replication;
(3) A honeybee - client; and a honeycomb server for primary storage.

On all configurations, Ceph monitor and BeeFS queen-bee run on the OpenStack infrastructure node; both Ceph client and BeeFS honeybee run on the compute node. This specifically means that the actual client of Ceph and the BeeFS is the hypervisor (*i.e.*, QEMU) running on the compute node. VMs were requested to launch on OpenStack by a user machine, one at a time, with a provided user-data script[4]. VMs ran on the compute node through QEMU, and it used the designated back-end to store the virtual disks (block device). All VMs were launched using a Ubuntu 16.04 server image previously deployed to Glance, the image service.

When a VM was ready, it sent a TCP package to the user machine who requested the VM, indicating:

- VM finished booting
- VM downloaded and installed packages to run the benchmark
- VM will begin to run the benchmark

We are then able to measure the total **boot time** to launch a VM (*i.e.*, time from launch request, through VM booting, to TCP package receipt). When a VM finishes its workload, it will send another package to the user machine indicating:

- VM finished the benchmark execution
- VM is ready to be deleted

Similarly to start time, we can measure the total **delete time** of a VM (*i.e.*, time from delete request, through VM resource deallocation, to TCP package receipt).

As part of the workload, VMs will run the **FIO** [1] benchmark, as it is a known and widely used benchmark for file system performance and cloud-based workloads [2]. The FIO benchmark suite can be installed with:

```
sudo apt update
sudo apt install fio
```

FIO will perform a workload of random writes of 1 Gib (~1074 MB) with the default block size of 4 KiB (4096 bytes), using the sync ioengine (basic write I/O manipulating file descriptors). Full `fio` configuration and parameters are the followings:

```
fio --name=benchmark --size=1G --filename=file
--ioengine=sync --randrepeat=0 --direct=1
--invalidate=1 --verify=0 --verify_fatal=0
--rw=randwrite --blocksize=4k --output output.txt
```

We created a flavor on the OpenStack that represents the VM hardware configuration. Flavors "define the compute, memory, and storage capacity of VMs. To put it simply, a

[4]According to OpenStack documentation, the user data is "a blob of data that the user can specify when they launch a VM. The VM can access this data through the metadata service or config drive. Commonly used to pass a shell script that the VM runs on boot".

flavor is an available hardware configuration for a server. It defines the size of a virtual server that can be launched" [11]. The flavor has 2 vCPUs, 1 GB RAM, and 5 GB of disk storage.

8 RESULTS AND DISCUSSION

For the performance evaluation of Nova we focused on measuring the followings:

- VM boot time
- VM delete time
- FIO runtime (in milliseconds)
- FIO average latency (in microseconds)
- FIO average bandwidth usage (in KB per second)

All the experiments ran separately, which means the environment changed before executing each one of the four possible scenarios. We collected 40 samples on each scenario, and in each sample we measured and registered the aforementioned metrics. We employed a percentile bootstrap of 10000 samples with replacement, with an alpha of 5% (α = 0.05 which implies on a confidence level of 95%), to obtain the median-unbiased as estimator.

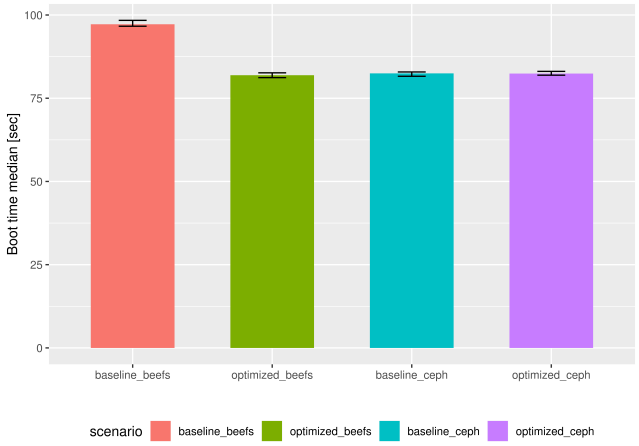

Figure 6: Boot time median (s)

We found that Nova performance is considerably affected when using BeeFS. For instance, VM boot time on *BeeFS optimized scenario* is compared to Ceph on *both scenarios* (Figure 6), in which Ceph needs to pre-allocate the entire VM disk (*i.e.*, 5 GB) across the RADOS cluster before running the VM on the compute node [19]. This means that a COW (copy-on-write) approach for image initialisation is performing as bad as an RAW disk image initialisation, when COW initialisation should perform so much better. On the other hand, deletion time does not seem to be a performance bottleneck for Nova neither using BeeFS or Ceph (Figure 7), as resource deallocation is pretty much the same for Nova

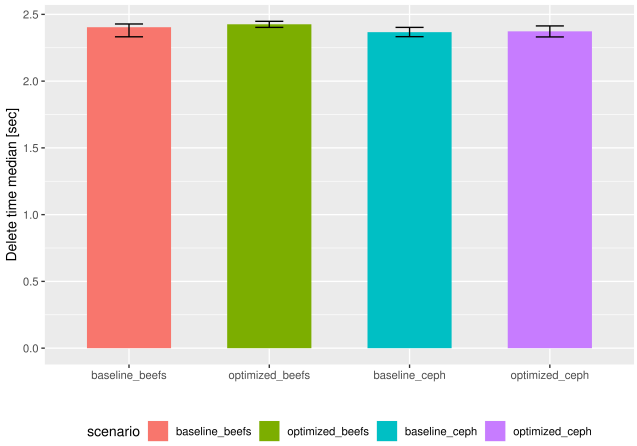

Figure 7: Delete time median (s)

on all scenarios (*i.e.*, stop the VM, delete network resources and delete virtual disk) and it only costs around 2.3 seconds on all scenarios.

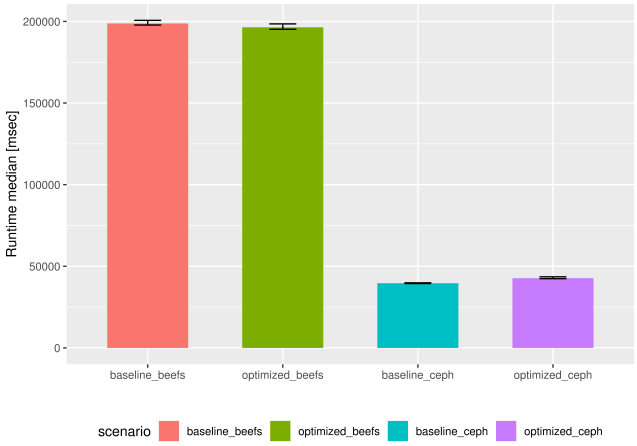

Figure 8: Runtime median (msec)

Regarding FIO benchmarking, Nova performs, in average, almost **5 times faster with Ceph** on both scenarios compared to *baseline and optimized* scenarios of BeeFS (Figures 8 and 9). It is clear that latency and runtime are extremely related: one can simply read runtime as the latency sum of all operations (*e.g.*, open, write) performed during the benchmark. We found that BeeFS is causing write bottlenecks on Nova, as one single *write* operation requested by a client potentially turns out on 3 more operations on data servers (*i.e.*, seek pointer position and open file, write, close the file), according to BeeFS implementation. Nova benefits from Ceph RBD, as Ceph uses a default writeback cache on the client [17] and QEMU communicates with RBD to only manipulate

objects/chunks of 4 MB in size. This can represent data inconsistency, but QEMU RBD driver implementation is aware of this behavior and, along with the host (*i.e.* compute node) kernel, they periodically execute barriers and flushes to send data to Ceph OSDs.

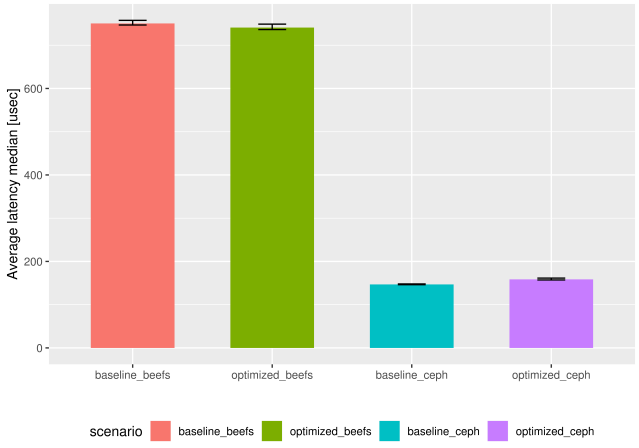

Figure 9: Average latency median (usec)

Considering the aforementioned behavior before, the outperform of Nova using Ceph against Nova using BeeFS is even more clear when we observe the average bandwidth used throughout the benchmark. With Ceph, Nova is able to write around to 25000 KB/s against only an average of 5000 KB/s when using BeeFS (Figure 10).

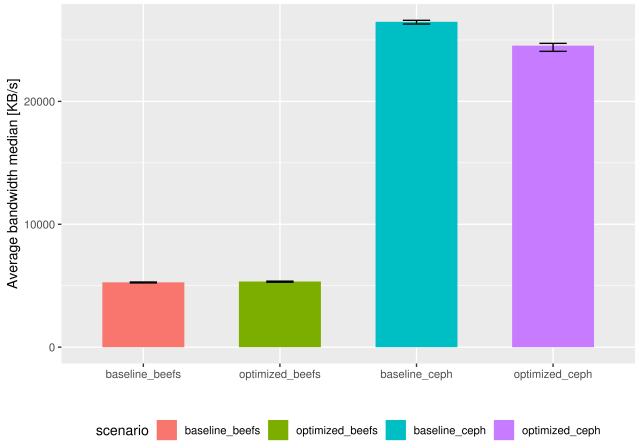

Figure 10: Average bandwidth median (KB/s)

However, we found out that Ceph performance is affected on the *optimized scenario* (*i.e.*, using two OSDs, and replicating all data within both of them). This happens because when a Ceph pool is configured to use replicas, a client (*i.e.*, Nova QEMU) only receives ACK of operations (*i.e.* write) when Ceph reproduces the same requested operation on all replicas [17]. This could lead to unwanted performance degradation if a Ceph pool is configured for replication with multiple OSDs, which could be addressed with a proper PG (Placement Group) configuration or even through the usage of erasure coding instead of replication.

9 CONCLUSION

This work examined the possibility to use the BeeFS distributed file system as back-end of OpenStack Nova virtual machine manager. Although would be desirable, it was not known that BeeFS could not outperform Ceph when serving as storage back-end for Nova, as Ceph RBD is *de facto* the best storage solution available on the market. Ceph easily increases Nova performance, through the usage of client cache and a consolidated data placement. However, it is not fair to compare BeeFS and Ceph RBD directly, as they are different solutions built for different purposes, and with different architectures each. For instance, QEMU is aware of how Ceph works, and it has a proper RBD driver optimized for communication with RADOS, when QEMU is not even aware of BeeFS existence.

In future, we would like to develop new functionalities and a proper QEMU driver focused for communication with BeeFS. Also, we want to present more deep analysis (*i.e.*, at lower levels) and performance comparison with other storage back-ends commonly used in OpenStack (*e.g.*, NFS, LVM). Furthermore, image and volume (*i.e.*, Cinder and Glance) service could also be used with BeeFS as the back-end, which would enable BeeFS as a feasible storage solution for OpenStack clouds.

REFERENCES

[1] Jens Axboe. 2017. fio - Flexible I/O tester. Retrieved June 16, 2019 from https://fio.readthedocs.io/en/latest/fio_doc.html

[2] binary lane. 2019. How to benchmark disk I/O. Retrieved June 16, 2019 from https://support.binarylane.com.au/support/solutions/articles/1000055889-how-to-benchmark-disk-i-o

[3] Ceph blog. 2018. Ceph User Survey 2018 results. Retrieved June 16, 2019 from https://ceph.com/ceph-blog/ceph-user-survey-2018-results/

[4] CEPH. 2019. Ceph storage. Retrieved June 16, 2019 from https://ceph.com/ceph-storage/

[5] Christian, Josh Brunner, Brunner, and Durgin Durgin. 2011. QEMU Block driver for RADOS (Ceph). Retrieved June 16, 2019 from https://github.com/qemu/qemu/blob/master/block/rbd.c

[6] Giacinto Donvito, Giovanni Marzulli, and Domenico Diacono. 2014. Testing of several distributed file-systems (HDFS, Ceph and GlusterFS) for supporting the HEP experiments analysis. *Journal of Physics: Conference Series* 513, 4 (jun 2014), 042014. https://doi.org/10.1088/1742-6596/513/4/042014

[7] DZone. 2016. OpenStack: Core Components. Retrieved June 16, 2019 from https://dzone.com/articles/openstack-core-components

[8] OpenStack Foundation. 2010. What is OpenStack? Retrieved June 15, 2019 from https://www.openstack.org/software/project-navigator/openstack-components#openstack-services

[9] OpenStack Foundation. 2012. HypervisorSupportMatrix - OpenStack. Retrieved June 16, 2019 from https://wiki.openstack.org/wiki/HypervisorSupportMatrix

[10] OpenStack Foundation. 2018. OpenStack Docs: Introduction: A Bit of OpenStack History. Retrieved June 16, 2019 from https://docs.openstack.org/project-team-guide/introduction.html

[11] OpenStack Foundation. 2019. OpenStack Docs: Flavors. Retrieved June 16, 2019 from https://docs.openstack.org/nova/rocky/user/flavors.html

[12] OpenStack Foundation. 2019. OpenStack Docs: OpenStack-Ansible Documentation. Retrieved June 10, 2019 from https://docs.openstack.org/openstack-ansible/rocky/

[13] FUSE. 2008. Fuse: Filesystem in userspace. Retrieved June 16, 2019 from http://fuse.sourceforge.net

[14] Johari Johanes, Khalid Joseph, Mohammad, Mydin Fairus, Mohd, and Mimos Berhad Nizam, Nazaruddin Wijee. 2014. Comparison of Various Virtual Machine Disk Images Performance on GlusterFS and Ceph Rados Block Devices. (2014).

[15] The Register. 2010. NASA and Rackspace open source cloud fluffer. Retrieved June 16, 2019 from https://www.theregister.co.uk/2010/07/19/nasa_rackspace_openstack/

[16] Inktank Storage. 2014. Hard Disk and File System Recommendations. Retrieved June 16, 2019 from http://docs.ceph.com/docs/jewel/rados/configuration/filesystem-recommendations

[17] Inktank Storage. 2014. librbd Cache Settings. Retrieved June 16, 2019 from http://docs.ceph.com/docs/jewel/rbd/rbd-config-ref/

[18] Inktank Storage. 2014. QEMU and Block Devices. Retrieved June 16, 2019 from http://docs.ceph.com/docs/jewel/rbd/qemu-rbd/

[19] Inktank Storage. 2014. RBD - Manage Rados Block Device (RBD) images. Retrieved June 16, 2019 from http://docs.ceph.com/docs/jewel/man/8/rbd/#striping

[20] Emmanuel Thiago, Jonhnny Pereira, Silva Alexandro Weslley, Francisco Soares, and Francisco Brasileiro. 2019. Beefs: A cheaper and naturally scalable distributed file system for corporate environments. (06 2019).

[21] Sage Weil. 2017. New in Luminous: BlueStore. Retrieved June 16, 2019 from https://ceph.com/community/new-luminous-blueStore/

[22] Sage A. Weil. 2007. *Ceph: Reliable, Scalable, and High-Performance Distributed Storage*. Ph.D. Dissertation. University of California, Santa Cruz, CA.

[23] Guido Winkelmann. 2013. Limitations of Ceph. Retrieved June 16, 2019 from http://lists.ceph.com/pipermail/ceph-users-ceph.com/2013-August/003942.html

[24] Maksym Yehorov. 2016. What are the advantages of using ceph? Retrieved June 12, 2019 from https://www.quora.com/What-are-the-advantages-of-using-ceph