FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Comparison of software defined storage with a classical enterprise disk array
Student:	David Šebek
Supervisor:	Ing. Tomáš Vondra, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2019/20

Instructions

1) Study the conceptual difference between enterprise SAN disk arrays and software-defined storage. Focus on the Ceph software.
2) Design a measurement methodology for comparing the performance of the two technologies of block storage.
3) From the supplied HPE servers and SAS disks, build a software-defined storage.
4) Compare performance with the supplied HPE 3PAR disk array with the same number of drives of equal class.
5) Discuss the strong and weak points of each system, mainly the tolerance for errors of varying magnitude and the avalibability/durability guarantees they can offer.
6) Describe the security best practices for these two systems when deployed in a public data center.

References

Will be provided by the supervisor.

prof. Ing. Pavel Tvrdík, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 30, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Comparison of software defined storage with a classical enterprise disk array

David Šebek

Department of Computer Systems
Supervisor: Ing. Tomáš Vondra, Ph.D.

May 15, 2019

Acknowledgements

Foremost, I want to thank my supervisor, Ing. Tomáš Vondra, Ph.D., for his willingness to help me solve various issues that arose during my work on this thesis, and for providing all technical equipment that was needed.

My thanks also go to everyone who has offered me the support of any kind, either directly by reviewing my written text passages, or indirectly by encouragement.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 15, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šebek, David. *Comparison of software defined storage with a classical enterprise disk array.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstract

This bachelor's thesis compares two storage approaches – SAN disk array, represented by an HPE 3PAR device, and software-defined storage built with Ceph software. The purpose of this comparison is to find whether a software-defined storage cluster built with regular servers is comparable performance-wise with a SAN disk array of similar hardware configuration.

After explaining the concepts of both types of storage and researching benchmark procedures, the author built a small Ceph cluster and measured its latency and throughput performance against a 3PAR disk array. The results revealed that 3PAR performed 31× better for 4 KiB data block writes compared to Ceph. On the contrary, Ceph cluster 1.4× surpassed 3PAR in 16 MiB large-block reads. This thesis is useful to data storage designers who may be deciding between the two storage technologies.

Keywords storage performance comparison, software-defined storage, enterprise data storage, storage benchmarking, SAN disk array, Ceph, 3PAR

Abstrakt

Tato bakalářská práce porovnává dva typy datových úložišť – diskové pole SAN, reprezentováno strojem HPE 3PAR, a softwarově definované úložiště postavené pomocí software Ceph. Cílem tohoto porovnání je zjistit, zda je softwarově definované úložiště postavené z běžných serverů výkonově srovnatelné s diskovým polem SAN podobné konfigurace.

Po vysvětlení konceptů obou typů úložišť a vyhledání informací k postupu měření výkonu, autor postavil malý Ceph cluster a změřil jeho latenci a datovou propustnost oproti diskovému poli 3PAR. Výsledky odhalily, že při zápisu malých 4KiB bloků dat byl 3PAR 31× rychlejší než Ceph. Při čtení velkých 16KiB bloků dat byl naopak Ceph 1.4× rychlejší. Tato práce je užitečná pro návrháře datových úložišť, kteří se rozhodují mezi těmito dvěma technologiemi datového úložiště.

Klíčová slova porovnání výkonnosti úložišť, softwarově definované úložiště, úložiště pro datová centra, měření výkonu úložiště, SAN diskové pole, Ceph, 3PAR

Contents

List of Figures

List of Tables

Introduction

Storage requirements grow as more and more data is being produced every day. Databases, backups, cloud storage, this all requires significant storage capacity. Moreover, the storage capacity needs to be easily expanded as the amount of data grows in time.

Historically, there have existed dedicated disk arrays for this purpose. One of the examples of disk arrays can be 3PAR products. These devices are purposely built with storage in mind. They offer redundancy and multiple RAID settings. The customer only needs to buy one or more of these devices, put some hard drives in them, configure the storage parameters and connect the device to the client servers. These devices are usually connected into a storage array network, SAN, using Fibre Channel Protocol. However, these proprietary solutions are not easily expandable and may require expensive licensing.

There is another approach that has emerged in recent years. It is software-defined storage. There is no need to buy an expensive storage device. Software-defined storage can be set up on regular servers. One of the examples is Ceph software. Ceph is an open source technology that can turn any conventional servers with conventional hard drives into a storage cluster. It is also designed to offer a vast amount of redundancy, depending on the number of devices used. The storage is also easily expandable if there is a need for adding more disk space. Because Ceph is open-source, there is no need to worry about not being able to use the storage after some license expires.

This thesis explores and compares these two approaches. Classical disk array, represented by a two-node 3PAR storage device, will be compared to a Ceph software-defined storage cluster made of two regular servers. The comparison will take into account the performance and strong and weak points of both systems. In order to make the comparison fair, the devices will be equipped with very similar hardware. Both Ceph servers will be similarly equipped as the two-node 3PAR device and will be configured to match as close as possible.

The findings of this thesis will help one cloud-providing company decide whether to invest in Ceph storage and base some of their offered services on software-defined storage. I chose this topic because I wanted to discover both technologies, disk array storage and software-defined storage, more in-depth. The documented experience and results of this work will also be beneficial to anyone who may be deciding which storage technology fits them the best.

The structure of this thesis is separated into two major parts – theoretical and practical. The theoretical part starts with an explanation of the architecture of the two storage technologies, represented by Ceph and 3PAR. Then it focuses on existing storage benchmarking procedures and storage benchmarking tools.

In the practical part, 3PAR device will be configured, and a Ceph storage cluster will be set up based on the researched information. The practical part of the thesis applies some of the researched benchmarking procedures on these two storage units. On successful completion, we will have enough data to evaluate the real storage performance. It will be possible to decide whether Ceph storage is a feasible option to replace 3PAR for cloud storage. The strong and weak points of the two systems, as well as their security settings, are mentioned.

Chapter 1

Goal

This thesis compares an HPE 3PAR disk array with a Ceph software-defined data storage.

The conceptual difference between enterprise SAN disk arrays (represented by 3PAR) and software-defined storage (Ceph) will be examined.

A measurement methodology for a performance comparison of the two technologies of block storage will be designed.

Software-defined storage will be built using two HPE servers with hard drives using SAS interface. Its performance will be compared to HPE 3PAR disk array with the same number of drives of the same class using the designed measurement technology.

Strong and weak points of each system will be discussed. These include the tolerance of errors of varying magnitude, and availability or durability guarantees they can offer. The best security practices will be mentioned for each of these two systems when deployed in a public data center.

Chapter 2

Analysis of Ceph and 3PAR

This chapter focuses on the conceptual principles of both of these storage concepts of two storage concepts – *software-defined storage* and *storage area network (SAN) disk array*. Two concrete examples were chosen to demonstrate each storage concept. *Ceph* software is used to represent software-defined storage, and an *HPE 3PAR StoreServ 7400c* system represents the concept of a SAN disk array. The main architectural building blocks of each storage approach will be examined as well as the way in which the data are stored.

2.1 What is 3PAR

HPE 3PAR is a brand name of hardware SAN storage array devices. Current models are sold under the Hewlett Packard Enterprise (HPE) brand [1].

"*A SAN (storage area network) is a network of storage devices that can be accessed by multiple servers or computers, providing a shared pool of storage space. Each computer on the network can access storage on the SAN as though they were local disks connected directly to the computer.*" [2]

Our HPE 3PAR 7400c model supports these types of connection to the client servers: Fibre Channel, iSCSI, and Fibre Channel over Ethernet (FCoE). The storage device can be managed using different software tools, such as *HP 3PAR Command Line Interface* (CLI), or the *HP 3PAR Management Console*. [3]

According to the QuickSpecs document [4], HPE 3PAR 7400c device supports these RAID levels:

- RAID 0,
- RAID 1,
- RAID 5 (data to parity ratios of 2 : 1 – 8 : 1),
- and RAID 6 (data to parity ratios of 4 : 2, 6 : 2, 8 : 2, 10 : 2 and 14 : 2).

The RAID sets are arranged in rows spanning multiple hard drives, forming RAID 10, RAID 50 or raid 60 [3].

2.2 3PAR Hardware Architecture

3PAR devices come in different hardware configurations. Hardware terminology mentioned in this section is similar to most models [3]. Figure 2.1 shows the hardware layout described in this section.

Physical hard drives are put in *drive magazines*. One magazine can hold multiple hard drives. These drive magazines are then inserted into a *drive cage*. Some 3PAR models use *drive enclosures* instead of drive cages. The difference between the two is that drive cage can hold multiple-disk drive magazines, while drive enclosure holds single-disk drive modules instead. Drive cages are then connected to the *controller nodes* using SAS interface. The nodes are then connected to the SAN, where client servers can connect to them. 3PAR storage system usually has multiple nodes for redundancy. [3]

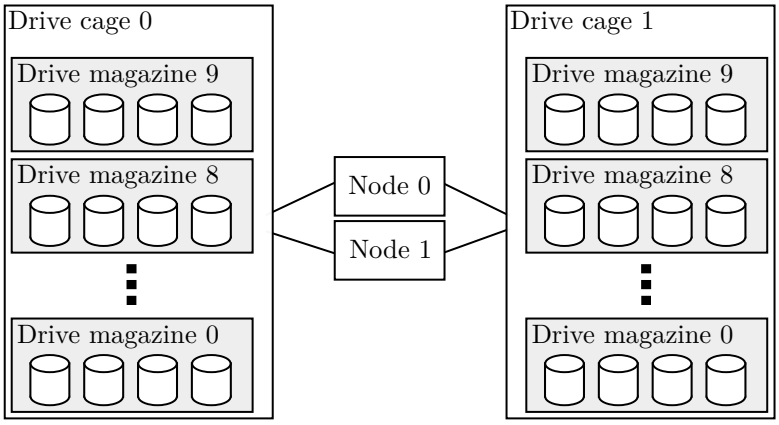


Figure 2.1: Hardware components of 3PAR. [3]

2.3 How 3PAR Stores Data

The HPE 3PAR storage consists of these five logical data layers, from the lowest level to the highest. Only the last layer, virtual volumes, is exported and visible to hosts (clients):

1. physical disks,
2. chunklets,
3. logical disks,

4. common provisioning groups (CPGs),

5. virtual volumes. [3]

The lowest layer, *physical disks*, consists of all physical hard drives connected to the system. Each disk that is added to the system is automatically divided into 256 MiB or 1 GiB units called *chunklets*. Chunklets from all physical disks are the building blocks for *logical disks*. Each logical disk is created from a group of chunklets using some user-specified RAID technique. This logical disk space is the base for creating virtual pools called *common provisioning groups*, or CPGs for short. From these common provisioning groups, individual *virtual volumes* can be created and exported to hosts. [3]

The only user-configurable layers are the CPGs and virtual volumes. Chunklets are created automatically by the 3PAR Operating System when the disk is added to the system. Logical disks are created automatically with the creation or growth of CPGs. [3]

Virtual volumes exported to the hosts can be of two types: *commonly provisioned* (allocated at creation, fixed size) or *thinly provisioned* (space is allocated on demand after creation). [3]

To form a logical disk, chunklets from different hard drives are arranged in RAID sets. These RAID types are supported by 3PAR:

- RAID 0,
- RAID 1 and 10,
- RAID 5 and 50,
- RAID Multi-parity (RAID 6).

RAID 0 offers no protection against hard drive failure, but it offers the best performance because the data is spread across multiple hard drives. RAID 1 mirrors the data across chunklets from multiple (2–4) hard drives and can sustain a failure of 1 or more hard drive (1–3). RAID 5 stores data across 2–8 chunklets from different hard drives, plus one chunklet with calculated parity. It can sustain a failure of one hard drive. RAID 6 works similarly to RAID 5, only it uses two chunklets for calculated parity and can sustain a failure of two hard drives. Data is stored in rows across chunklets. Each chunklet is divided into smaller parts with a configurable size of *step size*. [3] Simple illustrations of RAID 1 and RAID 5 data layout are in figures 2.2 and 2.3.

Not all of the chunklets are used for logical disks. There are always several chunklets on each drive used as spares. Additionally, 256 MiB of data on each drive is reserved for the table of contents (TOC). This TOC is the same on all drives. 4 MiB is reserved for diagnostic use. [3]

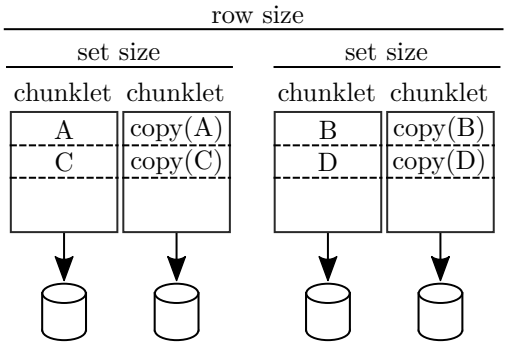


Figure 2.2: This picture illustrates how "ABCD" data block is stored across 4 chunklets on 3PAR using RAID 1 (2 data copies). [3]

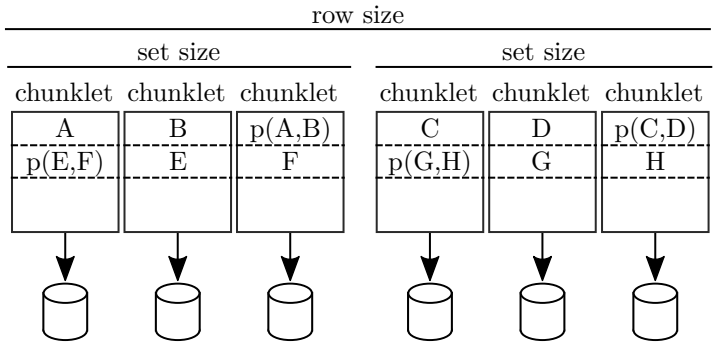


Figure 2.3: This picture illustrates how "ABCDEFGH" data block is stored across 6 chunklets on 3PAR using RAID 5 (2 data, 1 parity). [3]

2.4 What is Ceph

Ceph is open-source software released under GNU Lesser General Public License (LGPL), version 2.1 [5]. It is designed to work on Linux-based operating systems [6]. Its source code is written mostly in C++, Python and Terra [7].

What exactly is Ceph? According to the information on the main website, "*Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability*" [8].

"*Ceph provides highly scalable block and object storage in the same distributed cluster. Running on commodity hardware, it eliminates the costs of expensive, proprietary storage hardware and licenses. Built with enterprise use in mind, Ceph can support workloads that scale to hundreds of petabytes, such as artificial intelligence, data lakes and large object repositories.*" [9]

Ceph offers these types of storage in one unified system:

- object storage,
- block storage,
- file storage. [10]

2.5 Ceph Architecture

Ceph Storage Cluster typically consists of servers called *nodes*. "*A Ceph Node leverages commodity hardware and intelligent daemons, and a Ceph Storage Cluster accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically*" [10].

Ceph cluster can be built without having any single point of failure, offering redundancy at any level of operation. The cluster is based on Reliable Autonomic Distributed Object Store (RADOS) and is infinitely scalable. It consists of these two types of daemons:

- Ceph Monitors,
- Ceph OSD Daemons. [10]

According to the official documentation [11], these are the basic software daemons running in Ceph storage cluster, with their description:

MON (Monitor, *ceph-mon*)
: At least three monitors are usually required for redundancy. A monitor serves this purpose:

  - maintains maps of the cluster state required for the coordination of Ceph daemons,
  - and manages authentication between daemons and clients.

  "*Red Hat recommends deploying an odd number of monitors. An odd number of monitors has a higher resiliency to failures than an even number of monitors.* [...] *Summarizing, Ceph needs a majority of monitors to be running and to be able to communicate with each other, two out of three, three out of four, and so on.*" [12]

MGR (Manager, *ceph-mgr*)
: At least two managers are required for high availability. Their purpose is as follows:

  - keeps track of the runtime metrics and the state of the Ceph cluster (storage utilization, load, ...)
  - host modules to manage and expose Cluster information (Ceph Dashboard, REST API)

MDS (Metadata Server, *ceph-mds*)
: Stores metadata for Ceph File System (if used). It allows basic file system commands to be executed without a big performance impact on the Ceph storage.

OSD (Object Storage Daemon, *ceph-osd*)
At least three OSDs are required for redundancy and high availability.

- Stores data,
- handles data replication, recovery and rebalancing,
- provides monitoring information to Ceph Monitors and Managers.

The documentation [10] also desribes the principles of communication between the daemons. Ceph Monitor contains a master copy of the *Cluster Map*. Cluster Map contains the details about the cluster and its structure, including location of components such as Ceph Monitors, Ceph OSD Daemons, and other components in the cluster. The Cluster Map also contains the information needed to *calculate* the data location, not the location itself. Before each storage access, clients receive the recent copy of the cluster map from the monitor. Each Ceph OSD Daemon and Ceph Client uses a Controlled Replication Under Scalable Hashing (CRUSH) algorithm to compute the location of the data in the cluster and access that OSD daemon directly. There is no central lookup table or gateway for the data access. Ceph OSD Daemons also periodically perform data scrubbing, an activity, when they compare existing copies of the objects and try to find any inconsistencies.

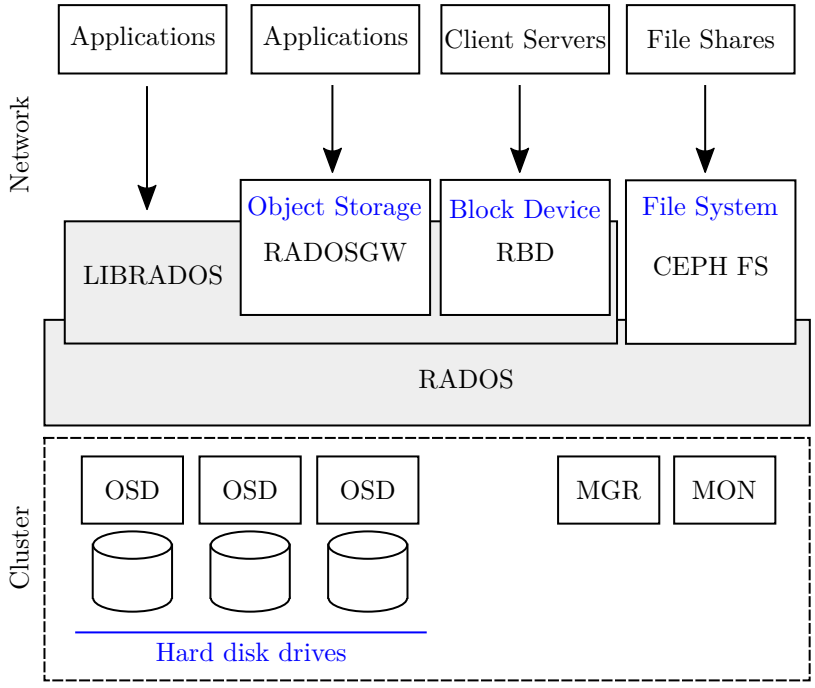


Figure 2.4: Ceph Architecture. [10], [13]

2.6 How Ceph Stores Data

"*Ceph stores data as objects within logical storage pools. Using the CRUSH algorithm, Ceph calculates which placement group should contain the object, and further calculates which Ceph OSD Daemon should store the placement group. The CRUSH algorithm enables the Ceph Storage Cluster to scale, rebalance, and recover dynamically.*" [11]

I would summarize the data layers for block device storage in Ceph as:

1. physical disks,
2. Ceph OSD Daemons,
3. placement groups,
4. objects,
5. pools,
6. RBD images.

2.6.1 Objects

Ceph stores all data as *objects*. Ceph OSD Daemons read or write the objects on the physical hard drives. The objects use flat namespace, not a directory hierarchy. Each object has an identifier, which is unique across the cluster, and contains binary data and metadata. Objects are replicated across multiple nodes by default, which provides a fault tolerance. [10]

Ceph Clients do not have to replicate the objects themselves. This is how a write operation is performed:

1. Client receives a copy of Cluster Map from the monitor,
2. client calculates the target OSD location and sends data to the OSD,
3. this primary OSD, which also has the Cluster Map, replicates the data to secondary, tertiary, ... OSDs;
4. primary OSD confirms the write. [10]

For the purposes of this thesis, the RBD block device functionality of Ceph was used. RBD stores data in objects, their size can be specified to be between 4 KiB and 32 MiB. The default object size for RBD is 4 MiB. [14]

2.6.2 Pools

Logical partitions for storing objects are called pools. Pools have a specified size (the number of object replicas), CRUSH rule, and number of placement groups. [10]

Ceph supports these two types of pools:

- replicated (used by default),
- erasure-coded.

In replicated pools, multiple copies of an object are stored. There is a total of n copies of each object. Each copy is stored on a different OSD. A simple illustration of a replicated pool is in figure 2.5. [15]

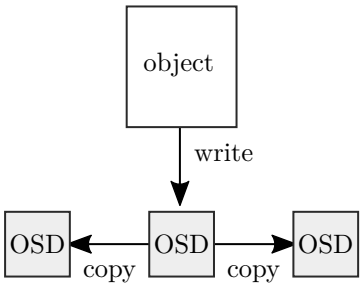


Figure 2.5: Write operation into a Ceph replicated pool with a size of 3 [10]

Erasure-coded pool splits an object into k data chunks, and m coding chunks are computed. Each of those chunks is stored on a different OSD. The pool uses k+m OSDs, up to m of them may become unavailable for the data to still be recoverable. Erasure-coded pools save space, but they need higher computational power. A simple illustration of an erasure-coded pool is in figure 2.6. [15]

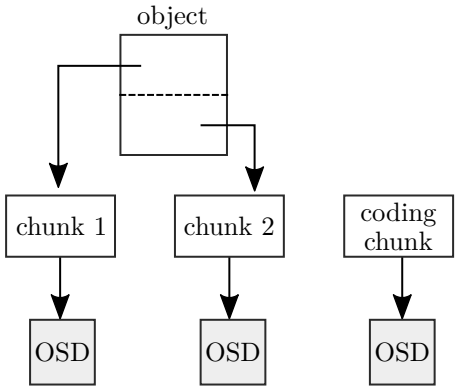


Figure 2.6: Ceph erasure coding ($k = 2$, $m = 1$) [10]

2.6.3 Object Striping

Similarly to how 3PAR stripes data across chunklets, Ceph also supports the concept of striping. In this case, objects are split into multiple stripes and the data is stored across multiple objects across different OSDs. Striping improves performance (similar to RAID 0) while keeping object mirroring. Note that Ceph Object Storage, Ceph Block Device, and Ceph File System by default do not stripe the objects, but stripe their data over the whole objects. [10]

2.6.4 Placement Groups

Perhaps one of the most confusing things when creating a pool is the number of placement groups, which has to be specified.

Objects in the pool are grouped into placement groups, because tracking each object in the cluster would be computationally expensive. Placement groups are mapped to OSDs. In a replicated pool of replication size n, each placement group stores its objects on n different OSDs. The placement groups are mapped to OSDs dynamically by the CRUSH algorithm. One OSD is not exclusively used by only one placement group – it can contain multiple placement groups. The number of PGs impacts the data durability, object distribution, and resource usage. Higher PG num results in more even data spread across all OSDs, but selecting a number that is too high results in more I/O operations during recovery, as there are more placement groups to move. The documentation recommends approximately 100 PGs per OSD. The number can be increased later, but not decreased. [16], [12]

"*For instance, if there was a single placement group for ten OSDs in a three replica pool, only three OSD would be used because CRUSH would have no other choice. When more placement groups are available, objects are more likely to be evenly spread among them. CRUSH also makes every effort to evenly spread OSDs among all existing Placement Groups.*" [16]

Below is a formula to calculate the number of placement groups *for a single pool*. If there are multiple pools created, the number of PGs needs to be changed accordingly.

$$TotalPGs = \frac{OSDs \times 100}{poolsize} \tag{2.1}$$

Where pool size is the number of replicas for replicated pools or K+M for erasure coded pools. The Total PGs result should be rounded up to the nearest power of two. [16]

From my experience, the PG number adds up with the creation of multiple pools, so one needs to know beforehand how many pools will be created to decide the best PG number.

The recently released new version of Ceph, Nautilus, seems to address many of these issues and supports automatic PG scaling and future decrementing. [17]

2.7 Ceph Hardware Requirements

Ceph requirements are decribed in detail in the official Ceph documentation page [18]. This is a short summary compiled from the documentation information. Note that each Ceph solution, such as Red Hat Ceph Storage or SUSE Enterprise Storage, may have slightly different requirements or supported CPU architectures [19], [13].

From the documentation it seems that Ceph is evolving very quickly and gaining new functionality each release. For this reason, the requirements may change between Ceph releases.

Ceph was designed to work on comodity hardware. This means that there is no need to buy any expensive specialized device, everything can be set up on regular servers. There are some recommendations for achieving the best cluster performance, such as running Ceph daemons on servers configured for the particular type of daemon. The number of servers used for each needs to be carefully decided based on the expected redundancy and the cost of the hardware. The documentation also recommends leaving Ceph cluster servers for Ceph only and using different host servers to access the data. [18]

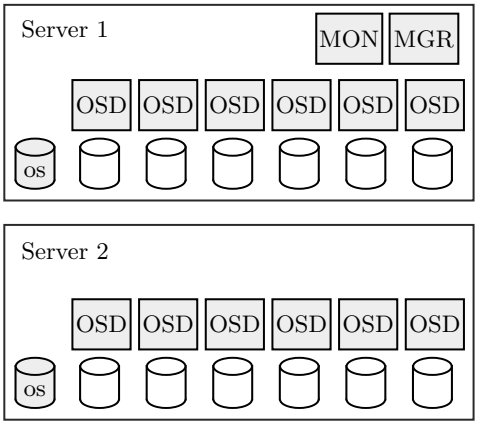


Figure 2.7: Two-node Ceph cluster. Each server has 6 hard drives, and runs a Ceph OSD Daemon for each drive. Server 1 also functions as a Ceph Monitor. This configuration will be used in the practical part of this thesis.

2.7.1 CPU

The documentation [18] mentions the following CPU requirements for each server running a Ceph service. Besides the CPU-intensive Ceph daemons, regular services and other software running on the system need some computational power too.

MDS Metadata servers should have significant processing power for load redistribution, *at least quad-core CPU is recommended.*

OSD OSDs run the RADOS service, calculate data placement with CRUSH, replicate data, etc. This requires CPU power. *At least dual-core CPU is recommended.*

MON Monitors *are not computation-intensive*, they only mantain their copy of the cluster map.

2.7.2 RAM

In general, approximately 1 GiB of RAM per 1 TiB of data storage is recommended. The memory requirements for each Ceph daemon are as follows.

MON, MGR The bigger the cluster, the more memory these daemons consume. *For small clusters,* 1–2 GiB *is sufficient.* The documentation mentions possibility of tuning the cache size use by these daemons, namely `mon_osd_cache_size` and `rocksdb_cache_size`.

MDS Metadata server's memory usage depends on its cache configuration. *At least* 1 GiB *is recommended.* Ceph allows the cache size to be set with the `mds_cache_memory` setting.

OSD *In default configuration, each OSD requires* 3–5 GiB *of memory.* The target memory amount (not necessarily the real amount) consumed by OSDs can be configured with `osd_memory_target` setting. Its default value is set to 4 GiB [20]. Note that this applies only to BlueStore OSDs. FileStore backend used in older Ceph releases behaves differently. [18]

2.7.3 Data storage

OSD OSDs are storing data. For best performance, *1 hard drive should be used for 1 OSD.* The documentation recommends hard disk size of at least 1 TiB each, for economic reasons – larger drives are more cost-effective. Although, with higher OSD capacity comes the need for higher RAM capacity. As mentioned previously, *approximately* 1 GiB *of RAM is recommended per* 1 TiB *of storage.*

Number of OSDs per host The throughput of all OSDs on the host should be lower than throughput of the network connection to the client (network should not be saturated). Also, a potential failure of the host should be taken into account. If all of the OSDs were on one host and it failed, Ceph would not be able to operate.

MON Each monitor daemon requires around 10 GiB of storage space.

MDS Each metadata server requires around 1 MiB of storage space. [18]

There will also need to be a drive where the operating system is installed. It is not a good idea to use the same drive for multiple purposes, such as having two OSDs on a single drive. Even though it is possible, the performance would suffer.

One might consider using SSDs instead of HDDs for Ceph storage. The documentation warns about potential issues, such as much higher price of SSDs, bad performance of cheap SSD drives, and that SSDs may require some SSD-specific precautions, such as proper partition alignment. It might be advantageous to use an SSD as a journal for the OSDs, not for the OSD storage itself. [18]

2.7.4 Network

The minimum bandwidth of the network depends on the number of OSDs in each server. The total OSD throughput should not exceed the throughput of the network. For faster data migration between cluster servers and avoiding possible Denial of Service (DoS) attacks, it is advantageous to have two networks for Ceph cluster:

- *front-side* network where clients can connect,
- and *back-side* network (disconnected from the Internet) for data migration between Ceph cluster nodes. [18]

2.7.5 Operating System

The offical documentation [6] gives the following recommendations for the operating system:

- deploy Ceph on newer releases of Linux,
- long-term support Linux releases are recommended.

The documentation recommends stable or long-term maintenance kernels. At least the following Linux kernel versions are recommended, older releases may not support all CRUSH tunables:

- 4.14.*z*,
- 4.9.*z*.

The official tool for Ceph installation, *ceph-deploy*, is written in Python programming language [21] and supports these Linux distributions: Ubuntu, Debian, Fedora, Red Hat, CentOS, SUSE, Scientific Linux, and Arch Linux [22]. The GitHub repository suggests that there is also a support for ALT Linux [23].

The Ceph documentation [24] uses the upstream Ceph repository for package installation. Both *deb* and *rpm* packages of various Ceph releases for

various processor architectures are available there. These provided packages support Debian, Ubuntu, RHEL, and CentOS distributions. The *rpm* repository does not support openSUSE which is said to already include recent Ceph releases in its default repositories.

Because of its open-source nature, many other Linux distributions may already include some (not necessarily the latest) version of Ceph in their repositories. Ceph source code can also be downloaded and then compiled on most Linux distributions. If those distributions are not supported by the ceph-deploy tool or any other alternative tools, it is always possible to install Ceph manually using instructions from the Ceph documentation [25].

2.8 Ceph Releases

Ceph is open source software. There are also other, usually commercial, products built on Ceph. This section mentions these different Ceph distributions.

2.8.1 Upstream Ceph

The release cycle of Ceph is explained in the official documentation page [26]. The current numbering scheme is this: *x.y.z*, where *x* is the release cycle (there is a new release every 9 months), *y* is the release type described below, and *z* is for bug-fix point releases.

- *x*.0.*z* for development releases,
- *x*.1.*z* for release candidates,
- *x*.2.*z* for stable or bugfix releases (stable point release every 4 to 6 weeks).

Additionally, each release cycle is named after a species of cephalopod. The name begins with *x*-th letter in the alphabet, where *x* is the release cycle number. For example, current stable release, as of wrtiting this paragraph (April 2019), is a newly released *14.2.0 Nautilus* [17]. Before that, there was *13.2.5 Mimic*.

List of currently supported Ceph release cycles with their names (as of April 2019):

- 14.2.z Nautilus,
- 13.2.z Mimic,
- 12.2.z Luminous (soon to reach end of life [EOL]). [26]

2.8.2 Upstream Ceph Support, Upgrades

There is a stable point release every 4 to 6 weeks. Each release cycle receives bug fixes or backports for two full release cycles (two 9-month release cycles, 18 months in total). An online upgrade can be done from the last two stable releases or from prior stable point releases. [26]

2.8.3 Other Ceph Solutions

It is totally possible to use the upstream open-source Ceph version and set it up by yourself. But there are multiple Ceph solutions for customers who wish to get Ceph with some kind of professional support. They are based on the same Ceph software and may provide additional integration tools and support. Some of the most popular solutions are offered by SUSE and Red Hat. [27]

- SUSE Enterprise Storage
- Red Hat Ceph Storage

Both Red Hat and SUSE are among the top Ceph contributors. These two companies were by far the most active contributors during the Ceph Mimic release cycle, both by number of repository commits and by lines of code. [28]

Chapter 3

Related Work

Before performing the actual performance measurement of our storage devices, the measurement methodology needs to be decided. This chapter researches some of the approaches to storage performance measurement. The emphasis will be given to Ceph performance measurement, as well as to the comparisons of Ceph with other types of storage. Some of the discovered approaches and software tools will be considered for my Ceph vs. 3PAR comparison.

3.1 Approach to Benchmarking a Ceph Cluster

Ceph wiki pages contain an article [29] with some benchmarking recommendations. The article recommends starting with the measurement of each individual storage component. Based on the results, one can get an estimate of the maximum achievable performance once the storage cluster is built. Namely, the article recommends measuring performance of the two main components of the Ceph infrastructure:

- disks,
- network.

As the simplest way of benchmarking *hard disk drives*, the article uses the *dd* command with the `oflag` parameter to avoid disk page cache. Each hard drive in the cluster would be tested with the following command, which will output the transfer speed in MiB/s.

```
$ dd if=/dev/zero of=here bs=1G count=1 oflag=direct
```

To measure *network throughput*, the article recommends the *iperf* tool. This tool needs to be run on two nodes, one instance in *server mode* and another instance in *client mode*. It will report the maximum throughput of the network in Mbit/s.

```
$ iperf -s
$ iperf -c 192.168.1.1
```

Once the disks and network are benchmarked, one can get an idea of what performance to expect from the whole Ceph cluster.

Ceph cluster performance can be measured at different levels:

- low-level benchmark of the storage cluster itself,
- higher-level benchmark of block devices, object gateways, etc.

Ceph contains a *rados bench* command to benchmark a RADOS storage cluster. The following example commands create a pool named *scbench*, run a write benchmark for 10 seconds followed by sequential read and random read benchmarks.

```
$ ceph osd pool create scbench 100 100
$ rados bench -p scbench 10 write --no-cleanup
$ rados bench -p scbench 10 seq
$ rados bench -p scbench 10 rand
$ rados -p scbench cleanup
```

The number of concurrent reads and writes can be specified with the `-t` parameter, size of the object can be specified by the `-b` parameter. To see how performance changes with multiple clients, it is possible to run multiple instances of this benchmark against different pools. [29]

To benchmark Ceph block device (RBD), the article [29] suggests the Ceph's *rbd bench* command, or the *fio* benchmark, which also supports RADOS block devices. First, they created a pool named *rbdbench*, and then an image named *image01* in that pool. The following example generates sequential writes to the image and measures the throughput and latency of the Ceph block device.

```
$ rbd bench-write image01 --pool=rbdbench
```

The second mentioned way of measuring the Ceph block device performance is through the *fio* benchmarking tool. Its installation contains an example script for benchmarking RBD. The script needs to be updated with the correct pool and image names and can be executed as:

```
$ fio examples/rbd.fio
```

The article [29] also recommends dropping all caches before subsequent benchmark runs. The same instruction is also mentioned in the Arch Wiki [30].

```
$ echo 3 | sudo tee /proc/sys/vm/drop_caches && sync
```

3.2 Commonly Used Benchmarking Tools and Their Parameters

This section contains results of my research to discover some of the commonly used software tools and practices for storage benchmarking. The findings will reveal that most of the storage benchmarking tools usually offer (and often require) a wide array of parameters to be specified, instructing it how to access the storage. These options usually include

- the storage operation performed (read/write),
- sequential or random access,
- the amount of data transferred in one access request (block size),
- number of requests issued at the same time (I/O queue depth),
- the number of requests performed or the total amount of data transferred (size),
- whether the storage should be accessed using a cache buffer of the operating system or whether they should be sent to the storage directly (buffered/direct).

For this reason, my research focused not only on the commonly used benchmarking tools, but also which parameters are the tools run with. Some of the benchmarking tools that might be useful for the practical part of this thesis and their possible parameters will be described at the end of this chapter.

3.2.1 Example 1

In a Proxmox document [31], there is Ceph storage benchmarked under different conditions and configuration. Among other things, the authors discovered that at least 10 Gbit/s network speed is needed for Ceph, 1 Gbit/s is significantly slower. The raw result performance numbers are to no use for us because they used different hardware, but we can at least see how they approached the benchmarking.

They also recommend testing individual disk write performance first. They used the following *fio* command to measure the write speed of individual disks (solid-state drives in their case).

```
fio --ioengine=libaio --filename=/dev/sdx --direct=1 --sync=1 --rw=
    write --bs=4K
--numjobs=1 --iodepth=1 --runtime=60 --time_based --group_reporting
    --name=fio
--output-format=terse,json,normal --output=fio.log --bandwidth-log
```

The command instructs *fio* to perform a sequential write with directly to a block device. They chose to write the data in 4 KiB blocks, one by one, without using any I/O request queue or any memory buffers of the operating system. It also creates a bandwidth log.

- *libaio* I/O engine;
- a block size of 4 KiB;
- direct I/O with a queue depth of 1;
- only 1 simultaneous job;
- run time of 60 s.

Read and write benchmarks were performed using the built-in Ceph benchmarking tools. Each test runs for 60 s and uses 16 simultaneous threads to first write, and then read, objects of size 4 MiB.

```
rados bench 60 write -b 4M -t 16
rados bench 60 read -t 16 (uses 4M from write)
```

3.2.2 Example 2

Another example of a performed benchmark is in a Red Hat Summit video [32, 9:10]. The authors also use *fio*, version 2.17, with these parameters:

- *libaio* I/O engine;
- a block size of 4 KiB for random access;
- a block size of 1 MiB for sequential access;
- direct I/O with a queue depth range of 1–32;
- write : read ratios of 100 : 0, 50 : 50, 30 : 70, 0 : 100;
- 1–8 jobs running in parallel;
- run time of 60 s ramp, 300 s run.

3.2.3 Example 3

Third example is from one of the OpenStack videos [33, 8:04], where *fio* benchmarking scripts can be seen. For latency testing, they measured

- 4 KiB random writes,
- I/O queue depth of 1,

- direct I/O flag.

For IOPS measurement, they use

- 4 KiB random writes,
- IO queue depth of 128,
- direct I/O flag.

In both cases, *aio* I/O engine is used.

3.2.4 Example 4

The last example is from a cloud provider that has a tutorial [34] describing "*how to benchmark cloud servers*". To benchmark storage, they use *fio* and *ioping* utilities. *Fio* is used to measure the throughput, *ioping* to measure the latency.

Random read/write performance:

```
fio --name=randrw --ioengine=libaio --direct=1 --bs=4k --iodepth=64
    --size=4G --rw=randrw --rwmixread=75 --gtod_reduce=1
```

Random read performance:

```
fio --name=randread --ioengine=libaio --direct=1 --bs=4k --iodepth
    =64 --size=4G --rw=randread --gtod_reduce=1
```

Random write performance:

```
fio --name=randwrite --ioengine=libaio --direct=1 --bs=4k --iodepth
    =64 --size=4G --rw=randwrite --gtod_reduce=1
```

Latency with *ioping*:

```
ioping -c 10 .
```

3.2.5 Summary of Discoveries

To sum up the findings, most researched Ceph and other storage benchmarks use *fio* tool with the specification of

- the *libaio* I/O engine,
- direct flag,
- random read or write with block size of 4 KiB (larger block size for sequential access),
- and I/O depth of 1 for latency measurements or larger for IOPS measurement.

Storage latency can also be measured with *ioping* utility.

3.2.6 Existing Ceph Latency Result

StorPool Storage presented their benchmark results at OpenStack Summit in Berlin, 2018. During a quick demo, they used the *fio* tool to benchmark individual disks. They measured storage latency using the PostgreSQL database benchmarking tool, *pgbench*.

They discovered the Ceph latency to be significantly higher than the latency of other storage solutions. They ran the tests on virtual machines from different cloud providers. Each virtual machne was of these specifications: 16 vCPUs, 32 GiB of RAM. The test consisted of *pgbench* random read/write (50 : 50) runs with queue depth of 1. Database size was chosen to be 4x the size of RAM. [33] Table 3.1 contains the average measured latency [33, 15:10].

Table 3.1: Average latency running pgbench on different cloud virtual machines. StorPool presentation at OpenStack Summit in Berlin 2018

DigitalOcean (Ceph)	1.97 ms
MCS.mail.ru (Ceph)	4.41 ms
DreamHost/DreamCompute (Ceph)	5.48 ms
AWS EBS gp2 10k	0.29 ms
StorPool NVMe/Hybrid 20k	0.14 ms

3.3 Storage Benchmarking Tools

In this section, some of the previously researched storage benchmarking software tools will be explored more in depth. The purpose of these tools is to measure the performance of data storage. Their simple usage will be demonstrated, and some of their options that might be useful for the practical part of this thesis will be briefly explained. All of the benchmarking tools mentioned in this chapter are free, command-line programs that can be run on Linux.

These storage benchmarking tools generate load on the storage, and then they usually report some metrics by which storage performance is described. These metrics usually are:

- amount of data transferred per second, or throughput, (usually in KiB/s or MiB/s);
- the number of input/output operations per second (IOPS);
- time required to transfer a block of data – latency (usually in ms).

Both IOPS and latency values depend on the amount of data transferred during one I/O operation. For this reason, the IOPS value and latency value

by themselves do not hold any information unless the size of a transfer block is known. For example, the transfer of a single 4 KiB data block takes less time than transfer of a 16 MiB data block (lower latency). This means that more 4 KiB I/O operations will fit in a one second time frame compared to 16 MiB operations, resulting in higher reported operations per second (IOPS) value.

3.3.1 The *dd* tool

The primary purpose of the command-line *dd* tool is copying files – "*dd copies a file* [...] *with a changeable I/O block size, while optionally performing conversions on it*" [35]. I am not aware of having used any Linux operating system where *dd* was not available. For this reason, the tool can be used as a quick measure of storage performance before installing some more sophisticated benchmarking software. On the Linux operating systems that I used during the writing of this thesis, *dd* command was installed as a part of the *coreutils package*.

As a simple write performance test of the storage, one can copy a sequence of zeros from the `/dev/zero` generator either into a file stored on the device (for example `/mountpoint/file.out`), or to the device itself (for example `/dev/sda`). Note that writing a sequence consisting solely of zeros does not always yield realistic write performance results if the storage device utilizes some form of data compression or de-duplication.

This tool can be used also as a simple test of the read speed of the storage. In this case the data can be copied from the device and discarded immediately afterwards (into `/dev/null`). Note that the data may already be cached in the system's memory, in which case the resulting transfer speed would not correspond with the real transfer speed of the storage device. This can be fixed either by flushing the system's cache beforehand or by instructing `dd` to read the data directly from the device and avoid any system cache. [30]

3.3.1.1 Command-line Parameters

The GNU coreutils documentation [35] describes many parameters and flags that instruct *dd* how to access files. These parameters specify the source and destination file and the size of the transfer:

`if=`*`filename`* Reads data from *filename* file.

`of=`*`filename`* Writes data to *filename* file.

`bs=`*`number`* The block size.

`count=`*`number`* Transfers *number* of blocks in total.

Below is a short list of flags that I personally found useful for storage benchmarking purposes. Many of the `oflag` output flags can also be applied as `iflag` flags for input files. Not all storage technologies support all of these flags, though.

`conv=fdatasync` "*Synchronize output data just before finishing.*" With this option, *dd* uses system buffer cache and then calls sync *at the end of the transfer.*

`conv=fsync` Similar to `fdatasync`, also includes metadata.

`oflag=dsync` "*Use synchronized I/O for data.*" This option uses system buffer cache, but it forces the physical write to be performed *after each data block is written.*

`oflag=sync` "*Use synchronized I/O for both data and metadata.*"

`oflag=direct` "*Use direct I/O for data, avoiding the buffer cache.*" This flag avoids operating system's memory subsystem and writes data directly to the destination device. [35]

3.3.1.2 Examples

The first command is a write performance benchmark. This command writes a total of 4 MiB of zeros to the `/dev/sdc` block device in 4 KiB blocks, while avoiding buffer cache of the operating system. It overwrites any previously stored data within the first 4 MiB of its capacity. After it finishes, it shows the achieved transfer speed. The second command can be used as a read performance benchmark. It is similar to the previous command, but it transfers data in the opposite direction. Data is read from the block device and discarded (not stored).

```
# dd if=/dev/zero of=/dev/sdc bs=4k count=1000 oflag=direct
# dd if=/dev/sdc of=/dev/null bs=4k count=1000 iflag=direct
```

3.3.2 IOzone

IOzone is another command-line tool. According to its official website, "*IOzone is a filesystem benchmark tool*" [36]. From my experience, not only *IOzone* does not come pre-installed on most major Linux distributions by default, but some of the distributions do not provide the package in their repositories at all. As of writing this paragraph (April 2019), current versions of Ubuntu and openSUSE include it in their repositories, Fedora does not. However, because it is open-source software [37], the source code can be downloaded and compiled manually.

3.3.2.1 Command-line Parameters

Its documentation [38] describes all of the operations this tool supports. For a quick performance evaluation of the storage, two command-line options are especially useful:

`-a` This switch runs an automatic test. It runs all file operations supported by *IOzone* (such as read, write, random read, random write, …) for file sizes of 64 KiB to 512 MiB and record sizes of 4 KiB to 16 MiB.

`-I` Uses direct I/O, thus bypassing the buffer cache and reading (writing) directly from (to) the storage device.

3.3.2.2 Example

A single automatic test can be run with this command:

```
$ iozone -a -I > program-output.txt
```

3.3.2.3 Plot Generation

There are also scripts available to generate a plot from the measured results. These scripts can be downloaded from the IOzone source code directory on their website [39]. These scripts may also come installed with the IOzone package, on openSUSE they are stored in `/usr/share/iozone`.

- `Gnuplot.txt` (contains the information about the scripts)
- `Generate_Graphs` (main script)
- `gengnuplot.sh` (generates a single plot)
- `gnu3d.dem` (generates all plots and displays them in gnuplot)

This command will use the scripts to generate three-dimensional plots showing the bandwidth based on file size and record size.

```
$ ./Generate_Graphs program-output.txt
```

3.3.3 fio

Another command-line tool is *fio*, which stands for flexible I/O tester. My research results mentioned earlier in this chapter show *fio* to be a popular tool for measuring storage performance.

"*Fio spawns a number of threads or processes doing a particular type of I/O action as specified by the user. fio takes a number of global parameters, each inherited by the thread unless otherwise parameters given to them overriding that setting is given. The typical use of fio is to write a job file matching the I/O load one wants to simulate.*" [40].

3.3.3.1 Command-line Parameters

Fio is a complex benchmarking tool which supports many options. Some of what I think are the most important *fio* options are be mentioned below. The complete list of supported parameters can be found in the documentation [40]. *Fio* jobs can be specified in a text file. It is also possible to pass the options directly on the command line. For example, the documentation mentions that "*for the job file parameter* `iodepth=2`*, the mirror command line option would be* `--iodepth 2` *or* `--iodepth=2`." On command line, jobs can be separated with the `--name` option. [40]

This group of *fio* command-line parameters specifies the compulsory job name and, if necessary, also specify the files or devices to perform the I/O operations with.

`--name=`*`job_name`* *Fio* does not run without this parameter. It specifies the name of the job to *job_name*. It may affect the names of output files.

`--filename=`*`file1`*`:`*`file2`* Explicitly specifies one or more file or device names to be used for I/O operations. If omitted, files will be generated automatically by job specification and stored in the current working directory.

`--directory=`*`dirname`* Puts the test and output files in the `dirname` directory instead of using the current working directory.

The following group of command-line parameters can be used to specify the action to be performed with the files or devices.

`--ioengine=`*`engine`* Some of the useful *engine* values are:

- `psync` *Fio* will use *pread(2)* or *pwrite(2)* calls. Default on most systems.
- `libaio` Linux native asynchronous I/O.
- `posixaio` POSIX asynchronous I/O.
- `windowsaio` Windows native asynchronous I/O. Default on Windows.
- `rados` Direct access to Ceph RADOS via librados.
- `rbd` Direct access to Ceph Rados Block Devices (RBD) via librbd. There is no need for a rbd driver in the kernel of the operating system.

`--direct=`*`1 or 0`* Whether to use non-buffered I/O (usually `O_DIRECT`).

`--rw=`*`operation`* Type of I/O pattern. Some of the useful values of *operation* are `read` or `write` for sequential access, `randread` or `randwrite` for random access, `rw` or `randrw` for mixed reads and writes.

`--rwmixread=`*`1 or 0`* **or** `--rwmixwrite=`*`1 or 0`* Percentage of reads (writes) in mixed read/write workloads (e.g. *int*=50).

--end_fsync=*1 or 0* Sync data after the complete write stage is finished (at the end of the job).

--fsync_on_close=*1 or 0* Sync data after every file is closed.

The following parameters specify the transfer size, block size, number of concurrent jobs, etc. of the performed I/O operations.

--size=*bytes* How many bytes each *fio* thread will transfer. If not specified, *fio* will use the full capacity of the file or device specified in a parameter, if it exists.

--bs=*bytes* The block size in bytes, defaults to 4096. Separate values for reads and writes can be specified, e.g. `bs=8k,32k` for 8 KiB reads and 32 KiB writes.

--iodepth=*number* "*Number of I/O units to keep in flight against the file.*" How many requests are sent sequentially to the storage.

--numjobs=*number* The job is cloned *int*-times into separate processes or threads.

--group_reporting if `--numjobs` is used, report as a whole, not per job.

--runtime=*time* Limits the execution time of *fio* to *time* seconds.

3.3.3.2 Plot Generation

This group of parameters instructs *fio* whether to save the results. It can generate log files that can be turned into plots.

--output=*filename* Writes the stdout output to a *filename* file.

--write_bw_log=*name* Stores the bandwidth log in files with a *name* prefix.

--write_lat_log=*name* Stores the latency log in files with a *name* prefix.

--write_iops_log=*name* Stores the IOPS log in files with a *name* prefix.

--log_avg_msec=*int* "*Setting this option makes fio average the each log entry over the specified period of time, reducing the resolution of the log.*" "*By default, fio will log an entry in the iops, latency, or bw log for every I/O that completes.*"

--log_hist_msec=*int* More accurate for generating completion latency histograms than the previous option. [40]

3.3.3.3 Examples

The first command measures the sequential write performance of the `/dev/sdc` block device in 4 KiB blocks, while avoiding buffer cache of the operating system. It writes 4 MiB of data in total. It overwrites any previously stored data within the first 4 MiB of its capacity. The second command tests the sequential read performance with the same parameters.

```
$ sudo fio --name=test --filename=/dev/sdc --rw=write --bs=4k --size
    =4M --direct=1
$ sudo fio --name=test --filename=/dev/sdc --rw=read --bs=4k --size
    =4M --direct=1
```

3.3.3.4 Client/Server

Fio can be run remotely from another computer. This is useful for performing simultaneous shared storage accesses from multiple clients.

On destination machine, *fio* can be started in server mode using the following command. By default, *fio* will be listening on all interfaces, default port 8765. [40]

```
$ fio --server
```

Remote *fio* client can connect to one or multiple *fio* servers. From my experience, job file had to be used instead of passing all job arguments as the command parameters, which did not work.

```
$ fio <local-args> --client=<server> <remote-args> <job file(s)>
```

Multiple servers can be accessed simultaneously.

```
$ fio --client=<server1> <job file(s)>  --client=<server2>  <job
    file(s)>
```

3.3.3.5 A Note on *fio* Benchmark Units

The output of *fio* version 2.2.10 displays units such as `KB/s`, `MB/s` etc. The manual page for this program version [41] indicates that, by default, these are base-2 units (1 KB = 2^{10} B = 1024 B).

Current *fio* version, version 3.13, displays units correctly (KiB, MiB, etc.). Input values can also be suffixed with the unit, not case sensitive. It allows the choice between the old *fio* unit specification and the correct units.

`--kb_base=`*`1000`* `4ki` equals to 4096 bytes, `4k` equals to 4000 bytes.

`--kb_base=`*`1024`* (default for compatibility reasons) `4k` equals to 4096 bytes, `4ki` equals to 4000 bytes.

3.3.4 IOPing

"*A tool to monitor I/O latency in real time. It shows disk latency in the same way as ping shows network latency.*" [42] Its manual page [43] describes all parameters.

Ioping requires at least one command-line argument: the destination storage location. It can be either a directory, a file, or a device. If a directory target is specified, *ioping* creates a temporary file `ioping.tmp` and uses this file for subsequent measurements. If a file or device is specified, it uses the whole size of the file or device to perform measurements. For this reason, performing write measurements are data-destructive when performed on file or device, but not when performed on a directory. [43]

3.3.4.1 Examples

The following commands show the read I/O latency of the storage.

```
$ ioping /mnt/directory
$ ioping /dev/device
```

These commands show the write I/O latency.

```
$ ioping -W /mnt/directory
$ ioping -WWW /dev/device
```

Chapter 4

Realization

This chapter documents the process of a configuration and subsequent benchmarking of two storage technologies:

- Ceph software-defined storage built on HPE DL 380 gen 9 servers (Ceph),
- HPE 3PAR StoreServ 7400c SAN disk array (3PAR).

First, the hardware specification will be introduced, both of the 3PAR and of the servers that will be used for Ceph. After that, the process of building a Ceph cluster will be briefly explained. The measurement methodology for comparing both Ceph and 3PAR will be explained, as well as the decision for using the particular benchmarking approaches and tools.

4.1 Hardware Used for the Tests

This section describes the hardware and software configuration of the devices that were used for the performance comparison.

4.1.1 Machine Specifications

Table 4.1 summarizes the 3PAR device specification. The information was obtained through the *HP Management Console* software connected to the device and from the QuickSpec document [4].

Table 4.2 summarizes the specifications of the servers. All of the servers are the same models with the same configuration. Two of these servers were used for Ceph storage. Other two servers were used as clients and load-generators for both Ceph and 3PAR storage. The standard Ubuntu kernel on the clients was upgraded to their newer HWE kernel in order to be able to access erasure-coded RBD images in Ceph.

Table 4.1: Specification of the HPE 3PAR storage device. Information was obtained with the *HP Management Console* from the 3PAR.

Model		HPE 3PAR 7400c
Number of controller nodes		2
Each node	CPU	2x six-core 1800 MHz
	Control memory	16 GiB
	Data memory	8 GiB
	SAS speed	6 Gbps
	FC speed	8 Gbps
	Operating system	HP 3PAR OS Version 3.3.1
	Battery backup	Yes
	Internal drive	SanDisk SATA 120 GB

Table 4.2: Configuration of the servers. Two of the servers were used for the Ceph storage, another two as clients for Ceph and 3PAR storage. Information was obtained with *lscpu* and *dmidecode* tools run on the servers.

Model	HPE DL 380 gen 9
CPU	1x octa-core 2.40 GHz, 2 threads per core
CPU model	Intel Xeon E5-2630 v3
RAM	16 GiB
Battery backup	No
Operating system	Ubuntu 16.04 LTS
Kernel (cluster)	Linux 4.4
Kernel (clients)	Linux 4.15 (HWE)

4.1.2 Number of Machines Used

One HPE 3PAR 7400c storage device was chosen to represent the classical storage array. For the comparison to be fair, Ceph storage cluster components were chosen to resemble the 3PAR configuration as close as possible. The 3PAR device consists of two controller nodes; each of the nodes has 16 GiB of RAM and two hexa-core processors (12 cores total). To match its 3PAR counterpart, the Ceph storage cluster consisted of two nodes (servers). Each server also had 16 GiB of RAM and one octa-core CPU with two threads per core (8 cores, 16 threads total).

Another two servers were used as clients. These clients were connected both to the 3PAR (using Fibre Channel connection) and to the Ceph cluster (over LAN Ethernet network).

4.1.3 Hard Drives Used

Because the license on our 3PAR system did not allow for more hard drives to be connected, the 3PAR system used a total of 12 hard drives. Ceph storage

also used 12 drives, 6 connected to each of the two server nodes. In addition to the 6 hard drives dedicated to Ceph storage, each Ceph node server uses 1 hard drive for the operating system. 3PAR uses an SSD for its operating system.

All hard drives used in 3PAR are of the same class and specification. Similarly, all hard drives used for Ceph storage are of the same class and specification. However, the drives used in 3PAR are not the same model as the drives used in Ceph. Although most of their parameters are the same, their capacity differs. The capacity of each drive used in 3PAR is 900 GB, the capacity of drives used for Ceph is only 300 GB.

Table 4.3 shows the parameters of hard drives used in 3PAR. The parameters of hard drives used for Ceph are detailed in table 4.4. The information was obtained through software tools from running systems.

Table 4.3: Specification of hard drives used for 3PAR storage. Information was obtained with the *HP Management Console* from the 3PAR.

Manufacturer	HITACHI
Model	HCBRE0900GBAS10K
Form factor	2.5”
Capacity	900 GB (reported 819 GiB)
RPM	10,000
Interface	SAS

Table 4.4: Specification of hard drives used for Ceph storage reported by various Linux tools.

Manufacturer	HP
Models	EG0300FBVFL, EG0300FCVBF, EG0300FCSPH
Form factor	2.5”
Capacity	300 GB (reported 279 GiB)
RPM	10,000
Reported sector size	512 B (physical and logical)
Interface	SAS (SPL-3)

4.1.4 Connection and Network Topology

All of the four servers are connected to the same TCP/IP network. Each server is connected through two 10 Gbit/s network interfaces, with the use of a bonding driver in Linux. The other two servers were used as clients connected both to the Ceph storage cluster and to the 3PAR device. 3PAR connects over a Fibre Channel SAN. Hostnames and corresponding IP ad-

dresses of the servers are detailed in table 4.5. The topology is depicted in figure 4.1.

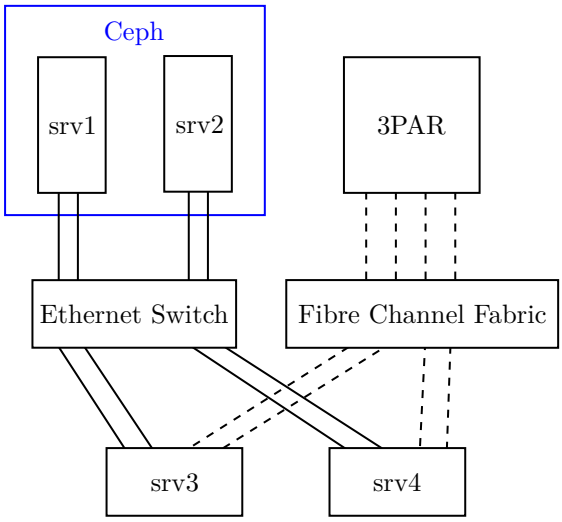


Figure 4.1: Topology of our servers and 3PAR device

Table 4.5: IP hostnames and addresses of the servers.

Hostname	IP address	Purpose
srv1	10.1.110.11	Ceph storage
srv2	10.1.110.12	Ceph storage
srv3	10.1.110.13	Client
srv4	10.1.110.14	Client

4.2 Process of Building the 3PAR Storage

3PAR was pre-installed with the HP 3PAR Operating System Version 3.3.1. There was no need to install any additional software or to perform any additional configuration steps besides creating the PGs and Virtual Volumes.

3PAR has an Ethernet network configuration interface. It was possible to *ssh* into the device and use the command-line interface, or use some other configuration software. I was using the *HP 3PAR Management Console, Version 4.7.3.2* software most of the time. Command-line interface came handy only when modifying some system options, such as a default restriction preventing RAID 0 or RAID 5 CPG creation.

4.2.1 Block Device Creation on 3PAR

These are the steps in the *3PAR Management Console* I needed to do in order to create a virtual volume and make it accessible to the client server. I used the default settings whenever possible.

1. create a CPG (set its name, RAID type, set size, and possibly the availability domain and step size within a chunklet),
2. create a virtual volume (set its name, size, CPG to use for data, and whether it should be thinly or fully provisioned),
3. export the virtual volume to the client server over Fibre Channel.

The creation was straightforward because the program uses a graphical user interface. Once the volume is exported, it can be detected by the client server. The client server may not detect the volume automatically. Either a reboot of the server or a SCSI bus scan may be performed to detect the new volume.

Each of our servers used four connections to the SAN. A *Device mapper multipathing* software driver was used to automatically create a single block device from these four connections. By default, it was configured to use a round robin algorithm to choose between the four SAN paths.

4.3 Process of Building the Ceph Storage

Two servers were used for Ceph cluster and another two servers as clients. I followed the Ceph documentation website [25] to set up and use the Ceph cluster.

4.3.1 Building the Cluster

Following the relevant Ceph documentation pages [44], I used the *ceph-deploy* tool to create the Ceph cluster and set up the clients. The first step I took before installing Ceph on the real hardware was to try to deploy a test Ceph cluster on virtual machines. After getting more familiar with the setup and configuration process, I proceeded with installation on real hardware.

Ubuntu 16.04 LTS operating system was pre-installed on all of the servers. Following the documentation instructions, I created a *cephuser* user on each of the Ceph and client servers. The *ceph-deploy* utility was run under this user. I chose server *srv1* to be the main administration server of the cluster.

I configured the system to use the official Ceph repository mentioned in the manual [24] for upstream Ceph package installation and installed the *ceph-deploy* tool from that repository. Then I used the tool to install the latest available stable version of Ceph, which was version 13.2 *Mimic* [45] at that time, both on Ceph servers (*srv1*, *srv2*) and on the client servers (*srv3*, *srv4*). Server *srv1* functioned as a Ceph Monitor and provided 6 OSDs, one per each

hard drive. Server *srv2* had 6 OSDs, one per hard drive, and no monitor. Because there are only two servers used for our Ceph cluster, adding Ceph Monitor daemon to the second server would not gain any real advantage – if there was one monitor on each of the two servers and one server failed, the remaining one monitor would not form a majority of monitors and would not be able to function anyway.

I left most Ceph options with their default settings, with a notable exception of the `osd_memory_target` OSD option. This option tells the Ceph OSD Daemon how much RAM it should use. It was preset to 4 GiB by default. This exceeds the total available memory of our servers. The servers use 6 OSDs and have a RAM size of 16 GiB. I set the `osd_memory_target` setting of each OSD to 2 GiB instead. This way, six OSDs will consume approximately 12 GiB of RAM total, and there should be enough RAM space left for the operating system and other daemons.

4.3.2 Block Device Creation on Ceph

The process of creating a Ceph block device, including the possible configuration options, can be found in the Ceph documentation [25]. The configuration options are spread across different documentation pages.

Ceph pools and images can be created from the admin node (in my case *srv1*) using command-line `ceph` and `rbd` commands. I kept the default settings whenever possible. One exception is the failure domain, which needed to be set to OSD instead of the host in most cases. The process is to:

1. create a pool (specify its name, number of placement groups, whether it is replicated or erasure-coded, and possibly the CRUSH rule or erasure-coding profile),

2. create an RBD image (specify its size, name, pool, possibly additional features).

Once an RBD image is created, it can be mapped using the `rbd` command from within a client server and used as a block device, for example `/dev/rbd0`.

4.4 Simulation of 3PAR RAID Levels on Ceph

3PAR supports RAID 0, RAID 1, RAID 5, and RAID 6. Ceph supports two types of pools: replicated and erasure-coded. This is how I imitated each RAID level on Ceph:

- RAID 0 only spreads data across multiple disks without any redundancy. This behavior can be achieved by a replicated Ceph pool with a replica size of 1 (one copy of the data).

- RAID 1 (RAID 10) keeps multiple copies of the same data block. Similar to Ceph replication with a size of 2 or greater.
- RAID 5 (RAID 50) and RAID 6 (RAID 60) can be mimicked by an erasure-coded $k+m$ pool in Ceph, with $m = 1$ for RAID 5 or $m = 2$ for RAID 6 simulation.

For example, the behavior of a RAID 5 with 5 data disks and 1 parity can be achieved with an erasure-coded pool where $k = 5$ and $m = 1$.

Ceph does not allow an RBD image to be created directly on an erasure-coded pool. An erasure-coded pool can be used as a data storage for an RBD image created on a replicated pool.

4.5 Block Size and I/O Depth Decision

Benchmarking tools, in general, allow these read/write characteristics to be specified:

- block size,
- depth of I/O queue,
- whether to use direct or buffered I/O.

To avoid any memory buffering effects in the client servers, I decided to run the benchmarks on Ceph and 3PAR using direct I/O mode. Now a few questions arise – how big should the block size be? How many simultaneous I/O requests should be queued to achieve the best real-world performance? Once the I/O requests get passed to the underlying device, will they be of the same block size and I/O depth as I specified in the parameters of the benchmark?

To answer these questions, I created a RAID 5 (5 + 1) pool both on 3PAR and on Ceph and used *fio* to access the storage from the clients with different block size and I/O depth parameters specified. I used the output of the `iostat -x` command to watch the block size and I/O queue depth of storage operations that were performed on the storage in reality. I focused on what happens to small-block operations (4 KiB) and large-block operations (1024 MiB) both in buffered (`--direct=0`) and direct (`--direct=1`) mode.

4.5.1 Small 4 KiB Block Size Behavior

I discovered that in order to keep the block size of the *fio* requests unchanged all the way to the storage, direct I/O needs to be used. Writing 4 KiB blocks to the storage using *buffered* I/O instead resulted in 4096 KiB blocks actually being queued and sent to the storage on our systems. The reported queue size was around 130 in 3PAR and 65 in Ceph. *Buffered* read with a block size of 4 KiB resulted in 256 KiB read requests to the device, with a queue size

of 1.5 both in Ceph and 3PAR. It is apparent that the block size sent to the storage is not the same as the block size issued by the *fio* benchmark when buffered I/O is used. The operating system seems to have merged multiple 4 KiB requests into one larger. Using direct I/O instead, the request size reported by iostat matched the 4 KiB block size specified in *fio*.

Direct I/O is necessary to preserve the block size of the storage access requests. To also specify the I/O depth of direct access to be other than 1, libaio engine can be used. I was able to verify that when writing (or reading) 4 KiB blocks with I/O depth of 10 with *fio* using direct I/O and libaio engine, iostat was indeed reporting 4 kibiB accesses to the storage and queue depth of 10.

4.5.2 Large 1 GiB Block Size Behavior

Next, I tried to discover the block size and I/O queue depth that yields the best performance. It was already mentioned that direct I/O is needed if one needs to make sure that a request of exactly the requested block size and I/O depth is passed to the underlying storage. Is there a maximum limit of block size or I/O depth that the devices can support? To find out, a simple *fio* benchmark was used to send *large,* 1024 MiB*, requests* to the Ceph and 3PAR storage devices, both in buffered and direct I/O. During the test, *iostat* was used to monitor the actual block size sent to the storage.

I discovered that there seems to be a 4 MiB block size limit for Ceph access on our systems, both read and write. With this block size, the maximum queue depth of about 120 for write and 170 for read was recorded. For 3PAR, there seems to be a 16 MiB block size limit when accessing the storage. The maximum queue depth with this block size was 60 for write and 40 for read.

4.5.3 Device-Reported Parameters

The findings can be compared with the Ceph and 3PAR block device parameters reported in `/sys/block/DEVICE/queue/` directory on the connected clients [46], [47].

Ceph reports the smallest request size (`logical_block_size`) of 512 B, the maximum supported block size (`max_hw_sectors_kb`) is 4 MiB. The smallest preferred I/O size (`minimum_io_size`) and the optimal I/O size (`optimal_io_size`) are reported to also be 4 MiB. The system is configured to allow the maximum request size (`max_sectors_kb`) of 4096 KiB. This seems consistent with my previous findings. Moreover, the system is configured to allow allocation of 128 requests (`nr_requests`) for read and the same amount also for write. The automatically assigned scheduler is `none`.

3PAR device supports request block sizes between 512 B and 32,767 KiB. The minimum preferred I/O size is reported to be 16 KiB, the optimal I/O size is reported to be 16 MiB. The system is configured to allow the maximum

request size of 16 MiB. The `nr_requests` value is 128. The default scheduler was `cfq`.

4.5.4 Decision for Tests

Based on the buffered read findings and the parameters reported by the storage devices, I decided to use these values for maximum sequential throughput measurement of the storage: block size of 4 MiB and I/O depth of 128. Because 3PAR was configured to allow block sizes of up to 16 MiB, this block size was also measured.

For random access tests, I decided to use multiple block sizes with different I/O depths. These include the 4 MiB and 16 MiB block sizes used for sequential access, and also 4 KiB block size (which seems to be used very often – based on my findings in the research part of this thesis and on the findings in [48]) and 128 KiB (the default block size used by the cp commands on gnu systems [49], I verified this information on my system too). I think the queue depth of 1 (only one outstanding request at a time) is important to test. The maximum tested queue depth would be around 128, based on previous findings.

4.6 Main Measurement Methodology for Performance Comparison

3PAR supports multiple RAID levels, including RAID 0, RAID 1, RAID 5, and RAID 6. Ceph uses replication by default but can be configured to use erasure coding. For each RAID option that 3PAR supports, a similarly configured Ceph erasure-coded pool was created. Two characteristics were measured: throughput and latency.

4.6.1 Pool (CPG) and Image (Virtual Volume) Parameters

For each RAID level supported by 3PAR, a CPG with that RAID setting was created. Only one CPG was created at a time. In this pool, one virtual volume with a size of 600 GiB was created. This size was chosen to be large enough not to fit in any possible cache of the machines (both client and the storage), and at the same time small enough to fit the total available hard drive capacity at the maximum replication level (RAID 1 with a replication of 4).

On Ceph cluster, a pool was created with settings matching the PG on 3PAR as close as possible. Again, one pool was created at a time, unless necessary otherwise. The only exceptions were the erasure-coded pools matching RAID 5 and RAID 6, which required an additional replicated pool for metadata storage. In such a case, one replicated pool with a size of 2 (2 replicas)

was created. An RBD image with a size of 600 GiB was created in the replicated pool and instructed to use the erasure-coded pool for data storage.

Thin-provisioned volumes and images were used on both 3PAR and Ceph. This means that the storage space was allocated on-demand as data was written to the storage. It was the default option on both systems. On our 3PAR device, the *Management Console* indicated that thin provisioning was an additional feature that was part of the system's license. I verified that the allocated space can easily be reclaimed after file deletion by issuing a TRIM command on the mounted file system:

```
$ sudo fstrim -v /MOUNTPOINT
```

I was also able to erase all contents of the volume by issuing the *blkdiscard* command:

```
$ sudo blkdiscard -v /dev/DEVICE
```

By issuing these commands, both virtual volume capacity usage reported by 3PAR and the RBD image allocated capacity on Ceph was reduced as expected.

4.6.2 Read Tests Must Read Allocated Data

One of the things I came across was the speed of reading non-existent data. This may occur when attempting to read from data location that was not written before, and thus does not hold any stored information. Reading non-existent data blocks, especially from Ceph cluster, results in unrealistically high throughput compared to the reading of the real stored data. My assumption is that Ceph and 3PAR do not need to look up the data blocks on physical disks if they do not exist. Figure 4.2 illustrates this scenario. It contains throughput plots of sequential read from empty thinly-provisioned Ceph and 3PAR volumes. The average throughput for Ceph was 9332 MiB/s, throughput of 3PAR was 1501 MiB/s. Both values are significantly higher than what can be achieved when reading existing data. When benchmarking the read performance of Ceph or 3PAR storage, it is important that the data was written there in the first place to get reliable results.

Moreover, the throughput measured on Ceph also indicates that the data did not travel through the network at all – 9332 MiB/s far exceeds the maximum network bandwidth, which is 10 Gbit/s, or 1192.1 MiB/s. The Ceph client calculates the location of data in the cluster, which probably does not give him any result. I assume that the client does not need to look up the data and only returns some random or zero value.

Throughput of 1501 MiB/s recorded on 3PAR suggests that this value may be the maximum Fibre Channel throughput on our systems.

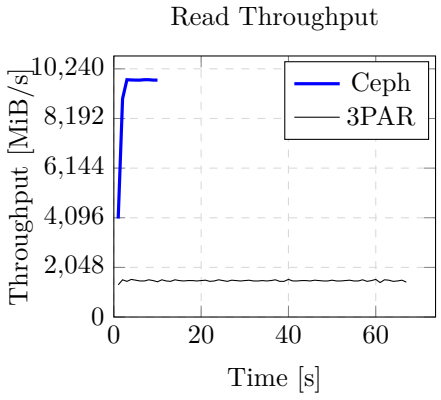


Figure 4.2: Unrealistically high throughput when reading from empty, unallocated 3PAR and Ceph storage volume. Volume size of 100 GiB, RAID 5 (5 + 1), *fio* version 3.13.

4.6.3 Sequence of Performed Tests

To avoid any file system limitations, the block device was accessed directly (the `drive` variable in the code snippets in this section). In case of Ceph RBD images, it usually was `/dev/rbd0` file, in case of 3PAR virtual volumes, it was something like `/dev/mapper/WWN`, where WWN is the World Wide Name string of the virtual volume.

These are the steps that were performed during the performance measurement, in this order:

1. issue the discard (TRIM) command to the device;
2. sequentially write to the whole block device, generate throughput plot;
3. test read and write random access for different block sizes and I/O queue depths;
4. measure read and write latency with *ioping*;
5. sequentially read the whole block device and generate throughput plot.

4.6.4 Erasing the Block Device

Before each test, the contents of the block device were discarded. The purpose of this step was to start with a clean volume that will be filled during the tests. This step was not necessary for newly-created volumes and images, which already did not hold any data, but it was performed anyways.

```
# blkdiscard -v "$drive"
```

4.6.5 Sequential Access (4 MiB Blocks)

Fio was used to measure sequential write throughput to the block device, overwriting (filling) all of its space. As mentioned previously, it was important to write data to the storage before any future read operations.

```
# fio --name=plot --filename="$drive" --ioengine=libaio --direct=1
    --rw=write --bs=4M --iodepth=128 --write_bw_log="plot-write-bw.
    log" --write_lat_log="plot-write-lat.log" --write_iops_log="plot
    -write-iops.log" --log_avg_msec=10000  --output="plot-write-
    output.txt"
```

At the end of the tests, the whole block device was sequentially read.

```
# fio --name=plot --filename="$drive" --ioengine=libaio --direct=1
    --rw=read --bs=4M --iodepth=128 --write_bw_log="plot-read-bw.log
    " --write_lat_log="plot-read-lat.log" --write_iops_log="plot-
    read-iops.log" --log_avg_msec=10000  --output="plot-read-output.
    txt"
```

The *fio* log files were saved and used to generate a plot of the sequential throughput in time.

4.6.6 Random Access (Multiple Block Sizes)

Because I did not find a definite answer to the question which block size and I/O depth is the best to use (described earlier in this chapter), I decided to test multiple block sizes and I/O depths. A random write and read test was performed for each block size and I/O depth combination for a total of 60 seconds before moving to another combination.

To make the results more accurate, the caching system of the device needs to be bypassed. One of the possible approaches is testing a data range that is too large to fit in the cache. Imagine a system that has 16 GiB of RAM which might be used as a data cache. Targeting all accesses in a small 1 GiB file may result in the contents of the file being cached in the memory, and all subsequent accesses to this file would access this cached copy. However, if the size of the file was larger than the size of RAM, the whole file would not fit in the memory cache on that particular system. The system still can put some chunks of the file in the cache, but not the whole file. Moreover, accessing the data in random order makes any predictions almost impossible.

Fio was used to randomly access the whole 900 GiB capacity of the block device to avoid any buffer caching on the target device. The buffer cache of the client was not used because *fio* was instructed to use direct I/O. *Fio* was run in random read or random write mode (`operation` variable value was one of `randwrite` or `randread`) accessing the block device directly, buffer cache was not used, each time with different block size (`blocksize`) and I/O depth (`iodepth`) settings. Block sizes of 4 KiB, 128 KiB, 1 MiB, 4 MiB, and 16 MiB;

I/O depth of 1, 16, 32, 64, 128, and 256 were tested. There was a maximum time limit of 60 seconds for each run.

```
for blocksize in {4k,128k,1M,4M,16M}; do
  for iodepth in {1,16,32,64,128,256}; do
    for operation in {randwrite,randread}; do
      fio -name=test --filename="${drive}" --output="${operation}-${
          blocksize}-${iodepth}.txt" --ioengine=libaio --direct=1 --
          rw="$operation" --runtime="$time_limit" --bs="$blocksize"
          --iodepth="$iodepth" --numjobs=1
    done
  done
done
```

4.6.7 Latency (4 KiB Blocks)

Ioping was used to measure the storage latency. By default, according to the program output, it reads 4 KiB blocks from the specified file or location and measures the time spent for each block. The following command performs 20 reads from a block device.

```
# ioping -c 20 "$drive" > ioping-read.txt
```

It is also possible to measure the write latency. The manual page discourages that unless it is performed on a directory, because it may overwrite the specified file. In our case, it did not matter because the block device was filled only with random data.

```
# ioping -c 20 -WWW "$drive" > ioping-write.txt
```

4.7 Additional Tests

Some additional tests were performed to measure additional parameters of the storage or other scenarios than the one-client-one-storage.

4.7.1 Sequential Access (16 MiB Blocks)

For all RAID levels, an additional sequential write/read test was run. This time, it measured sequential access throughput with a higher block size of 16 MiB instead of the previously used 4 MiB. The results will show whether the performance observed on 3PAR will be higher (as mentioned earlier, 3PAR reports the optimal block size of 16 MiB). This time the volume size was smaller, 100 GiB instead of 600 GiB.

```
for operation in {write,read}; do
  fio --name=plot --filename="$drive" --ioengine=libaio --direct=1
      --rw="$operation" --bs=16M --iodepth=128 --write_bw_log="
      bandwidth-${operation}.log" --write_lat_log="latency-${
```

```
      operation}.log" --write_iops_log="iops-${operation}.log" --
      log_avg_msec=1000  --output="stdout-${operation}.txt"
done
```

4.7.2 Random Access Within a Small File

The purpose of this test was to see the effect of data caching in the target Ceph and 3PAR storage devices.

Both 3PAR virtual volume and Ceph image were formatted with the XFS file system (`sudo mkfs.xfs /dev/rbd0` for Ceph) and mounted. These are the steps performed during each benchmark run:

1. before each run, the previously created test file was deleted and free space in the volume was reclaimed. `fio` is the name of the *fio* test file upon which the reads and writes are performed. `mountpoint` is the directory where the 3PAR virtual volume or Ceph image were mounted.

   ```
   # rm -f "${mountpoint}/${filename}"
   # sync; fstrim -v "${mountpoint}"; sync
   ```

2. *Fio* was run in random read or random write mode (`operation` variable value was one of `randwrite` or `randread`) accessing a file named `fio` of size 10 GiB, each time with different block size (`blocksize`) and I/O depth (`iodepth`) settings. Block sizes of 4 KiB, 128 KiB, 1 MiB, 4 MiB, and 16 MiB; I/O depth of 1, 16, 32, 64, 128, and 256 were tested. There was a maximum time limit of 60 seconds for each run.

   ```
   # fio -name=test --filename="${mountpoint}/${filename}" --
       output="${operation}-${blocksize}-${iodepth}.txt" --
       ioengine=libaio --direct=1 --rw="$operation" --size="$size"
        --runtime="$time_limit" --bs="$blocksize" --iodepth="
       $iodepth" --numjobs=1
   ```

One thing to note is that the *fio* test file was deleted before each *fio* run. Because of that, *fio* created and allocated a new test file of the target size before each run. Before randwrite run, the file was pre-allocated but not filled with data. This is acceptable because the data will be written during the benchmark run. Before randread run, however, *fio* created the test file and completely filled it with some data. This is important. The read test needs to read data that was previously written and stored in the storage.

Chapter 5

Results

This chapter describes the results of the benchmarking methodology described in the previous chapter. Detailed results (tables and plots) can be found in Appendix chapter B.

5.1 Performance of Individual Components

Ceph storage was configured on two network-connected servers. Each of these servers was using six hard drives. To have a better idea of what performance to expect from the completed Ceph storage, the performance of individual server components was tested first. Particularly, the attention was paid to these crucial components:

- hard disk drives,
- network bandwidth.

In the case of 3PAR, there does not seem to be an easy way of testing its individual components. The hard drives could be put from the 3PAR into a server and benchmarked there, but the owner of the system did not wish to do so. I was not able to find reliable performance parameters of the drives used in our 3PAR system from other sources.

5.1.1 Ceph Network Throughput and Latency

Each of the 4 servers (two servers for Ceph and two clients) is connected by two 10 Gbps Ethernet network interfaces (*ens2f0* and *ens2f1*) bonded together into a *bond0* interface. The default *layer2* transmit hash policy was used for this initial test. The bandwidth was measured with the `iperf` tool in both directions. Response time was measured with the *ping* tool, and the median value of three requests was recorded. Measured data are recorded in table 5.1.

Table 5.1: Measured network characteristics between servers. (*iperf* version 3.0.11, *ping* utility from *iputils-s20121221*)

From	To	Bandwidth	Response time
srv1	srv2	9.31 Gbits/s	0.175 ms
srv2	srv1	9.31 Gbits/s	0.131 ms
srv1	srv3	9.36 Gbits/s	0.136 ms
srv3	srv1	9.40 Gbits/s	0.129 ms
srv1	srv4	9.31 Gbits/s	0.166 ms
srv4	srv1	9.40 Gbits/s	0.143 ms
srv2	srv3	9.31 Gbits/s	0.145 ms
srv3	srv2	9.40 Gbits/s	0.140 ms
srv2	srv4	9.32 Gbits/s	0.162 ms
srv4	srv2	9.40 Gbits/s	0.140 ms
srv3	srv4	9.41 Gbits/s	0.143 ms
srv4	srv3	9.40 Gbits/s	0.145 ms
srv4	srv4	53.3 Gbits/s	0.031 ms

The measured bandwidth of the network connectivity between servers ranged from 9.31 Gbps to 9.40 Gbps. A test run against the same server, thus bypassing its physical network interface because the network packets did not need to leave the server, resulted in a bandwidth of 53.3 Gbps.

5.1.2 Ceph Network – The Effect of Ethernet Bonding

Each of our servers has two physical network interfaces and uses *bonding*. The Linux Ethernet Bonding Driver HOWTO document [50] contains all of the information needed to understand and configure Ethernet bonding. This sub-section describes the gist of it.

"*The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface.*" Bonded interface serves mainly two purposes, depending on the specific configuration:

- hot standby,
- load balancing.

One packet can only go through one network interface. This means that a communication between two devices will utilize only one of the network interfaces at a time. However, if one device communicates with multiple other devices, the bonding driver needs to decide through which interface the packets will transmitted. The driver supports these bonding modes:

- round-robin `balance-rr`, where the network interfaces are selected one by one sequentially for each packet, provides load balancing and fault tolerance;
- `active-backup` uses only one network interface until it fails, then it switches to another interface, provides fault tolerance;
- `balance-xor` the network interface is selected by `xmit_hash_policy` option (explained below), provides load balancing and fault tolerance;
- `broadcast` transmits everything on all network interfaces, provides fault tolerance;
- IEEE 802.3ad Dynamic link aggregation `802.3ad`, needs to be supported by other network devices in order to work, also uses `xmit_hash_policy`;
- adaptive transmit load balancing `balance-tlb`;
- adaptive load balancing `balance-alb`. [50]

Our servers and network switches are configured to use the *IEEE 802.3ad Dynamic link aggregation* mode. This mode uses an `xmit_hash_policy` option to choose the transmission interface. The network interface (slave) used for the transmission is determined by a calculated hash value: slave number = hash mod slave count. These hash value calculations are supported:

- `layer2`, where hash = source MAC⊕destination MAC⊕packet type ID;
- `layer2+3` uses source MAC, destination MAC, packet type ID, source IP, destination IP;
- `layer3+4` uses source port, destination port, source IP, destination IP – although "*this algorithm is not fully 802.3ad compliant*". [50]

The following command was used to observe the behavior of each mode setting on our systems. It simultaneously sends packets from the *srv4* server to *srv1*, *srv2*, and *srv3* servers and measures the throughput.

```
sebek@srv4:~/net$ iperf3 -c 10.1.110.11 > srv1.txt & iperf3 -c
    10.1.110.12 > srv2.txt & iperf3 -c 10.1.110.13 > srv3.txt &
```

Table B.1 contains the results. The *layer2* setting seems to have used only one network interface, as the aggregated throughput was just below 10 Gbit/s in all measured cases. Other `xmit_hash_policy` settings mostly resulted in the aggregated throughput of almost 20 Gbit/s, even though the number sometimes fell to 10 Gbit/s.

5.1.3 Ceph Hard Drive Performance

There will be a total of 12 hard drives used as Ceph OSDs, 6 drives in each of the two servers. Each hard drive is a 2.5" 10,000 rpm model connected through a SAS interface. To get a better idea of what throughput and latency to expect from these drives, a throughput benchmark was run on each of these drives. Drives were benchmarked one at a time.

5.1.3.1 Sequential Throughput

First, a simple test was run using the *dd* tool, which is usually pre-installed on most Linux operating systems. In this test, 1 GiB of zeros were written directly to one of the hard drives. To achieve the most reliable and realistic performance numbers, memory caching was avoided by using the `direct` output flag. Variable block sizes were used to see how much the block size affect sequential transfer speed.

I have created a script for this purpose. The script runs the following command for various block size on a hard disk device, for example `/dev/sda`.

```
dd if=$SOURCE of=$DESTINATION bs=$BS count=$((TOTAL_SIZE/BS)) oflag=
    direct
```

For example, the following command would be executed for a write speed test with a block size of 1 KiB on `/dev/sda` device.

```
dd if=/dev/zero of=/dev/sda bs=1024 count=1048576 oflag=direct
```

Read speed would be tested by the next command.

```
dd if=/dev/sda of=/dev/null bs=4096 count=262144 iflag=direct
```

The dd test is useful as the easiest option. However, writing the pattern of all zeros to a drive may not reflect the actual performance. It would not work for storage devices which offer some kind of de-duplication or compression. To verify the dd findings, the same tests were performed again, this time using a real storage benchmarking tool, *fio*.

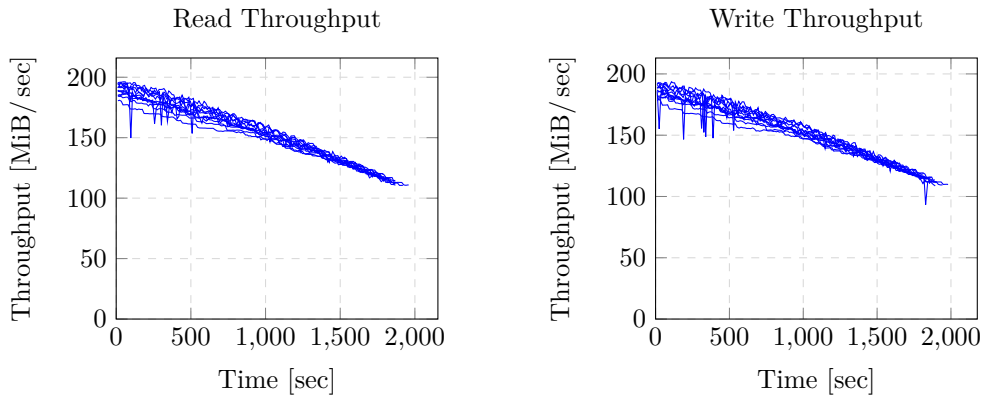
```
fio --name=test --filename=/dev/sda --ioengine=libaio --direct=1 --
    rw=write --bs=4096 --iodepth=1 --size=1073741824
fio --name=test --filename=/dev/sda --ioengine=libaio --direct=1 --
    rw=read --bs=4096 --iodepth=1 --size=1073741824
```

Table B.2 contains the results. There does not seem to be any significant difference between the *dd* and *fio* results. The maximum measured read and write throughput was around 180 MiB/s. The best write performance seems to have been achieved for block sizes of 32 MiB and higher. Read speed has not significantly increased from the block size of 16 KiB higher.

Block size of 512 MiB was found to be the best-performing during the previous sequential read and write block size test. The test accessed only

a relatively small area of the hard drive space. To see how much performance varies across the whole hard drive capacity, a sequential read and write tests were performed across the whole capacity of the drive. All drives were tested to detect whether they all perform similarly. Results are in figure 5.1.

```
for drive in sd{a,b,c,e,f,g}; do
  fio --name="read-$drive" --filename=/dev/"$drive" --ioengine=
      libaio --direct=1 --rw=read --bs=512M --iodepth=1 --
      write_bw_log="bandwidth-${drive}-read.log" --write_lat_log="
      latency-${drive}-read.log" --write_iops_log="iops-${drive}-
      read.log" --log_avg_msec=10000 > "stdout-${drive}-read.txt"
  fio --name="write-$drive" --filename=/dev/"$drive" --ioengine=
      libaio --direct=1 --rw=write --bs=512M --iodepth=1 --
      write_bw_log="bandwidth-${drive}-write.log" --write_lat_log="
      latency-${drive}-write.log" --write_iops_log="iops-${drive}-
      write.log" --log_avg_msec=10000 > "stdout-${drive}-write.txt"
done
```



(a) Sequential read

(b) Sequential write

Figure 5.1: Sequential access throughput for each individual hard drive that will be used for Ceph. Drives were tested one by one, one plot contains all drives. Block size of 512 MiB, *Fio* version 2.2.10 was used.

5.1.3.2 Random Throughput and Latency

Previous test measured the performance of sequential disk access with I/O depth of 1. *Fio* can also measure random access performance. A quick benchmark yielded results in table 5.2.

```
fio --name=read4k --filename=/dev/sda --ioengine=libaio --direct=1
    --rw=randread --bs=4k --iodepth=1 --size=4G
fio --name=write4k --filename=/dev/sda --ioengine=libaio --direct=1
    --rw=randwrite --bs=4k --iodepth=1 --size=4G
fio --name=read4kQ32 --filename=/dev/sda --ioengine=libaio --direct
    =1 --rw=randread --bs=4k --iodepth=32 --size=4G
```

```
fio --name=write4kQ32 --filename=/dev/sda --ioengine=libaio --direct
    =1 --rw=randwrite --bs=4k --iodepth=32 --size=4G
fio --name=read4kQ64 --filename=/dev/sda --ioengine=libaio --direct
    =1 --rw=randread --bs=4k --iodepth=64 --size=4G
fio --name=write4kQ64 --filename=/dev/sda --ioengine=libaio --direct
    =1 --rw=randwrite --bs=4k --iodepth=64 --size=4G
```

Table 5.2: Random 4 KiB disk performance (Ubuntu 16.04.6 LTS, kernel version 4.4.0-143-generic, *fio* version 2.2.10)

I/O depth	Read throughput	Read lat.	Write throughput	Write lat.
1	991.39 KiB/s	4.03 ms	921.750 KiB/s	4.44 ms
32	3494.3 KiB/s	36.63 ms	2557.5 KiB/s	50.04 ms
64	3610.4 KiB/s	70.90 ms	2646.4 KiB/s	96.73 ms

One thing to notice is that random read and write throughput with I/O queue length of 1 is higher than sequential read and write measured previously in table B.2. A quick run of *fio* confirmed that 4 KiB sequential throughput is indeed slower than a 4 KiB random throughput with the same parameters.

5.1.3.3 Reported Ceph OSD Performance

Ceph has a functionality to quickly evaluate the performance of the OSDs. The following command measured a write 1 GiB of data to each OSD using block size of 4 MiB.

```
$ ceph tell osd.\* bench
```

It found the performace of each OSD in our Ceph cluster to be between 88.9 MiB/s and 158.9 MiB/s. Another run resulted in measured throughput of around 82 MiB to around 125 MiB. The measured values varied between runs.

5.1.4 Summary

The following list summarizes my findings regarding the performance of individual components.

Ceph hard drives

- Maximum sequential access throughput was achieved for block sizes of at least 16 KiB for read, 32 MiB for write access.
- Maximum sequential access throughput ranges between 100 MiB/s and 200 MiB/s. It is faster at the beginning of the capacity, slower at the end.

- 4 KiB random access throughput was measured to be around 990 KiB for read, 920 KiB for write access. With higher I/O depth (32, 64), the throughput was around 3500 KiB/s for read, and 2600 KiB/s for write.
- 4 KiB random access latency was measured to be around 4 ms for read access, and around 4.4 ms for write access.

Ceph network

- The measured bandwidth of one interface was between 9.3 GB/s and 9.4 GB/s.
- The measured latency was mostly around 0.14 ms.
- Bonding in some case doubled the throughput during a communication with more than one other device, with the exception of *layer2*, which did not seem to have any advantage in our setup.

Maximum data throughput Ceph
It can be concluded, that the maximum possible data throughput between Ceph storage and a client will be around 9.31 Gbps (the measured network bandwidth), which is 1109.8 MiB/s.

Maximum data throughput 3PAR
In previous chapter, I measured the throughput of non-existent data reads from 3PAR. The result of 1501 MiB/s suggests that this value may be the maximum data throughput between 3PAR and a client.

5.2 Quick Ceph vs. 3PAR Benchmark

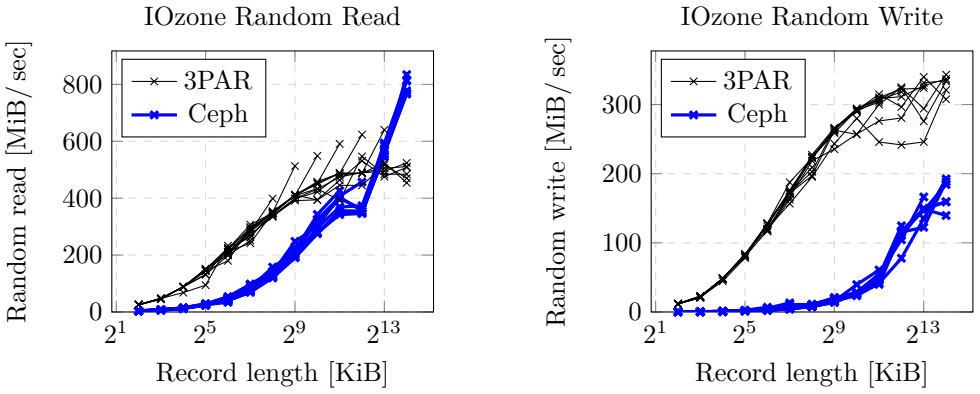
To get an idea of the Ceph and 3PAR behavior, an automatic IOzone test was performed. In this test, 3PAR virtual volume was set to RAID 5 with a set size of 5 data, 1 parity, availabilty set to magazine. Ceph was configured to use erasure-coded pool with $k = 5$, $m = 1$.

```
$ iozone -a -I
```

Results in table 5.2 reveal that Ceph read performance may be slightly slower than performance of 3PAR. Small-block write operations were significantly slower on Ceph than they were on 3PAR. However, the difference got smaller with increasing block size.

I assume that the slow write performance on Ceph compared to 3PAR may be caused by the absence of a backup battery in the Ceph cluster. Because of this, unlike 3PAR [51], Ceph cannot use write-back cache when writing to the physical disks and has to wait for each request to be physically written to the drives. i was not able to verify this assumption, because some of our drives could not be switched to a write back cache mode to simulate a battery-backed

enclosure scenario. Some sources agree with this assumption and recommend using a battery-backed RAID controller for Ceph [52], [53, p. 27].



(a) Random read

(b) Random write

Figure 5.2: Automatic IOzone test. It tested random reads and writes of different block sizes into files of different size. Each plot shows all of the file sizes at once. The results for each file size mostly overlap, with the exception of read access, where peaks are visible at the end of each read where record size equaled the file size. IOzone version 3.429.

5.3 Main Ceph vs. 3PAR Performance Results

This section presents summary of the results of the Ceph vs. 3PAR comparison tests. These tests were run on all RAID configurations supported by our 3PAR. The complete measured results can be found in sections B.3, B.5, and B.6.

The following tables contain the average throughput during sequential access and a random access latency. Table 5.3 summarizes the results for RAID 0 and RAID 1, table 5.4 summarizes RAID 5, and table 5.5 RAID 6.

RAID 6 (14 + 2) configuration could not be tested because of insufficient number of hard drives in both systems. In this RAID configuration, data need to be spread across 16 different hard drives. Both our Ceph cluster and 3PAR device have only 12 hard drives available. As a result, 3PAR refuses to create a pool with this configuration. Ceph creates a pool but reports problems with placement group availability and the pool is unusable.

Table 5.3: Sequential throughput and random 4 KiB latency results for RAID 0 and RAID 1 configuration.

RAID Level	Read Access		Write Access	
	Ceph	3PAR	Ceph	3PAR
Sequential throughput (4 MiB blocks, queue of 128)				
0 (1)	1103 MiB/s	805 MiB/s	939 MiB/s	731 MiB/s
1 (2)	1068 MiB/s	699 MiB/s	500 MiB/s	347 MiB/s
1 (3)	1103 MiB/s	703 MiB/s	324 MiB/s	259 MiB/s
1 (4)	761 MiB/s	470 MiB/s	227 MiB/s	124 MiB/s
Sequential throughput (16 MiB block size, queue of 128)				
0 (1)	1184 MiB/s	1040 MiB/s	886 MiB/s	698 MiB/s
1 (2)	1050 MiB/s	701 MiB/s	408 MiB/s	330 MiB/s
1 (3)	1069 MiB/s	724 MiB/s	301 MiB/s	251 MiB/s
1 (4)	752 MiB/s	545 MiB/s	219 MiB/s	106 MiB/s
Average latency (4 KiB blocks) – *ioping*				
0 (1)	8.25 ms	7.04 ms	7.60 ms	469.9 µs
1 (2)	7.97 ms	6.31 ms	9.46 ms	459.7 µs
1 (3)	7.01 ms	6.56 ms	11.9 ms	482.1 µs
1 (4)	7.46 ms	6.22 ms	12.8 ms	472.2 µs
Average latency (4 KiB blocks) – *fio*				
0 (1)	7.08 ms	5.55 ms	7.78 ms	328.72 µs
1 (2)	7.99 ms	8.24 ms	11.74 ms	493.39 µs
1 (3)	7.68 ms	9.97 ms	15.29 ms	710.46 µs
1 (4)	7.98 ms	13.35 ms	18.63 ms	892.99 µs

Table 5.4: Sequential throughput and random 4 KiB latency results for RAID 5 configuration.

RAID Level	Read Access		Write Access	
	Ceph	3PAR	Ceph	3PAR
Sequential throughput (4 MiB blocks, queue of 128)				
5 (2 + 1)	1106 MiB/s	816 MiB/s	642 MiB/s	362 MiB/s
5 (3 + 1)	800 MiB/s	492 MiB/s	426 MiB/s	255 MiB/s
5 (4 + 1)	819 MiB/s	625 MiB/s	618 MiB/s	415 MiB/s
5 (5 + 1)	690 MiB/s	757 MiB/s	545 MiB/s	541 MiB/s
5 (6 + 1)	669 MiB/s	493 MiB/s	540 MiB/s	384 MiB/s
5 (7 + 1)	669 MiB/s	481 MiB/s	464 MiB/s	459 MiB/s
5 (8 + 1)	659 MiB/s	583 MiB/s	614 MiB/s	494 MiB/s
Sequential throughput (16 MiB block size, queue of 128)				
5 (2 + 1)	1106 MiB/s	1064 MiB/s	579 MiB/s	412 MiB/s
5 (3 + 1)	1023 MiB/s	841 MiB/s	387 MiB/s	310 MiB/s
5 (4 + 1)	1008 MiB/s	996 MiB/s	576 MiB/s	514 MiB/s
5 (5 + 1)	951 MiB/s	952 MiB/s	517 MiB/s	597 MiB/s
5 (6 + 1)	994 MiB/s	579 MiB/s	544 MiB/s	464 MiB/s
5 (7 + 1)	897 MiB/s	635 MiB/s	432 MiB/s	503 MiB/s
5 (8 + 1)	826 MiB/s	641 MiB/s	593 MiB/s	542 MiB/s
Average latency (4 KiB blocks) – *ioping*				
5 (2 + 1)	12.6 ms	6.10 ms	27.8 ms	465.1 µs
5 (3 + 1)	16.4 ms	7.38 ms	31.0 ms	463.1 µs
5 (4 + 1)	14.8 ms	6.03 ms	30.4 ms	462.2 µs
5 (5 + 1)	16.7 ms	5.81 ms	38.7 ms	496.5 µs
5 (6 + 1)	30.0 ms	9.56 ms	33.5 ms	470.1 µs
5 (7 + 1)	19.9 ms	6.34 ms	45.9 ms	469.5 µs
5 (8 + 1)	40.4 ms	6.06 ms	64.1 ms	464.6 µs
Average latency (4 KiB blocks) – *fio*				
5 (2 + 1)	7.6 ms	7.52 ms	24.47 ms	696.53 µs
5 (3 + 1)	9.34 ms	4.28 ms	27.19 ms	879.48 µs
5 (4 + 1)	9.40 ms	6.71 ms	32.55 ms	756.10 µs
5 (5 + 1)	9.56 ms	11.00 ms	32.54 ms	766.48 µs
5 (6 + 1)	10.17 ms	7.14 ms	36.44 ms	704.40 µs
5 (7 + 1)	10.63 ms	9.71 ms	37.04 ms	698.45 µs
5 (8 + 1)	10.55 ms	10.58 ms	38.59 ms	710.69 µs

Table 5.5: Sequential throughput and random 4 KiB latency results for RAID 6 configuration.

RAID Level	Read Access		Write Access	
	Ceph	3PAR	Ceph	3PAR
Sequential throughput (4 MiB blocks, queue of 128)				
6 (4 + 2)	880 MiB/s	771 MiB/s	570 MiB/s	585 MiB/s
6 (6 + 2)	716 MiB/s	482 MiB/s	425 MiB/s	483 MiB/s
6 (8 + 2)	643 MiB/s	705 MiB/s	575 MiB/s	579 MiB/s
6 (10 + 2)	566 MiB/s	875 MiB/s	380 MiB/s	567 MiB/s
Sequential throughput (16 MiB block size, queue of 128)				
6 (4 + 2)	995 MiB/s	889 MiB/s	436 MiB/s	583 MiB/s
6 (6 + 2)	933 MiB/s	528 MiB/s	439 MiB/s	509 MiB/s
6 (8 + 2)	865 MiB/s	962 MiB/s	612 MiB/s	600 MiB/s
6 (10 + 2)	806 MiB/s	1161 MiB/s	393 MiB/s	629 MiB/s
Average latency (4 KiB blocks) – *ioping*				
6 (4 + 2)	17.1 ms	6.44 ms	33.3 ms	455.6 µs
6 (6 + 2)	24.3 ms	5.04 ms	35.2 ms	469.5 µs
6 (8 + 2)	22.6 ms	6.26 ms	46.5 ms	467.2 µs
6 (10 + 2)	31.4 ms	5.42 ms	48.1 ms	448.8 µs
Average latency (4 KiB blocks) – *fio*				
6 (4 + 2)	9.77 ms	16.44 ms	32.83 ms	985.60 µs
6 (6 + 2)	10.31 ms	16.11 ms	37.73 ms	966.84 µs
6 (8 + 2)	11.00 ms	13.64 ms	42.85 ms	930.31 µs
6 (10 + 2)	14.40 ms	15.73 ms	49.78 ms	955.00 µs

5.4 Additional Tests

This section mentions the results of additional tests performed on the Ceph and 3PAR storage systems. These tests do not test all possible RAID configurations, but pick some configuration and focus on other specific scenarios instead.

5.4.1 Random Access Within a Small File

It can be seen from the results (in section B.4) that a file of 10 GiB in size fits in the memory cache of the 3PAR device. It is apparent especially in 3PAR benchmark runs – the read performance is around 1500 MiB/s in most cases. Previously I concluded that 1500 MiB/s is the maximum throughput of the interface. One of the things that may have contributed to this phenomenon is the fact, that the read performance of a newly created file was measured (*fio* created a file and filled it with data before the read benchmark). The file was probably written to the 3PAR's cache, and subsequent reads were accessing only this cache, not its hard drives.

5.4.2 Multiple Hosts

The results of a simultaneous sequential storage access from two clients are in section B.7. The resulting plot shows that the throughput was divided between two clients. Once one of the clients finished the data transfer from/to the Ceph storage, the throughput on the other client almost doubled. Clients transfer from/to 3PAR took about the same time to finish their task.

5.4.3 Ethernet Bonding – `xmit_hash_policy`

Setting the `xmit_hash_policy` to *layer2+3* from *layer2* did not seem to have any significant effect on the sequential throughput of two simultaneous client accesses, as can be seen in the results in section B.8.

5.5 Result Discussion

I will leave the interpretation of the results mostly on the reader, who will pick the scenarios that fit them the best. In general, the presence of a battery-backed cache on 3PAR [51] seems to have resulted in much higher write performance for small blocks compared to Ceph, which was probably waiting for data to be physically written to drives. Also the write latency of 3PAR was under 1 ms in all RAID configurations.

5.5.1 Random Access

All observations in this subsection take only the queue depth of 1 into account. The first observation compares the difference in performance between read and write access observed on Ceph. On average, the read performance of Ceph was better than its write performance – 2.3× better in 4 KiB access, and 1.8× better in 16 MiB access.

This observation compares the read/write difference on 3PAR. Here, an opposite phenomenon can be seen – the 4 KiB write performance was 13× faster than the read performance. 16 MiB write performance was only 1.1× faster than the read performance.

Finally, the performance between Ceph and 3PAR is observed. On average, 4 KiB read performance of Ceph was comparable to that of 3PAR. However, 3PAR performed 31× better in 4 KiB writes. For 16 MiB read performance, Ceph was 1.4× faster than 3PAR, which was, in turn, 1.4× faster than Ceph in write performance.

5.5.2 Sequential Access

Ceph storage did not see a significant difference between the average sequential throughput with a block size of 4 MiB and 16 MiB. This might be beacuse the storage accepts writes with a block size of 4 MiB, as it was explained earlier. It can be seen in the plot – ceph throughput is more stable in 4 MiB plot compared to 16 MiB, where the requests were likely split. Ceph seems to be limited by network bandwidth during sequential read in RAID 0, RAID 1 and RAID 5 (2 + 1) configurations, with the exception of RAID 1 (4). This can be seen from the 4 MiB plot. The sequential performance of 3PAR was slightly higher during the 16 MiB access compared to 4 MiB.

5.6 Hardware Utilization During the Tests

The hardware utilization of both 3PAR and Ceph cluster was monitored during the tests. The purpose of this monitoring was to detect any possible performance issues rising from insufficient system resources.

5.6.1 Storage systems

At the end of a RAID 6 (4 + 2) test, these load values were taken on Ceph cluster nodes: 15-minute system load of 3.62 on *srv1* and 3.86 on *srv2*. The reported RAM usage on both servers was 11 GiB out of the reported 15 GiB, with additional 3.2 GiB used as buffer/cache. Plots generated from *colectd* data showed a stable RAM usage without any peaks or significant amounts of swap space being used.

The *HP Management Console* program reported the Control Memory usage of the whole 3PAR to be 18.8 GiB out of 31.3 GiB and Data Memory usage of 12.7 GiB out of 13.8 GiB.

5.6.2 Client Servers

The observed system load was low on both client servers during the storage access. Accessing Ceph storage was slightly more demanding with 15-minute average system load over one-hour period of 0.27 vs. 0.16 for 3PAR. Reported system load CPU usage was at minimum. Out of 16 GiB of RAM, client accessing the Ceph storage used the total of 457 MiB (including the operating system), client accessing 3PAR storage reported the total RAM usage of 431 MiB. Swap space was empty in both cases all of the time.

Chapter 6

Summary

This chapter summarizes the strong and weak points of Ceph and 3PAR, including the tolerance for errors of varying magnitude and availability/durability guarantees of these two systems. The security best practices for a public data center deployment follow.

6.1 Strong and Weak Points of 3PAR and Ceph

This section discusses the strong and weak points that I discovered about each of our systems. The tolerance for errors of both 3PAR and Ceph systems is described first, followed by the availability and durability guarantees these systems can offer.

6.1.1 Disk Failure Tolerance, Failure Domains

Our 3PAR model supports different RAID modes:

- RAID 0 (no disk failure tolerance);
- RAID 1 with a configurable set size of 2 (default), 3, and 4 (tolerance of 1, 2, 3 failed disks);
- RAID 5 (tolerance of one failed disk);
- RAID 6 (tolerance of two failed disks).

Similarly, Ceph supports pools that use:

- replication with the number of replicas of 1 (no disk failure tolerance) or higher (defaults to 3 replicas which allow for 2 disk failures);
- $k + m$ erasure coding, where m is the number of disks that can fail (m is settable to 0 or higher, defaults to 1).

Ceph allows a failure domain to be specified for each pool. It can be set to *osd* (each object copy is stored on different OSD), *host* (each object copy is stored on different cluster node), *chassis*, *rack*, etc. [54] By default, the *host* failure domain was set by Ceph on our system. It could be changed in an erasure code profile when creating an erasure-coded pool, or in a CRUSH rule when creating a replicated pool. 3PAR offers similar configuration option for its CPGs. The *Management Console* allows the specification of an *availability*. On our system, it offered three options to select from: *Port (Higher)*, *Cage (Default)*, or *Magazine (Lower)*. The tooltip in the application described the *availability* option this way: "[t]*he layout must support the failure of one port pair, one cage, or one drive magazine (mag)*" [55].

In Ceph, each pool has configurable numeric `size` and `min_size` settings. According to the Ceph socumentation [56], these settings are valid only for replicated pools. The `size` option specifies the number of replicas of each stored object, and `min_size` sets the minimum number of replicas required for I/O operations. In other words, if the number of available object replicas drops below `min_size`, the pool cannot be read from or written to as a safety mechanism. From my experience, these settings are specified even on erasure-coded pools. For example, an erasure-coded pool with $m = 4$, $n = 2$ has a `size` of 6 and `min_size` settable between 4 and 6 (defaults to 5). Erasure-coded pool with $m = 5$ and $n = 1$ has `size` of 6 and `min_size` settable between 5 and 6 (default 6). I verified by stopping OSD daemons to simulate OSD failure, that an $m = 5$, $n = 1$ pool becomes inaccessible (block device access stops responding) if one OSD fails, even though there are still enough objects available to reconstruct the data. After lowering the `min_size` value it becomes operable in degraded state.

6.1.2 Tolerance for Errors of Varying Magnitude

Components on different levels of hardware hierarchy can fail. Below are some of the possible scenarios and their expected impact on availability or performance of Ceph and 3PAR storage.

Drive failure As explained in previous sub-section, both Ceph and 3PAR can function in case of a hard drive failure depending on the replication level r or the n value in erasure-coded or RAID 5 and 6 pools. I tested that erasure-coded pool on Ceph with $m = 5$ and $n = 1$ survives a failure of one hard drive out of 12, but, according to my expecatation, the pool becomes inoperable if two or more drives fail.

3PAR cage failure This may happen in situations when all ports connecting the cage to the 3PAR fail. Depending on the total number of disk cages, RAID configuration, and the availability setting on CPG (how data is spread among the cages), the result of such failure may cause complete data inaccessibility. In our case (two cages), the only CPG

configuration that can survive a complete cage failure is RAID 1 with a set size of 2 and *availability* set to *Cage*. This way the data is mirrored between two cages.

Ceph network interface failure There are two interfaces in total, bonded into one virtual interface. I fone interface fails, the other interface will take care of the traffic. It may slow data traffic down a little bit in some scenarios when the data would otherwise travel in both interfaces simultaneously.

3PAR Fibre Channel interface failure There is a total of 4 Fibre Channel connections to the client server, a failure of at most 3 connections to one server may result in reduced throughput but would still work.

3PAR cage port failure There are two SAS interfaces connecting 3PAR nodes and each drive cage. Again, one of those ports can fail without affecting data availability.

Ceph node failure In our case, a failure of one node causes half of the hard drives to become unavailable. This case is similar to a 3PAR drive cage failure mentioned above. The biggest danger of running a two-node Ceph cluster is the number of Ceph Monitor daemons in the cluster. Running monitor daemons are crucial for cluster operation and data access. There is one monitor in our cluster, it is located on one of the node servers. If that server becomes unavailable, the whole cluster becomes inoperable. However, if the other server fails instead, the cluster remains operable. The addition of a second monitor to the second server not only does not help, but it would make things even worse. In such case, a failure of any one of the two node would result in the remaining monitor not forming a majority and being unable to operate the cluster. For high availability, I recommend at least three nodes in a Ceph cluster.

3PAR node failure Because both of our nodes are connected to both disk cages (and each cage is connected to both nodes), in case of a failure of one node, the remaining node should have access to all disk drives.

6.1.3 Availability Our Systems Can Offer

"*Availability is the probability that an object is accesible at any point of time. Durability is the probability that the object is not permanently lost after a failure.*" [57]

HPE guarantees 6-nine (99.9999 %) availability for their 4-node 3PAR products that are configured following their best practices. [58] This means that the allowed unavailability can be expected to be less than 32 seconds in a year. The availability of a 2-node 3PAR is not specified in that document, but my expectation is that it may fall somewhere between the general 99.999 %

availability for a properly configured system, which is mentioned in the 3PAR datasheet [51], and the previously mentioned 99.9999 % for a 4-node system.

The availability guarantee of a Ceph cluster, in general, is harder to estimate because it can be built on hardware of varying quality. Also, from my experience, Ceph allows the user to easily misconfigure the cluster configuration.

A Ceph wiki page [57] contains possible reliability estimation formulas based on combinatorial analysis. $P(x_i) = p$ is a probability that a block is accessible after x hours of operation, $q = (1-p)$ is a probability of a failure. Reliability $R(t)$ of *serial* connections (where all blocks need to be accessible for proper functionality) can be calculated using formula 6.1.

$$R(t) = P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2)\cdots P(x_n) = pn \tag{6.1}$$

Reliability of *parallel* configuration (where there are multiple paths to access the block) can be calculated using the formula 6.2. This configuration may represent the replication mode.

$$R(t) = P(x_1 + x_2 + \cdots + x_n) = 1 - (1-p)(1-p)\cdots(1-p) = 1 - qn \tag{6.2}$$

Formula 6.3 can be used to analyze *erasure-coded* pools using a (k, m) Reed-Solomon code, where k blocks are needed to reconstruct the object and m failures are tolerated. In the formula, n is the sum of k and m. It is said to work for components that are independent.

$$R(t) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{6.3}$$

Their rough estimate of reliability for various redundancy configurations is in table 6.1.

Table 6.1: Ceph: Reference Table – Reliability values for various configurations. [57]

	1	2	3	4	5	6
p, *q*	3-replication	4-replication	5-replication	(3,6) RS	(6,3) RS	(10,4) RS
0.95, 0.05	3 nines	5 nines	6 nines	7 nines	3 nines	3 nines
0.98, 0.02	5 nines	6 nines	8 nines	10 nines	4 nines	5 nines
0.99, 0.01	6 nines	8 nines	10 nines	12 nines	5 nines	6 nines

6.1.4 Overall Easiness of Use

One of the advantages of 3PAR is the possibility of its configuration through a GUI interface, such as the *HP Management Console*. Ceph is configured through command-line commands, such as *ceph* and *rbd*. There is a web GUI interface called *Ceph Dashboard* that can be enabled on Ceph, but in the version of Ceph I was using, it did not seem to offer any advanced functionality

besides displaying the cluster information. The new version of Ceph adds more features to the *Ceph Dashboard* [59], so there is hope that one day the Dashboard may gain some more advanced features.

Our 3PAR came as a complete, pre-configured device that was ready to use. The only action needed was to create virtual volumes, assign FibreChannel ports to connected client servers, and export the virtual volumes to the desired client servers. Ceph, on the other hand, required much more work to set up. This might have been caused by the fact, that I was using the upstream Ceph and installing it on servers with a clean installation of the operating system. If we purchased a complete pre-configured Ceph solution, the experience might have been different. The Ceph documentation pages provide enough information for advanced users but may be harder to understand for beginners.

My biggest issue was with the *ceph-deploy* tool. I recommend installing the latest version of the tool from the upstream repository instead of using some older version packaged in a Linux distribution's repositories. For example, trying to build a Ceph cluster on CentOS operating system (during my tests on virtual machines) using their packaged ceph-deploy tool resulted in an error, when the tool seemed to have been trying to parse some error code from an HTML page. I did not try to dig deeper into the problem and used the current upstream Ceph and ceph-deploy, which succeeded without any problem. And even the latest ceph-deploy version was not without its issues. On Ubuntu, it was returning errors because of a missing python2 package that needed to be installed to fix the problem. And other small annoyances of this type.

6.1.5 Addition of New Pools (CPGs)

It is easier to create additional CPGs on 3PAR than it is to create additional pools on Ceph. 3PAR automatically allocates the space needed for the new CPG. On the other hand, each Ceph pool has a number of placement groups (PGs) specified for each pool. This number must be manually specified and can be increased for existing pools, but not decreased. The addition of a new pool increases the number of PGs per OSD, which may exceed the optimal PG count of 100 per OSD. The recently released new version of Ceph, Nautilus, addresses this issue and enables both automatic PG count as well as a possibiity to decrease this number for existing pools [59].

6.1.6 Limitations of Ceph

One of the limitations of Ceph compared to 3PAR is that erasure-coded profile cannot be modified once the pool is created. PG number can be increased, not decreased. The other limitation I want to mention stems from the fact that Ceph is rapidly evolving. To use the latest features, the clients need to have

recent kernel modules for Ceph access. In my case, the Linux kernel on client servers had to be upgraded from the original 4.4 version shipped by Ubuntu 16.04 to a newer version to be able to access erasure-coded RBD images on the cluster. I used Ubuntu's HWE kernel version 4.15.

6.1.7 Limitations of 3PAR

In order to enable some advanced features of 3PAR, a license allowing that functionality is required. Some features are licensed per drive, some per system. For HP 3PAR StoreServ 7000 family of storage systems, additional license is required for functionality like data encryption, thinly-provisioned virtual volumes, transferring copies of volumes between systems, creating copy-on-write snapshots, software management suites. [4], [3]

6.2 Recommended Security Best Practices

This section describes my recommendations regarding the security practices when deploying Ceph and 3PAR in a public data center. I will illustrate these recommendations on the steps I performed during the build and configuration of the storage.

6.2.1 System User for Ceph Deployment

I used the *ceph-deploy* tool for Ceph cluster deployment. The Ceph documentation [24] recommends creating a new user on all cluster nodes specifically for deployment with the ceph-deploy tool. This user needs to be granted a password-less *sudo* access to the system. Also, private and public SSH keys need to be generated for a password-less SSH access for this user between the srv1 admin server and the other cluster nodes. This way, ceph-deploy is able to install additional software and configuration files on that system without being stopped by a password prompt. The documentation gives some other recommendations for the user name selection, such as using the same user name on all servers in the cluster and avoiding obvious user names that could become a target of a brute-force password attack.

I chose the username to be *cephuser*. To eliminate the possibility of a password attack on the cephuser user, and because the password-based login was no longer necessary, I locked the *cephuser* user account password (`sudo passwd -l cephuser`). By doing this, the cephuser could no longer use the password authentication and was not able to use the password to login to the system. Other local user accounts with sudo access can still switch to this user using `sudo -u cephuser -i`. The *ceph-deploy* utility was still able to remotely login using *ssh* key and execute *sudo* commands.

6.2.2 Communication Between Ceph Nodes

Ceph uses its own `cephx` authentication protocol to authenticate client servers and the Ceph cluster daemons. Ceph monitor authenticates clients and distributes secret keys. It uses shared secret keys, which means, that both monitor and a client have a copy of the key. The key has an expiration time to avoid attacks with old keys. If the ceph-deploy tool is used for cluster deployment, it sets up the authentication and installs required keys by itself. [10], [60] From my experience, even if the ceph-deploy tool installs the key on the client server, read permissions on the key file need to be applied manually for a particular user in order to use `ceph` commands without the use of `sudo`.

Ceph supports specification of users. By default, a `client.admin` user is created. Additional users with different permissions to cluster access and its configuration (pool creation, etc.) can be created. [61]

The network communication is authenticated, but not encrypted. The data can be encrypted on Ceph OSDs (not enabled by default), but not during the transfer over the network. To encrypt the transfer, the data needs to be encrypted on the client side before being stored in the Ceph storage. [62], [63] In our case, the RBD block device, such as `/dev/rbd0` would need to be encrypted in order to send the data encrypted over the network to the Ceph cluster.

Also, I strongly suggest the firewall to be enabled and configured on Ceph cluster nodes, or using some external firewall appliance, or both. Aside from a port for a remote SSH access, the firewalls on all servers in the Ceph cluster need to be configured to allow incoming communication from Ceph daemons from other servers in the cluster. As per the documentation [64], these ports are preset to:

- 6789 for Ceph Monitors;
- a range 6800–7300 for other daemons, such as OSDs.

The port numbers can be changed by the administrator.

The Ceph documentation [64] suggests using two separate networks. One public network where clients can connect, and a separate cluster network. If a denial of service attack (DoS) occurs from one or multiple clients congesting the public network, the separate cluster network is still able to allow cluster nodes to communicate and replicate data between each other. To avoid DoS attack even to the cluster network, is important that the cluster network is disconnected from the Internet and connects only the servers in the cluster.

6.2.3 3PAR Users

According to the Concepts Guide [3], HPE 3PAR requires a user account in order to manage the device. User accounts are assigned a role with appropriate

permissions (e.g. to create, view or edit CPGs, virtual volumes, …). Users can be:

- local users (Management Console, SSH),
- LDAP users,
- domain users.

6.2.4 Conclusion of Security Practices

I strongly recommend setting the firewall on the Ceph cluster nodes to reject incoming connections on all ports except the ports assigned to Ceph daemons and ports used for remote administration, such as ssh. Having two separate networks for the cluster, public network and cluster network, can avoid possible denial of service attacks on the cluster. In such situation, the client access may be limited because of congested network, but data replication between the servers in the cluster would still be possible over the secondary network.

Strong remote login passwords should be set on 3PAR and on all servers in a Ceph cluster. This includes the user name and password required to connect to 3PAR from the Management Console program, and the Linux user accounts that can be used to login to any one of the cluster servers.

User accounts for different types of storage users can be created both on Ceph and on 3PAR. The principle of least privilege should be used. For example, non-administrator account should not be able to delete or create new data pools. This is especially true for the Ceph storage. In my Ceph cluster, I was working under the default `client.admin` cephx account. This meant that I could not only connect to the cluster and use the RBD block device, but I was also able to create and delete RBD images and pools. In practice, a regular user connecting to the cluster should use a cephx account that allows him to connect to the storage cluster but not alter its configuration.

Conclusion

The goals of this thesis were to study the conceptual differences between a 3PAR storage array and Ceph software-defined storage, build a Ceph storage cluster, design a storage performance measurement methodology, and compare the performance of a Ceph cluster with the performance of 3PAR. After that, strong and weak points of both 3PAR and Ceph were to be discussed, as well as the security best practices for their deployment.

As the first step, I studied and described the principles of 3PAR and Ceph architecture. The focus was put on the physical components and the logical data layers and the data layout in various RAID configurations. Having learned the principles of the Ceph storage, I was able to build a fully functional Ceph storage cluster. I configured the Ceph cluster and a 3PAR device to be used from our client servers for performance tests. My measurement methodology was based on extensive research of possible benchmarking approaches and my observations and used free software benchmarking tools. The methodology measured random storage throughput for requests of different sizes, as well as the latency and sequential throughput of the storage.

After performing the performance measurement, I discussed the strong and weak points of 3PAR and Ceph that I consider important or which I came across during the tests. The tolerance of various errors, including failures ranging from a failure of a hard drive, a network interface, all the way to the failure of a storage node, was discussed. I mentioned the promised availability and durability guarantees of 3PAR and the method of reliability calculation for Ceph. As the last thing, I discussed my recommended security practices for a Ceph and 3PAR deployment in a data center.

The results of the Ceph vs. 3PAR performance comparison reveal that my Ceph cluster performed better during data reads with larger block sizes, while the battery-backed 3PAR excelled in small-block data writes. This thesis meets all requirements of the assignment and contains relevant information for a decision whether to a Ceph cluster can replace a 3PAR device for cloud storage.

Bibliography

1. HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP. *HPE 3PAR StoreServ Storage* [online]. © 2019 [visited on 2019-05-06]. Available from: https://www.hpe.com/cz/en/storage/3par.html.
2. VMWARE, INC. *Storage Area Network (SAN)* [online]. © 2019 [visited on 2019-05-06]. Available from: https://www.vmware.com/topics/glossary/content/storage-area-network-san.
3. HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. *HP 3PAR StoreServ Storage Concepts Guide: HP 3PAR OS 3.1.2 MU2* [Product manual]. 2013. QR482-96384. Available also from: https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c03606434&lang=en-us&cc=us.
4. HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. *HP 3PAR StoreServ 7000 Storage: QuickSpecs* [Product manual]. 2015. C04164476 — DA – 14433 Worldwide — Version 35. Available also from: https://h20195.www2.hpe.com/V2/Getdocument.aspx?docname=c04164476&ver=35.
5. COPYING file. In: *GitHub* [Software repository]. 2018 [visited on 2019-04-06]. Available from: https://github.com/ceph/ceph/blob/master/COPYING. Ceph component licenses.
6. RED HAT, INC, AND CONTRIBUTORS. OS Recommendations. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-06]. Available from: http://docs.ceph.com/docs/master/start/os-recommendations/.
7. ceph/ceph. In: *GitHub* [Software repository] [visited on 2019-04-06]. Available from: https://github.com/ceph/ceph.
8. *Ceph Homepage* [online]. Red Hat, Inc., © 2019 [visited on 2019-04-06]. Available from: https://ceph.com/.

9. WILLIAMS, Philip. *Maximize the Performance of Your Ceph Storage Solution* [online]. 2018 [visited on 2019-04-19]. Available from: https://blog.rackspace.com/maximize-performance-ceph-storage-solution.

10. RED HAT, INC, AND CONTRIBUTORS. Architecture. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-06]. Available from: http://docs.ceph.com/docs/master/architecture/.

11. RED HAT, INC, AND CONTRIBUTORS. Intro to Ceph. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-03]. Available from: http://docs.ceph.com/docs/master/start/intro/.

12. RED HAT, INC. Chapter 1. Managing the storage cluster size. *Red Hat ceph Storage 3 Operations Guide* [online]. © 2019 [visited on 2019-04-08]. Available from: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html/operations_guide/managing-the-storage-cluster-size.

13. SUSE. SUSE Enterprise Storage. *SUSE* [online]. © 2019 [visited on 2019-04-25]. Available from: https://www.suse.com/products/suse-enterprise-storage/.

14. RED HAT, INC, AND CONTRIBUTORS. rbd – manage rados block device (RBD) images. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-11]. Available from: http://docs.ceph.com/docs/mimic/man/8/rbd/.

15. RED HAT, Inc. *Storage Strategies Guide: Creating storage strategies for Red Hat Ceph Storage clusters* [online]. © 2019 [visited on 2019-04-08]. Available from: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html-single/storage_strategies_guide/index.

16. RED HAT, INC, AND CONTRIBUTORS. Placement Groups. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-11]. Available from: http://docs.ceph.com/docs/mimic/rados/operations/placement-groups/.

17. ABHISHEKL. v14.2.0 Nautilus released. *The Ceph Blog* [online]. 2019 [visited on 2019-04-06]. Available from: https://ceph.com/releases/v14-2-0-nautilus-released/.

18. RED HAT, INC, AND CONTRIBUTORS. Hardware Recommendations. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-06]. Available from: http://docs.ceph.com/docs/master/start/hardware-recommendations/.

19. RED HAT CEPH STORAGE DOCUMENTATION TEAM. Chapter 6. Recommended Minimum Hardware. *Red Hat Ceph Storage Hardware Selection Guide* [online]. 2019 [visited on 2019-04-25]. Available from: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html/red_hat_ceph_storage_hardware_selection_guide/ceph-hardware-min-recommend.

20. RED HAT, INC, AND CONTRIBUTORS. BlueStore Config Reference. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-25]. Available from: http://docs.ceph.com/docs/mimic/rados/configuration/bluestore-config-ref/.

21. ceph/ceph-deploy. In: *GitHub* [Software repository] [visited on 2019-04-06]. Available from: https://github.com/ceph/ceph-deploy.

22. INKTANK. install. *ceph-deploy 2.0.2 documentation* [online]. © 2013 [visited on 2019-04-06]. Available from: http://docs.ceph.com/ceph-deploy/docs/install.html.

23. ceph/ceph-deploy: hosts directory. In: *GitHub* [Software repository]. 2018 [visited on 2019-04-06]. Available from: https://github.com/ceph/ceph-deploy/tree/master/ceph_deploy/hosts.

24. RED HAT, INC, AND CONTRIBUTORS. Preflight Checklist. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-06]. Available from: http://docs.ceph.com/docs/master/start/quick-start-preflight/.

25. RED HAT, INC, AND CONTRIBUTORS. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-05]. Available from: http://docs.ceph.com/docs/master/.

26. RED HAT, INC, AND CONTRIBUTORS. Ceph Releases. *Ceph Documentation* [online]. © 2016 [visited on 2019-04-06]. Available from: http://docs.ceph.com/docs/master/releases/schedule/.

27. SUSE LLC. *Choosing a Ceph Solution* [online]. 2016. 261-002574-001 [visited on 2019-04-06]. Storage Solution Flyer. Available from: https://www.suse.com/media/flyer/choosing_a_ceph_solution_flyer.pdf.

28. ABHISHEKL. Ceph Mimic contributor credits. *The Ceph Blog* [online]. 2018 [visited on 2019-04-06]. Available from: https://ceph.com/community/mimic-contributor-credits/.

29. Benchmark Ceph Cluster Performance. *Ceph Wiki* [online]. 2015 [visited on 2019-04-19]. Available from: https://tracker.ceph.com/projects/ceph/wiki/Benchmark_Ceph_Cluster_Performance.

30. Benchmarking. *ArchWiki* [online]. 2019 [visited on 2019-04-19]. Available from: https://wiki.archlinux.org/index.php/benchmarking.

31. PROXMOX SERVER SOLUTIONS GMBH. *Ceph Benchmark: Hyper-converged infrastructure with Proxmox VE virtualization platform and integrated Ceph Storage.* [online]. 2018 [visited on 2019-04-20]. Available from: `https://www.proxmox.com/en/downloads/item/proxmox-ve-ceph-benchmark`.

32. RED HAT SUMMIT. Tuning High Performance Ceph for Hyper-converged infastructures, including NFVi. In: *Youtube* [online]. 2018 [visited on 2019-04-19]. Available from: `https://www.youtube.com/watch?v=wOkpfjmpiv4`.

33. STORPOOL STORAGE. Latency: #1 Metric for your cloud - StorPool at OpenStack Summit Berlin 2018. In: *Youtube* [online]. 2018 [visited on 2019-04-08]. Available from: `https://www.youtube.com/watch?v=shDenVLd9Kc`. StorPool Storage.

34. RUOSTEMAA, Janne. How to benchmark cloud servers. *Community Tutorials* [online]. © 2018 [visited on 2019-04-25]. Available from: `https://upcloud.com/community/tutorials/how-to-benchmark-cloud-servers/`.

35. MACKENZIE, David et al. *GNU Coreutils: Core GNU utilities* [online]. For version 8.31. 2019 [visited on 2019-04-19]. Available from: `https://www.gnu.org/software/coreutils/manual/coreutils.pdf`.

36. *IOzone Filesystem Benchmark* [online]. 2016 [visited on 2019-04-14]. Available from: `http://iozone.org/`.

37. *Iozone License* [online] [visited on 2019-04-14]. Available from: `http://www.iozone.org/docs/Iozone_License.txt`.

38. *Iozone Filesystem Benchmark* [online] [visited on 2019-04-14]. Available from: `http://www.iozone.org/docs/IOzone_msword_98.pdf`.

39. *IOzone Filesystem Benchmark* [online]. 2016 [visited on 2019-04-14]. Available from: `http://iozone.org/` Path: Download Source; Latest files.

40. AXBOE, Jens. *fio Documentation* [online]. Release 3.13-4-g7114-dirty. 2019 [visited on 2019-04-14]. Available from: `https://fio.readthedocs.io/en/latest/` PDF version available at `https://buildmedia.readthedocs.org/media/pdf/fio/latest/fio.pdf`.

41. CARROLL, Aaron. *fio(1) User Manual* [Linux man page]. 2004 [visited on 2019-04-07]. Available from: `https://manpages.ubuntu.com/manpages/xenial/en/man1/fio.1.html`. fio_2.2.10-1ubuntu1_amd64.deb package.

42. README.md. In: *GitHub koct9i/ioping* [Software repository]. 2018 [visited on 2019-05-07]. Available from: `https://github.com/koct9i/ioping`.

43. ioping.1. In: *GitHub koct9i/ioping* [Software repository]. 2018 [visited on 2019-05-07]. Available from: https://github.com/koct9i/ioping.

44. RED HAT, INC, AND CONTRIBUTORS. Installation (ceph-deploy). *Ceph Documentation* [online]. © 2016 [visited on 2019-05-05]. Available from: http://docs.ceph.com/docs/master/start/.

45. ABHISHEKL. 13.2.4 Mimic released. *The Ceph Blog* [online]. 2019 [visited on 2019-05-10]. Available from: https://ceph.com/releases/13-2-4-mimic-released/.

46. SUCHÁNEK, Marek; NAVRÁTIL, Milan; BAILEY, Laura, et al. 8.4. Configuration Tools. *Performance Tuning Guide* [online]. 2019 [visited on 2019-05-12]. Available from: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/performance_tuning_guide/sect-red_hat_enterprise_linux-performance_tuning_guide-storage_and_file_systems-configuration_tools.

47. AXBOE, Jens. *Queue sysfs files* [online]. 2009 [visited on 2019-05-12]. Available from: https://www.kernel.org/doc/Documentation/block/queue-sysfs.txt.

48. ADAMSON, David. Storage performance benchmarks are useful – if you read them carefully. *Hewlett Packard Enterprise Community* [online]. 2018 [visited on 2019-05-12]. Available from: https://community.hpe.com/t5/Around-the-Storage-Block/Storage-performance-benchmarks-are-useful-if-you-read-them/ba-p/7000599.

49. Efficient File Copying On Linux. *Evan Klitzke's web log* [online]. 2017 [visited on 2019-05-12]. Available from: https://eklitzke.org/efficient-file-copying-on-linux.

50. DAVIS, Thomas; TARREAU, Willy; GAVRILOV, Constantine, et al. *Linux Ethernet Bonding Driver HOWTO* [online]. 2011 [visited on 2019-04-08]. Available from: https://www.kernel.org/doc/Documentation/networking/bonding.txt.

51. HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP. *HPE 3PAR StoreServ Storage: designed for mission-critical high availability* [online]. 2015. 4AA3-8316ENW, Rev. 6 [visited on 2019-05-13]. Available from: https://www.optiodata.com/documents/spec/HPE-3PAR-StoreServ-storage-datasheet.pdf.

52. POAT, M D; LAURET, J. Achieving cost/performance balance ratio using tiered storage caching techniques: A case study with CephFS. *Journal of Physics: Conference Series*. 2017, vol. 898, pp. 062022. Available from DOI: 10.1088/1742-6596/898/6/062022.

53. FISK, N. *Mastering Ceph: Infrastructure storage solutions with the latest Ceph release, 2nd Edition*. Packt Publishing, 2019. ISBN 9781789615104. Available also from: https://books.google.cz/books?id=vuiLDwAAQBAJ.

54. RED HAT, INC, AND CONTRIBUTORS. Manually editing a CRUSH Map. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-12]. Available from: http://docs.ceph.com/docs/mimic/rados/operations/crush-map-edits/.

55. HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. *HP 3PAR Management Console. Version 4.7.3.2* [software]. © 2016.

56. RED HAT, INC, AND CONTRIBUTORS. Pools. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-10]. Available from: http://docs.ceph.com/docs/mimic/rados/operations/pools/.

57. GALINANES, Veronica Estrada. Final report. *Ceph Wiki – Reliability model* [online]. 2015 [visited on 2019-05-13]. Available from: https://tracker.ceph.com/projects/ceph/wiki/Final_report.

58. HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP. *99.9999 percent data availability* [online]. 2016. 4AA5-2846ENW, Rev. 3 [visited on 2019-05-13]. Available from: https://cdn.cnetcontent.com/1f/38/1f3804de-d8af-4def-a202-5b0e97f9b355.pdf.

59. RED HAT, INC, AND CONTRIBUTORS. v14.2.1 Nautilus. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-06]. Available from: http://docs.ceph.com/docs/master/releases/nautilus/.

60. RED HAT, INC, AND CONTRIBUTORS. Cephx Config Reference. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-10]. Available from: http://docs.ceph.com/docs/mimic/rados/configuration/auth-config-ref/.

61. RED HAT, INC, AND CONTRIBUTORS. User Management. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-10]. Available from: http://docs.ceph.com/docs/mimic/rados/operations/user-management/.

62. RED HAT, INC, AND CONTRIBUTORS. Encryption. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-10]. Available from: http://docs.ceph.com/docs/master/radosgw/encryption/.

63. RED HAT CEPH STORAGE DOCUMENTATION TEAM. Chapter 3. Encryption and Key Management. *Data Security and Hardening Guide* [online]. © 2019 [visited on 2019-05-10]. Available from: https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html/data_security_and_hardening_guide/assembly-encryption-and-key-management.

64. RED HAT, INC, AND CONTRIBUTORS. Network Configuration Reference. *Ceph Documentation* [online]. © 2016 [visited on 2019-05-11]. Available from: http://docs.ceph.com/docs/mimic/rados/configuration/network-config-ref/.

Appendix A

Acronyms

CPG Common Provisioning Group (3PAR).

CRUSH Controlled Replication Under Scalable Hashing.

DoS Denial of Service.

HPE Hewlett Packard Enterprise.

I/O input/output.

IOPS input/output operations per second.

LGPL GNU Lesser General Public License.

PG Placement Group (Ceph).

RADOS Reliable Autonomic Distributed Object Store.

RAID Redundant Array of Independent Disks.

RAID 0 (1) RAID 0 (or replication) with 1 data block (1 replica).

RAID 1 (r) RAID 1 (or replication) with r data blocks (r replicas).

RAID 5 ($m + 1$) RAID 5 (or erasure coding) with m data blocks and 1 parity block.

RAID 6 ($m + 2$) RAID 6 (or erasure coding) with m data blocks and 2 parity blocks.

RBD RADOS Block Device.

SAN storage area network.

Appendix B

Detailed Benchmark Results

This chapter presents detailed results of benchmarks performed in the practical part of this thesis. It contains mostly tables and plots that were not included in the text because of their size.

B.1 Bonded Network Performance

This section contains measured network performance data.

Table B.1: Comparison of different *IEEE 802.3ad Dynamic link aggregation* `xmit_hash_policy` settings. In this test, *srv4* uploaded data to all of the other servers (*srv1*, *srv2*, *srv3*) simultaneously. Three bandwidth measurements (not necessarily consequent) were recorded. (*iperf* version 3.0.11)

`xmit_hash_policy`	From	To	Bandwidth (test 1)	Bandwidth (test 2)	Bandwidth (test 3)
`layer2` (default)	srv4	srv1	3.14 Gbits/sec	2.36 Gbits/sec	2.36 Gbits/sec
		srv2	3.14 Gbits/sec	4.71 Gbits/sec	2.36 Gbits/sec
		srv3	3.14 Gbits/sec	2.36 Gbits/sec	4.71 Gbits/sec
`layer2+3`	srv4	srv1	9.40 Gbits/sec	9.35 Gbits/sec	9.40 Gbits/sec
		srv2	4.71 Gbits/sec	97.2 Mbits/sec	4.71 Gbits/sec
		srv3	4.71 Gbits/sec	97.6 Mbits/sec	4.71 Gbits/sec
`layer3+4`	srv4	srv1	4.71 Gbits/sec	3.14 Gbits/sec	4.71 Gbits/sec
		srv2	9.40 Gbits/sec	3.14 Gbits/sec	4.71 Gbits/sec
		srv3	4.70 Gbits/sec	3.14 Gbits/sec	9.40 Gbits/sec
`encap2+3`	srv4	srv1	9.40 Gbits/sec	9.40 Gbits/sec	9.40 Gbits/sec
		srv2	4.71 Gbits/sec	4.71 Gbits/sec	4.72 Gbits/sec
		srv3	4.71 Gbits/sec	4.71 Gbits/sec	4.71 Gbits/sec
`encap3+4`	srv4	srv1	9.40 Gbits/sec	3.14 Gbits/sec	4.71 Gbits/sec
		srv2	4.71 Gbits/sec	3.14 Gbits/sec	9.40 Gbits/sec
		srv3	4.71 Gbits/sec	3.14 Gbits/sec	4.71 Gbits/sec

B.2 Individual Hard Drive Block Size Performance

Detailed individual hard drive performance results are in this section.

Table B.2: Disk transfer speed by write block size measured on the same single hard drive. Ubuntu 16.04.6 LTS, kernel version 4.4.0-143-generic, dd (coreutils) 8.25, fio-2.2.10.

Block size	Sequential write		Sequential read	
	dd	fio	dd	fio
512 B	82.75 KiB/s	82.75 KiB/s	12.32 MiB/s	11.09 MiB/s
1 KiB	165.39 KiB/s	165.36 KiB/s	22.79 MiB/s	21.36 MiB/s
2 KiB	330.5 KiB/s	330.72 KiB/s	43.58 MiB/s	40.81 MiB/s
4 KiB	659.74 KiB/s	659.13 KiB/s	80.44 MiB/s	75.39 MiB/s
8 KiB	1315.26 KiB/s	1317.3 KiB/s	136.99 MiB/s	123.97 MiB/s
16 KiB	2.55 MiB/s	2.55 MiB/s	184.38 MiB/s	180.25 MiB/s
32 KiB	5.03 MiB/s	5.05 MiB/s	184.27 MiB/s	184.14 MiB/s
64 KiB	9.75 MiB/s	9.85 MiB/s	184.3 MiB/s	184.27 MiB/s
128 KiB	18.64 MiB/s	18.72 MiB/s	184.34 MiB/s	184.21 MiB/s
256 KiB	33.67 MiB/s	33.95 MiB/s	182.74 MiB/s	184.27 MiB/s
512 KiB	56.66 MiB/s	57.4 MiB/s	184.24 MiB/s	184.31 MiB/s
1 MiB	85.93 MiB/s	87.55 MiB/s	184.3 MiB/s	184.17 MiB/s
2 MiB	116.64 MiB/s	118.59 MiB/s	184.19 MiB/s	184.34 MiB/s
4 MiB	142.07 MiB/s	144.1 MiB/s	184.17 MiB/s	184.31 MiB/s
8 MiB	158.15 MiB/s	161.62 MiB/s	184.27 MiB/s	183.88 MiB/s
16 MiB	167.1 MiB/s	172.22 MiB/s	183.38 MiB/s	183.84 MiB/s
32 MiB	170.02 MiB/s	177.93 MiB/s	184.36 MiB/s	184.17 MiB/s
64 MiB	173.39 MiB/s	178.77 MiB/s	183.95 MiB/s	184.24 MiB/s
128 MiB	174.92 MiB/s	180.95 MiB/s	183.99 MiB/s	184.07 MiB/s
256 MiB	175.79 MiB/s	182.08 MiB/s	184.08 MiB/s	184.01 MiB/s
512 MiB	175.63 MiB/s	182.43 MiB/s	184.13 MiB/s	184.21 MiB/s

B.3 Random Access Throughput

This section presents the most important measured data from my Ceph vs. 3PAR performance comparison. The following tables show the random access throughput measured for storage accesses of different block size and I/O depth.

B.3.1 RAID 1 and RAID 0

The following tables show the results obtained on RAID 0 and RAID 1 3PAR and corresponding Ceph configurations.

Table B.3: RAID 0 (1 replica). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	564 KiB/s	5474 KiB/s	8230 KiB/s	9.96 MiB/s	10.8 MiB/s	11.6 MiB/s
	rand. write	514 KiB/s	2896 KiB/s	4155 KiB/s	5823 KiB/s	8449 KiB/s	8451 KiB/s
128 KiB	rand. read	17.8 MiB/s	181 MiB/s	254 MiB/s	313 MiB/s	351 MiB/s	361 MiB/s
	rand. write	7902 KiB/s	84.8 MiB/s	133 MiB/s	190 MiB/s	265 MiB/s	267 MiB/s
1 MiB	rand. read	61.9 MiB/s	611 MiB/s	811 MiB/s	915 MiB/s	923 MiB/s	962 MiB/s
	rand. write	39.9 MiB/s	396 MiB/s	576 MiB/s	739 MiB/s	898 MiB/s	880 MiB/s
4 MiB	rand. read	85.5 MiB/s	796 MiB/s	1010 MiB/s	1096 MiB/s	1104 MiB/s	1107 MiB/s
	rand. write	69.6 MiB/s	553 MiB/s	731 MiB/s	862 MiB/s	863 MiB/s	869 MiB/s
16 MiB	rand. read	224 MiB/s	1049 MiB/s	1107 MiB/s	1117 MiB/s	1117 MiB/s	1116 MiB/s
	rand. write	178 MiB/s	741 MiB/s	850 MiB/s	848 MiB/s	852 MiB/s	850 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	720 KiB/s	6118 KiB/s	12.8 MiB/s	22.4 MiB/s	35.4 MiB/s	48.4 MiB/s
	rand. write	11.8 MiB/s	16.4 MiB/s	16.5 MiB/s	16.5 MiB/s	16.5 MiB/s	16.5 MiB/s
128 KiB	rand. read	15.4 MiB/s	134 MiB/s	302 MiB/s	324 MiB/s	339 MiB/s	326 MiB/s
	rand. write	155 MiB/s	278 MiB/s	279 MiB/s	280 MiB/s	279 MiB/s	278 MiB/s
1 MiB	rand. read	69.2 MiB/s	449 MiB/s	492 MiB/s	565 MiB/s	584 MiB/s	583 MiB/s
	rand. write	279 MiB/s	518 MiB/s	516 MiB/s	518 MiB/s	518 MiB/s	506 MiB/s
4 MiB	rand. read	139 MiB/s	707 MiB/s	688 MiB/s	706 MiB/s	705 MiB/s	706 MiB/s
	rand. write	303 MiB/s	612 MiB/s	605 MiB/s	591 MiB/s	581 MiB/s	563 MiB/s
16 MiB	rand. read	199 MiB/s	849 MiB/s	805 MiB/s	823 MiB/s	826 MiB/s	828 MiB/s
	rand. write	302 MiB/s	640 MiB/s	631 MiB/s	619 MiB/s	596 MiB/s	598 MiB/s

Table B.4: RAID 1 (2 replicas). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	500 KiB/s	5303 KiB/s	9673 KiB/s	11.0 MiB/s	12.2 MiB/s	14.1 MiB/s
	rand. write	341 KiB/s	1586 KiB/s	2241 KiB/s	3388 KiB/s	4885 KiB/s	4779 KiB/s
128 KiB	rand. read	16.0 MiB/s	168 MiB/s	219 MiB/s	243 MiB/s	257 MiB/s	261 MiB/s
	rand. write	6292 KiB/s	58.0 MiB/s	84.1 MiB/s	118 MiB/s	161 MiB/s	171 MiB/s
1 MiB	rand. read	58.3 MiB/s	539 MiB/s	671 MiB/s	746 MiB/s	801 MiB/s	790 MiB/s
	rand. write	34.9 MiB/s	271 MiB/s	365 MiB/s	414 MiB/s	527 MiB/s	487 MiB/s
4 MiB	rand. read	87.5 MiB/s	770 MiB/s	970 MiB/s	1055 MiB/s	1076 MiB/s	1074 MiB/s
	rand. write	65.7 MiB/s	354 MiB/s	441 MiB/s	471 MiB/s	478 MiB/s	479 MiB/s
16 MiB	rand. read	228 MiB/s	1013 MiB/s	1097 MiB/s	1115 MiB/s	1115 MiB/s	1100 MiB/s
	rand. write	130 MiB/s	439 MiB/s	450 MiB/s	449 MiB/s	452 MiB/s	455 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	485 KiB/s	5366 KiB/s	10.9 MiB/s	20.3 MiB/s	31.4 MiB/s	42.3 MiB/s
	rand. write	8054 KiB/s	8429 KiB/s	8440 KiB/s	8423 KiB/s	8429 KiB/s	8433 KiB/s
128 KiB	rand. read	14.4 MiB/s	123 MiB/s	295 MiB/s	215 MiB/s	449 MiB/s	277 MiB/s
	rand. write	131 MiB/s	135 MiB/s	135 MiB/s	135 MiB/s	135 MiB/s	134 MiB/s
1 MiB	rand. read	70.6 MiB/s	437 MiB/s	469 MiB/s	532 MiB/s	549 MiB/s	553 MiB/s
	rand. write	253 MiB/s	258 MiB/s	255 MiB/s	255 MiB/s	253 MiB/s	254 MiB/s
4 MiB	rand. read	131 MiB/s	636 MiB/s	598 MiB/s	617 MiB/s	620 MiB/s	601 MiB/s
	rand. write	274 MiB/s	276 MiB/s	282 MiB/s	283 MiB/s	283 MiB/s	272 MiB/s
16 MiB	rand. read	171 MiB/s	712 MiB/s	646 MiB/s	674 MiB/s	679 MiB/s	682 MiB/s
	rand. write	247 MiB/s	286 MiB/s	290 MiB/s	302 MiB/s	282 MiB/s	287 MiB/s

Table B.5: RAID 1 (3 replicas). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	520 KiB/s	5125 KiB/s	9.79 MiB/s	13.3 MiB/s	14.4 MiB/s	15.2 MiB/s
	rand. write	262 KiB/s	1075 KiB/s	1657 KiB/s	2340 KiB/s	3386 KiB/s	3423 KiB/s
128 KiB	rand. read	15.4 MiB/s	160 MiB/s	215 MiB/s	253 MiB/s	264 MiB/s	277 MiB/s
	rand. write	5934 KiB/s	46.5 MiB/s	67.6 MiB/s	91.8 MiB/s	115 MiB/s	122 MiB/s
1 MiB	rand. read	56.2 MiB/s	512 MiB/s	669 MiB/s	743 MiB/s	750 MiB/s	723 MiB/s
	rand. write	32.9 MiB/s	200 MiB/s	261 MiB/s	303 MiB/s	318 MiB/s	317 MiB/s
4 MiB	rand. read	77.8 MiB/s	706 MiB/s	909 MiB/s	1002 MiB/s	1020 MiB/s	1016 MiB/s
	rand. write	51.5 MiB/s	241 MiB/s	288 MiB/s	301 MiB/s	303 MiB/s	299 MiB/s
16 MiB	rand. read	210 MiB/s	1008 MiB/s	1080 MiB/s	1109 MiB/s	1109 MiB/s	1109 MiB/s
	rand. write	102 MiB/s	289 MiB/s	289 MiB/s	289 MiB/s	287 MiB/s	287 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	401 KiB/s	4092 KiB/s	8988 KiB/s	15.5 MiB/s	23.9 MiB/s	32.3 MiB/s
	rand. write	5604 KiB/s	5795 KiB/s	5814 KiB/s	5771 KiB/s	5811 KiB/s	5655 KiB/s
128 KiB	rand. read	13.8 MiB/s	123 MiB/s	290 MiB/s	200 MiB/s	418 MiB/s	245 MiB/s
	rand. write	79.8 MiB/s	84.0 MiB/s	84.7 MiB/s	84.0 MiB/s	84.9 MiB/s	85.5 MiB/s
1 MiB	rand. read	66.6 MiB/s	414 MiB/s	437 MiB/s	488 MiB/s	488 MiB/s	496 MiB/s
	rand. write	155 MiB/s	159 MiB/s	155 MiB/s	154 MiB/s	154 MiB/s	153 MiB/s
4 MiB	rand. read	126 MiB/s	604 MiB/s	552 MiB/s	565 MiB/s	562 MiB/s	566 MiB/s
	rand. write	175 MiB/s	181 MiB/s	180 MiB/s	179 MiB/s	178 MiB/s	177 MiB/s
16 MiB	rand. read	171 MiB/s	682 MiB/s	586 MiB/s	603 MiB/s	626 MiB/s	622 MiB/s
	rand. write	191 MiB/s	187 MiB/s	189 MiB/s	193 MiB/s	158 MiB/s	156 MiB/s

Table B.6: RAID 1 (4 replicas). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	501 KiB/s	4721 KiB/s	9106 KiB/s	12.7 MiB/s	13.7 MiB/s	14.7 MiB/s
	rand. write	215 KiB/s	833 KiB/s	1257 KiB/s	1863 KiB/s	2485 KiB/s	2589 KiB/s
128 KiB	rand. read	13.0 MiB/s	150 MiB/s	187 MiB/s	198 MiB/s	200 MiB/s	197 MiB/s
	rand. write	5839 KiB/s	41.3 MiB/s	58.7 MiB/s	77.7 MiB/s	90.8 MiB/s	87.2 MiB/s
1 MiB	rand. read	48.4 MiB/s	410 MiB/s	470 MiB/s	472 MiB/s	465 MiB/s	470 MiB/s
	rand. write	27.7 MiB/s	126 MiB/s	154 MiB/s	175 MiB/s	175 MiB/s	176 MiB/s
4 MiB	rand. read	69.9 MiB/s	608 MiB/s	654 MiB/s	658 MiB/s	651 MiB/s	656 MiB/s
	rand. write	45.5 MiB/s	166 MiB/s	175 MiB/s	160 MiB/s	176 MiB/s	176 MiB/s
16 MiB	rand. read	195 MiB/s	689 MiB/s	704 MiB/s	702 MiB/s	701 MiB/s	701 MiB/s
	rand. write	83.4 MiB/s	177 MiB/s	174 MiB/s	172 MiB/s	174 MiB/s	172 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	299 KiB/s	3679 KiB/s	8055 KiB/s	12.9 MiB/s	19.9 MiB/s	26.8 MiB/s
	rand. write	4463 KiB/s	4651 KiB/s	4657 KiB/s	4652 KiB/s	4640 KiB/s	4426 KiB/s
128 KiB	rand. read	12.6 MiB/s	119 MiB/s	275 MiB/s	189 MiB/s	405 MiB/s	227 MiB/s
	rand. write	57.7 MiB/s	62.6 MiB/s	62.9 MiB/s	62.8 MiB/s	63.3 MiB/s	63.3 MiB/s
1 MiB	rand. read	59.5 MiB/s	422 MiB/s	445 MiB/s	495 MiB/s	499 MiB/s	507 MiB/s
	rand. write	117 MiB/s	117 MiB/s	117 MiB/s	117 MiB/s	115 MiB/s	115 MiB/s
4 MiB	rand. read	104 MiB/s	585 MiB/s	550 MiB/s	550 MiB/s	558 MiB/s	547 MiB/s
	rand. write	125 MiB/s	124 MiB/s	126 MiB/s	123 MiB/s	123 MiB/s	123 MiB/s
16 MiB	rand. read	130 MiB/s	653 MiB/s	582 MiB/s	611 MiB/s	629 MiB/s	616 MiB/s
	rand. write	114 MiB/s	128 MiB/s	128 MiB/s	131 MiB/s	110 MiB/s	118 MiB/s

B.3.2 RAID 5

The following tables show the results obtained on RAID 5 3PAR and corresponding Ceph configuration.

Table B.7: RAID 5 (2 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	521 KiB/s	4123 KiB/s	7156 KiB/s	9714 KiB/s	10.4 MiB/s	11.5 MiB/s
	rand. write	163 KiB/s	1183 KiB/s	2691 KiB/s	4134 KiB/s	6073 KiB/s	5426 KiB/s
128 KiB	rand. read	13.9 MiB/s	81.2 MiB/s	110 MiB/s	129 MiB/s	139 MiB/s	150 MiB/s
	rand. write	5356 KiB/s	42.0 MiB/s	64.4 MiB/s	87.5 MiB/s	110 MiB/s	113 MiB/s
1 MiB	rand. read	54.4 MiB/s	413 MiB/s	485 MiB/s	544 MiB/s	568 MiB/s	565 MiB/s
	rand. write	32.0 MiB/s	209 MiB/s	259 MiB/s	297 MiB/s	320 MiB/s	313 MiB/s
4 MiB	rand. read	112 MiB/s	770 MiB/s	918 MiB/s	995 MiB/s	1031 MiB/s	1049 MiB/s
	rand. write	80.4 MiB/s	428 MiB/s	508 MiB/s	526 MiB/s	543 MiB/s	622 MiB/s
16 MiB	rand. read	267 MiB/s	1041 MiB/s	1105 MiB/s	1113 MiB/s	1113 MiB/s	1113 MiB/s
	rand. write	178 MiB/s	507 MiB/s	636 MiB/s	634 MiB/s	645 MiB/s	609 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	532 KiB/s	5299 KiB/s	12.5 MiB/s	22.7 MiB/s	35.5 MiB/s	48.9 MiB/s
	rand. write	5715 KiB/s	6245 KiB/s	6254 KiB/s	6261 KiB/s	6151 KiB/s	6053 KiB/s
128 KiB	rand. read	15.6 MiB/s	140 MiB/s	327 MiB/s	254 MiB/s	290 MiB/s	288 MiB/s
	rand. write	81.3 MiB/s	85.4 MiB/s	85.6 MiB/s	85.9 MiB/s	85.4 MiB/s	85.8 MiB/s
1 MiB	rand. read	80.7 MiB/s	429 MiB/s	437 MiB/s	475 MiB/s	487 MiB/s	482 MiB/s
	rand. write	208 MiB/s	209 MiB/s	209 MiB/s	209 MiB/s	209 MiB/s	207 MiB/s
4 MiB	rand. read	138 MiB/s	645 MiB/s	599 MiB/s	617 MiB/s	619 MiB/s	608 MiB/s
	rand. write	256 MiB/s	255 MiB/s	255 MiB/s	251 MiB/s	249 MiB/s	237 MiB/s
16 MiB	rand. read	191 MiB/s	774 MiB/s	693 MiB/s	696 MiB/s	703 MiB/s	704 MiB/s
	rand. write	233 MiB/s	260 MiB/s	244 MiB/s	242 MiB/s	247 MiB/s	242 MiB/s

Table B.8: RAID 5 (3 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	428 KiB/s	2625 KiB/s	4044 KiB/s	5605 KiB/s	5923 KiB/s	6410 KiB/s
	rand. write	147 KiB/s	894 KiB/s	1943 KiB/s	2836 KiB/s	4627 KiB/s	4234 KiB/s
128 KiB	rand. read	12.1 MiB/s	53.7 MiB/s	61.7 MiB/s	69.0 MiB/s	69.0 MiB/s	70.7 MiB/s
	rand. write	3754 KiB/s	22.2 MiB/s	27.0 MiB/s	35.0 MiB/s	38.5 MiB/s	40.8 MiB/s
1 MiB	rand. read	53.8 MiB/s	306 MiB/s	355 MiB/s	372 MiB/s	380 MiB/s	360 MiB/s
	rand. write	24.1 MiB/s	129 MiB/s	162 MiB/s	173 MiB/s	181 MiB/s	174 MiB/s
4 MiB	rand. read	135 MiB/s	713 MiB/s	803 MiB/s	830 MiB/s	840 MiB/s	839 MiB/s
	rand. write	95.7 MiB/s	366 MiB/s	402 MiB/s	414 MiB/s	444 MiB/s	444 MiB/s
16 MiB	rand. read	296 MiB/s	886 MiB/s	924 MiB/s	932 MiB/s	930 MiB/s	929 MiB/s
	rand. write	143 MiB/s	396 MiB/s	418 MiB/s	421 MiB/s	418 MiB/s	420 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	934 KiB/s	4337 KiB/s	9872 KiB/s	16.4 MiB/s	25.2 MiB/s	34.1 MiB/s
	rand. write	4531 KiB/s	5374 KiB/s	5402 KiB/s	5383 KiB/s	5383 KiB/s	5110 KiB/s
128 KiB	rand. read	13.3 MiB/s	109 MiB/s	253 MiB/s	188 MiB/s	382 MiB/s	212 MiB/s
	rand. write	63.4 MiB/s	68.7 MiB/s	68.6 MiB/s	68.8 MiB/s	68.7 MiB/s	68.8 MiB/s
1 MiB	rand. read	66.2 MiB/s	399 MiB/s	408 MiB/s	430 MiB/s	441 MiB/s	436 MiB/s
	rand. write	163 MiB/s	168 MiB/s	168 MiB/s	168 MiB/s	166 MiB/s	166 MiB/s
4 MiB	rand. read	120 MiB/s	585 MiB/s	528 MiB/s	547 MiB/s	551 MiB/s	540 MiB/s
	rand. write	193 MiB/s	197 MiB/s	194 MiB/s	196 MiB/s	190 MiB/s	191 MiB/s
16 MiB	rand. read	143 MiB/s	658 MiB/s	604 MiB/s	616 MiB/s	633 MiB/s	631 MiB/s
	rand. write	178 MiB/s	205 MiB/s	199 MiB/s	194 MiB/s	169 MiB/s	157 MiB/s

Table B.9: RAID 5 (4 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	425 KiB/s	2238 KiB/s	3560 KiB/s	4576 KiB/s	4772 KiB/s	4816 KiB/s
	rand. write	123 KiB/s	772 KiB/s	1619 KiB/s	2428 KiB/s	3734 KiB/s	3621 KiB/s
128 KiB	rand. read	13.2 MiB/s	52.0 MiB/s	67.1 MiB/s	81.1 MiB/s	89.9 MiB/s	90.1 MiB/s
	rand. write	6332 KiB/s	29.9 MiB/s	44.8 MiB/s	64.7 MiB/s	85.4 MiB/s	77.4 MiB/s
1 MiB	rand. read	61.8 MiB/s	330 MiB/s	399 MiB/s	414 MiB/s	425 MiB/s	427 MiB/s
	rand. write	36.6 MiB/s	200 MiB/s	252 MiB/s	282 MiB/s	300 MiB/s	281 MiB/s
4 MiB	rand. read	137 MiB/s	756 MiB/s	852 MiB/s	890 MiB/s	920 MiB/s	921 MiB/s
	rand. write	99.8 MiB/s	479 MiB/s	564 MiB/s	607 MiB/s	639 MiB/s	621 MiB/s
16 MiB	rand. read	292 MiB/s	790 MiB/s	949 MiB/s	963 MiB/s	968 MiB/s	960 MiB/s
	rand. write	179 MiB/s	529 MiB/s	605 MiB/s	569 MiB/s	612 MiB/s	614 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	596 KiB/s	4505 KiB/s	10.2 MiB/s	17.7 MiB/s	27.1 MiB/s	36.5 MiB/s
	rand. write	5267 KiB/s	5501 KiB/s	5481 KiB/s	5443 KiB/s	5485 KiB/s	5264 KiB/s
128 KiB	rand. read	13.4 MiB/s	114 MiB/s	265 MiB/s	198 MiB/s	402 MiB/s	221 MiB/s
	rand. write	65.8 MiB/s	71.5 MiB/s	71.4 MiB/s	71.3 MiB/s	72.2 MiB/s	72.0 MiB/s
1 MiB	rand. read	68.3 MiB/s	392 MiB/s	400 MiB/s	425 MiB/s	426 MiB/s	426 MiB/s
	rand. write	178 MiB/s	182 MiB/s	181 MiB/s	179 MiB/s	176 MiB/s	177 MiB/s
4 MiB	rand. read	125 MiB/s	582 MiB/s	522 MiB/s	533 MiB/s	549 MiB/s	545 MiB/s
	rand. write	208 MiB/s	219 MiB/s	220 MiB/s	216 MiB/s	212 MiB/s	211 MiB/s
16 MiB	rand. read	159 MiB/s	676 MiB/s	607 MiB/s	619 MiB/s	632 MiB/s	629 MiB/s
	rand. write	189 MiB/s	224 MiB/s	220 MiB/s	211 MiB/s	212 MiB/s	203 MiB/s

Table B.10: RAID 5 (5 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	418 KiB/s	2005 KiB/s	3082 KiB/s	3749 KiB/s	3887 KiB/s	2982 KiB/s
	rand. write	123 KiB/s	706 KiB/s	1405 KiB/s	2164 KiB/s	3078 KiB/s	3087 KiB/s
128 KiB	rand. read	11.0 MiB/s	38.6 MiB/s	42.3 MiB/s	52.7 MiB/s	53.3 MiB/s	53.6 MiB/s
	rand. write	3734 KiB/s	17.8 MiB/s	25.9 MiB/s	31.3 MiB/s	28.6 MiB/s	27.4 MiB/s
1 MiB	rand. read	54.9 MiB/s	238 MiB/s	278 MiB/s	288 MiB/s	280 MiB/s	301 MiB/s
	rand. write	21.2 MiB/s	93.8 MiB/s	123 MiB/s	135 MiB/s	126 MiB/s	139 MiB/s
4 MiB	rand. read	152 MiB/s	705 MiB/s	789 MiB/s	817 MiB/s	816 MiB/s	818 MiB/s
	rand. write	108 MiB/s	438 MiB/s	515 MiB/s	540 MiB/s	544 MiB/s	514 MiB/s
16 MiB	rand. read	287 MiB/s	733 MiB/s	775 MiB/s	786 MiB/s	793 MiB/s	796 MiB/s
	rand. write	172 MiB/s	491 MiB/s	534 MiB/s	540 MiB/s	621 MiB/s	600 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	363 KiB/s	4140 KiB/s	9748 KiB/s	16.6 MiB/s	25.8 MiB/s	35.1 MiB/s
	rand. write	5196 KiB/s	5390 KiB/s	5384 KiB/s	5388 KiB/s	5396 KiB/s	5233 KiB/s
128 KiB	rand. read	12.2 MiB/s	105 MiB/s	249 MiB/s	188 MiB/s	387 MiB/s	218 MiB/s
	rand. write	59.6 MiB/s	64.1 MiB/s	62.9 MiB/s	64.3 MiB/s	64.4 MiB/s	64.3 MiB/s
1 MiB	rand. read	61.6 MiB/s	360 MiB/s	371 MiB/s	395 MiB/s	399 MiB/s	398 MiB/s
	rand. write	142 MiB/s	146 MiB/s	148 MiB/s	148 MiB/s	145 MiB/s	145 MiB/s
4 MiB	rand. read	124 MiB/s	566 MiB/s	499 MiB/s	505 MiB/s	510 MiB/s	504 MiB/s
	rand. write	170 MiB/s	173 MiB/s	177 MiB/s	176 MiB/s	171 MiB/s	174 MiB/s
16 MiB	rand. read	164 MiB/s	678 MiB/s	590 MiB/s	611 MiB/s	620 MiB/s	625 MiB/s
	rand. write	186 MiB/s	204 MiB/s	199 MiB/s	190 MiB/s	186 MiB/s	184 MiB/s

Table B.11: RAID 5 (6 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	393 KiB/s	1687 KiB/s	2514 KiB/s	2969 KiB/s	2807 KiB/s	2718 KiB/s
	rand. write	110 KiB/s	636 KiB/s	1290 KiB/s	2064 KiB/s	2868 KiB/s	2749 KiB/s
128 KiB	rand. read	11.2 MiB/s	34.8 MiB/s	46.6 MiB/s	49.7 MiB/s	51.7 MiB/s	53.4 MiB/s
	rand. write	3335 KiB/s	16.8 MiB/s	30.7 MiB/s	32.1 MiB/s	38.7 MiB/s	30.5 MiB/s
1 MiB	rand. read	53.3 MiB/s	197 MiB/s	244 MiB/s	261 MiB/s	266 MiB/s	261 MiB/s
	rand. write	22.7 MiB/s	92.9 MiB/s	122 MiB/s	146 MiB/s	160 MiB/s	154 MiB/s
4 MiB	rand. read	140 MiB/s	615 MiB/s	679 MiB/s	720 MiB/s	710 MiB/s	710 MiB/s
	rand. write	101 MiB/s	389 MiB/s	474 MiB/s	542 MiB/s	549 MiB/s	555 MiB/s
16 MiB	rand. read	265 MiB/s	698 MiB/s	744 MiB/s	743 MiB/s	708 MiB/s	756 MiB/s
	rand. write	166 MiB/s	499 MiB/s	564 MiB/s	573 MiB/s	555 MiB/s	575 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	560 KiB/s	4600 KiB/s	10.4 MiB/s	18.2 MiB/s	28.3 MiB/s	38.5 MiB/s
	rand. write	5652 KiB/s	5918 KiB/s	5881 KiB/s	5910 KiB/s	5874 KiB/s	5786 KiB/s
128 KiB	rand. read	13.4 MiB/s	115 MiB/s	272 MiB/s	204 MiB/s	420 MiB/s	238 MiB/s
	rand. write	75.6 MiB/s	78.5 MiB/s	79.2 MiB/s	78.6 MiB/s	79.4 MiB/s	79.2 MiB/s
1 MiB	rand. read	64.5 MiB/s	357 MiB/s	365 MiB/s	408 MiB/s	423 MiB/s	423 MiB/s
	rand. write	173 MiB/s	177 MiB/s	177 MiB/s	179 MiB/s	175 MiB/s	174 MiB/s
4 MiB	rand. read	135 MiB/s	572 MiB/s	506 MiB/s	514 MiB/s	533 MiB/s	530 MiB/s
	rand. write	212 MiB/s	221 MiB/s	219 MiB/s	218 MiB/s	210 MiB/s	204 MiB/s
16 MiB	rand. read	179 MiB/s	688 MiB/s	601 MiB/s	614 MiB/s	631 MiB/s	622 MiB/s
	rand. write	210 MiB/s	251 MiB/s	247 MiB/s	239 MiB/s	230 MiB/s	231 MiB/s

Table B.12: RAID 5 (7 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	376 KiB/s	1200 KiB/s	1821 KiB/s	2251 KiB/s	2378 KiB/s	2454 KiB/s
	rand. write	108 KiB/s	566 KiB/s	1212 KiB/s	1869 KiB/s	2284 KiB/s	2367 KiB/s
128 KiB	rand. read	11.3 MiB/s	33.7 MiB/s	42.6 MiB/s	43.9 MiB/s	49.1 MiB/s	48.9 MiB/s
	rand. write	3170 KiB/s	16.2 MiB/s	26.3 MiB/s	26.3 MiB/s	33.5 MiB/s	23.5 MiB/s
1 MiB	rand. read	54.8 MiB/s	200 MiB/s	221 MiB/s	227 MiB/s	216 MiB/s	233 MiB/s
	rand. write	18.2 MiB/s	80.3 MiB/s	99.4 MiB/s	118 MiB/s	122 MiB/s	127 MiB/s
4 MiB	rand. read	147 MiB/s	604 MiB/s	640 MiB/s	646 MiB/s	634 MiB/s	642 MiB/s
	rand. write	91.3 MiB/s	333 MiB/s	396 MiB/s	430 MiB/s	408 MiB/s	406 MiB/s
16 MiB	rand. read	259 MiB/s	618 MiB/s	678 MiB/s	689 MiB/s	694 MiB/s	694 MiB/s
	rand. write	125 MiB/s	363 MiB/s	389 MiB/s	406 MiB/s	403 MiB/s	407 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	412 KiB/s	4387 KiB/s	10.3 MiB/s	18.3 MiB/s	28.5 MiB/s	38.6 MiB/s
	rand. write	5699 KiB/s	5912 KiB/s	5917 KiB/s	5922 KiB/s	5897 KiB/s	5866 KiB/s
128 KiB	rand. read	13.6 MiB/s	114 MiB/s	267 MiB/s	202 MiB/s	412 MiB/s	241 MiB/s
	rand. write	64.1 MiB/s	68.0 MiB/s	67.1 MiB/s	66.1 MiB/s	65.7 MiB/s	66.1 MiB/s
1 MiB	rand. read	58.2 MiB/s	333 MiB/s	350 MiB/s	399 MiB/s	403 MiB/s	405 MiB/s
	rand. write	174 MiB/s	176 MiB/s	175 MiB/s	172 MiB/s	172 MiB/s	172 MiB/s
4 MiB	rand. read	127 MiB/s	531 MiB/s	465 MiB/s	473 MiB/s	477 MiB/s	482 MiB/s
	rand. write	208 MiB/s	217 MiB/s	215 MiB/s	213 MiB/s	209 MiB/s	208 MiB/s
16 MiB	rand. read	183 MiB/s	670 MiB/s	569 MiB/s	576 MiB/s	576 MiB/s	582 MiB/s
	rand. write	213 MiB/s	244 MiB/s	235 MiB/s	228 MiB/s	221 MiB/s	216 MiB/s

Table B.13: RAID 5 (8 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	379 KiB/s	1363 KiB/s	1986 KiB/s	2262 KiB/s	2336 KiB/s	2388 KiB/s
	rand. write	104 KiB/s	544 KiB/s	1156 KiB/s	1905 KiB/s	2281 KiB/s	2205 KiB/s
128 KiB	rand. read	11.5 MiB/s	34.5 MiB/s	45.4 MiB/s	57.1 MiB/s	60.0 MiB/s	69.1 MiB/s
	rand. write	4309 KiB/s	19.5 MiB/s	35.7 MiB/s	55.5 MiB/s	58.0 MiB/s	68.4 MiB/s
1 MiB	rand. read	63.8 MiB/s	260 MiB/s	301 MiB/s	334 MiB/s	348 MiB/s	344 MiB/s
	rand. write	37.3 MiB/s	183 MiB/s	234 MiB/s	271 MiB/s	296 MiB/s	295 MiB/s
4 MiB	rand. read	144 MiB/s	612 MiB/s	654 MiB/s	680 MiB/s	673 MiB/s	669 MiB/s
	rand. write	108 MiB/s	426 MiB/s	510 MiB/s	553 MiB/s	583 MiB/s	571 MiB/s
16 MiB	rand. read	268 MiB/s	634 MiB/s	675 MiB/s	686 MiB/s	689 MiB/s	692 MiB/s
	rand. write	136 MiB/s	460 MiB/s	554 MiB/s	554 MiB/s	564 MiB/s	568 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	378 KiB/s	4072 KiB/s	9346 KiB/s	16.4 MiB/s	26.1 MiB/s	35.0 MiB/s
	rand. write	5601 KiB/s	5783 KiB/s	5815 KiB/s	5804 KiB/s	5800 KiB/s	5626 KiB/s
128 KiB	rand. read	13.7 MiB/s	109 MiB/s	257 MiB/s	195 MiB/s	406 MiB/s	232 MiB/s
	rand. write	68.6 MiB/s	75.6 MiB/s	76.1 MiB/s	76.5 MiB/s	76.9 MiB/s	75.9 MiB/s
1 MiB	rand. read	62.9 MiB/s	325 MiB/s	329 MiB/s	373 MiB/s	389 MiB/s	392 MiB/s
	rand. write	188 MiB/s	192 MiB/s	194 MiB/s	193 MiB/s	192 MiB/s	190 MiB/s
4 MiB	rand. read	135 MiB/s	535 MiB/s	464 MiB/s	470 MiB/s	475 MiB/s	477 MiB/s
	rand. write	225 MiB/s	247 MiB/s	245 MiB/s	243 MiB/s	237 MiB/s	235 MiB/s
16 MiB	rand. read	196 MiB/s	668 MiB/s	567 MiB/s	578 MiB/s	582 MiB/s	582 MiB/s
	rand. write	224 MiB/s	275 MiB/s	272 MiB/s	265 MiB/s	258 MiB/s	257 MiB/s

B.3.3 RAID 6

The following tables show the results obtained on RAID 6 3PAR and corresponding Ceph configuration.

Table B.14: RAID 6 (4 data, 2 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	409 KiB/s	2331 KiB/s	3806 KiB/s	4888 KiB/s	5452 KiB/s	6016 KiB/s
	rand. write	122 KiB/s	720 KiB/s	1438 KiB/s	2271 KiB/s	3165 KiB/s	3225 KiB/s
128 KiB	rand. read	12.8 MiB/s	50.4 MiB/s	67.9 MiB/s	81.0 MiB/s	98.0 MiB/s	96.0 MiB/s
	rand. write	5188 KiB/s	23.0 MiB/s	41.3 MiB/s	58.1 MiB/s	82.9 MiB/s	73.5 MiB/s
1 MiB	rand. read	60.7 MiB/s	330 MiB/s	379 MiB/s	411 MiB/s	426 MiB/s	431 MiB/s
	rand. write	35.1 MiB/s	184 MiB/s	244 MiB/s	242 MiB/s	278 MiB/s	288 MiB/s
4 MiB	rand. read	137 MiB/s	746 MiB/s	855 MiB/s	895 MiB/s	909 MiB/s	921 MiB/s
	rand. write	95.9 MiB/s	432 MiB/s	509 MiB/s	552 MiB/s	571 MiB/s	573 MiB/s
16 MiB	rand. read	292 MiB/s	899 MiB/s	979 MiB/s	998 MiB/s	1001 MiB/s	963 MiB/s
	rand. write	168 MiB/s	464 MiB/s	539 MiB/s	557 MiB/s	558 MiB/s	508 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	243 KiB/s	3545 KiB/s	7749 KiB/s	13.4 MiB/s	21.6 MiB/s	30.1 MiB/s
	rand. write	4044 KiB/s	4145 KiB/s	4150 KiB/s	4089 KiB/s	3985 KiB/s	3772 KiB/s
128 KiB	rand. read	13.2 MiB/s	118 MiB/s	272 MiB/s	215 MiB/s	392 MiB/s	243 MiB/s
	rand. write	37.7 MiB/s	44.6 MiB/s	44.1 MiB/s	44.5 MiB/s	44.6 MiB/s	44.7 MiB/s
1 MiB	rand. read	66.9 MiB/s	423 MiB/s	456 MiB/s	515 MiB/s	532 MiB/s	532 MiB/s
	rand. write	120 MiB/s	124 MiB/s	125 MiB/s	124 MiB/s	123 MiB/s	122 MiB/s
4 MiB	rand. read	128 MiB/s	632 MiB/s	588 MiB/s	595 MiB/s	611 MiB/s	599 MiB/s
	rand. write	150 MiB/s	150 MiB/s	151 MiB/s	151 MiB/s	148 MiB/s	149 MiB/s
16 MiB	rand. read	176 MiB/s	737 MiB/s	646 MiB/s	662 MiB/s	691 MiB/s	684 MiB/s
	rand. write	184 MiB/s	184 MiB/s	176 MiB/s	174 MiB/s	132 MiB/s	160 MiB/s

Table B.15: RAID 6 (6 data, 2 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	388 KiB/s	1611 KiB/s	2488 KiB/s	3008 KiB/s	3221 KiB/s	3457 KiB/s
	rand. write	106 KiB/s	581 KiB/s	1218 KiB/s	2001 KiB/s	2439 KiB/s	2495 KiB/s
128 KiB	rand. read	11.8 MiB/s	35.8 MiB/s	45.5 MiB/s	48.8 MiB/s	55.8 MiB/s	53.0 MiB/s
	rand. write	3184 KiB/s	16.6 MiB/s	26.4 MiB/s	29.1 MiB/s	39.4 MiB/s	28.6 MiB/s
1 MiB	rand. read	55.7 MiB/s	220 MiB/s	245 MiB/s	259 MiB/s	260 MiB/s	257 MiB/s
	rand. write	21.7 MiB/s	90.5 MiB/s	112 MiB/s	140 MiB/s	144 MiB/s	146 MiB/s
4 MiB	rand. read	137 MiB/s	592 MiB/s	658 MiB/s	682 MiB/s	685 MiB/s	694 MiB/s
	rand. write	92.1 MiB/s	361 MiB/s	432 MiB/s	470 MiB/s	498 MiB/s	504 MiB/s
16 MiB	rand. read	263 MiB/s	639 MiB/s	720 MiB/s	737 MiB/s	742 MiB/s	740 MiB/s
	rand. write	151 MiB/s	429 MiB/s	438 MiB/s	481 MiB/s	483 MiB/s	484 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	248 KiB/s	3060 KiB/s	7024 KiB/s	11.0 MiB/s	19.8 MiB/s	27.0 MiB/s
	rand. write	4122 KiB/s	4188 KiB/s	4148 KiB/s	4122 KiB/s	4046 KiB/s	3953 KiB/s
128 KiB	rand. read	12.6 MiB/s	106 MiB/s	240 MiB/s	179 MiB/s	375 MiB/s	215 MiB/s
	rand. write	27.9 MiB/s	35.6 MiB/s	36.3 MiB/s	36.4 MiB/s	36.4 MiB/s	36.7 MiB/s
1 MiB	rand. read	66.8 MiB/s	366 MiB/s	378 MiB/s	425 MiB/s	445 MiB/s	448 MiB/s
	rand. write	115 MiB/s	116 MiB/s	115 MiB/s	114 MiB/s	112 MiB/s	111 MiB/s
4 MiB	rand. read	133 MiB/s	569 MiB/s	489 MiB/s	493 MiB/s	517 MiB/s	509 MiB/s
	rand. write	149 MiB/s	148 MiB/s	145 MiB/s	143 MiB/s	140 MiB/s	138 MiB/s
16 MiB	rand. read	187 MiB/s	678 MiB/s	574 MiB/s	582 MiB/s	593 MiB/s	589 MiB/s
	rand. write	179 MiB/s	188 MiB/s	176 MiB/s	164 MiB/s	136 MiB/s	139 MiB/s

Table B.16: RAID 6 (8 data, 2 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	363 KiB/s	1311 KiB/s	1911 KiB/s	2248 KiB/s	2448 KiB/s	2546 KiB/s
	rand. write	93.3 KiB/s	518 KiB/s	1082 KiB/s	1655 KiB/s	1806 KiB/s	1989 KiB/s
128 KiB	rand. read	11.2 MiB/s	33.0 MiB/s	43.5 MiB/s	55.0 MiB/s	55.5 MiB/s	65.1 MiB/s
	rand. write	4380 KiB/s	19.2 MiB/s	30.8 MiB/s	46.5 MiB/s	51.7 MiB/s	59.9 MiB/s
1 MiB	rand. read	63.2 MiB/s	254 MiB/s	309 MiB/s	338 MiB/s	331 MiB/s	321 MiB/s
	rand. write	36.9 MiB/s	180 MiB/s	240 MiB/s	276 MiB/s	271 MiB/s	268 MiB/s
4 MiB	rand. read	128 MiB/s	589 MiB/s	652 MiB/s	685 MiB/s	687 MiB/s	685 MiB/s
	rand. write	100 MiB/s	390 MiB/s	497 MiB/s	548 MiB/s	571 MiB/s	581 MiB/s
16 MiB	rand. read	259 MiB/s	642 MiB/s	687 MiB/s	697 MiB/s	693 MiB/s	692 MiB/s
	rand. write	160 MiB/s	467 MiB/s	529 MiB/s	524 MiB/s	517 MiB/s	515 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	293 KiB/s	3866 KiB/s	8452 KiB/s	14.1 MiB/s	22.4 MiB/s	31.1 MiB/s
	rand. write	4284 KiB/s	4420 KiB/s	4416 KiB/s	4406 KiB/s	4373 KiB/s	4130 KiB/s
128 KiB	rand. read	13.4 MiB/s	118 MiB/s	275 MiB/s	213 MiB/s	398 MiB/s	249 MiB/s
	rand. write	33.6 MiB/s	39.1 MiB/s	38.8 MiB/s	39.0 MiB/s	39.5 MiB/s	39.5 MiB/s
1 MiB	rand. read	69.4 MiB/s	418 MiB/s	443 MiB/s	516 MiB/s	564 MiB/s	567 MiB/s
	rand. write	102 MiB/s	103 MiB/s	102 MiB/s	101 MiB/s	100 MiB/s	98.5 MiB/s
4 MiB	rand. read	140 MiB/s	629 MiB/s	586 MiB/s	604 MiB/s	609 MiB/s	619 MiB/s
	rand. write	138 MiB/s	134 MiB/s	129 MiB/s	128 MiB/s	126 MiB/s	124 MiB/s
16 MiB	rand. read	209 MiB/s	747 MiB/s	667 MiB/s	673 MiB/s	693 MiB/s	683 MiB/s
	rand. write	179 MiB/s	176 MiB/s	164 MiB/s	151 MiB/s	118 MiB/s	112 MiB/s

Table B.17: RAID 6 (10 data, 2 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (600 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	278 KiB/s	976 KiB/s	1377 KiB/s	1680 KiB/s	1838 KiB/s	1959 KiB/s
	rand. write	80.3 KiB/s	493 KiB/s	984 KiB/s	1093 KiB/s	1649 KiB/s	1779 KiB/s
128 KiB	rand. read	10.5 MiB/s	26.4 MiB/s	38.4 MiB/s	38.9 MiB/s	43.2 MiB/s	44.5 MiB/s
	rand. write	2830 KiB/s	13.0 MiB/s	27.0 MiB/s	21.6 MiB/s	29.6 MiB/s	21.2 MiB/s
1 MiB	rand. read	51.1 MiB/s	153 MiB/s	169 MiB/s	179 MiB/s	182 MiB/s	170 MiB/s
	rand. write	16.4 MiB/s	56.7 MiB/s	70.8 MiB/s	86.8 MiB/s	93.9 MiB/s	87.9 MiB/s
4 MiB	rand. read	138 MiB/s	489 MiB/s	512 MiB/s	523 MiB/s	528 MiB/s	530 MiB/s
	rand. write	74.0 MiB/s	270 MiB/s	306 MiB/s	324 MiB/s	332 MiB/s	249 MiB/s
16 MiB	rand. read	240 MiB/s	508 MiB/s	566 MiB/s	580 MiB/s	582 MiB/s	584 MiB/s
	rand. write	95.5 MiB/s	269 MiB/s	288 MiB/s	314 MiB/s	280 MiB/s	316 MiB/s

(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	254 KiB/s	3340 KiB/s	7249 KiB/s	12.3 MiB/s	19.6 MiB/s	27.1 MiB/s
	rand. write	4173 KiB/s	4233 KiB/s	4232 KiB/s	4200 KiB/s	4236 KiB/s	4028 KiB/s
128 KiB	rand. read	13.0 MiB/s	111 MiB/s	254 MiB/s	198 MiB/s	397 MiB/s	243 MiB/s
	rand. write	24.9 MiB/s	27.9 MiB/s	29.5 MiB/s	30.1 MiB/s	28.0 MiB/s	28.4 MiB/s
1 MiB	rand. read	65.9 MiB/s	409 MiB/s	436 MiB/s	523 MiB/s	581 MiB/s	571 MiB/s
	rand. write	86.1 MiB/s	87.1 MiB/s	86.0 MiB/s	86.3 MiB/s	84.8 MiB/s	85.9 MiB/s
4 MiB	rand. read	141 MiB/s	636 MiB/s	604 MiB/s	639 MiB/s	641 MiB/s	643 MiB/s
	rand. write	118 MiB/s	112 MiB/s	110 MiB/s	110 MiB/s	106 MiB/s	106 MiB/s
16 MiB	rand. read	218 MiB/s	757 MiB/s	700 MiB/s	704 MiB/s	707 MiB/s	702 MiB/s
	rand. write	167 MiB/s	161 MiB/s	145 MiB/s	132 MiB/s	94.4 MiB/s	103 MiB/s

B.4 Random Access Within a 10 GiB File

This section contains results of an additional random access measurement performed on Ceph and 3PAR storage. Previous measurements were performed on the whole size of the block device. This time, the block device was formatted with an XFS file system and the measurements targeted one 10 GiB file. The effects of caching on the Ceph and 3PAR devices is obvious when compared with previous results.

Table B.18: RAID 5 (5 data, 1 parity). Random access performance for requests of different block size and I/O depth. Accesses were performed across the whole drive (100 GiB) with *fio* version 3.13.

(a) Ceph RBD image.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	343 KiB/s	1614 KiB/s	1828 KiB/s	2039 KiB/s	2172 KiB/s	2188 KiB/s
	rand. write	112 KiB/s	483 KiB/s	651 KiB/s	792 KiB/s	1012 KiB/s	1014 KiB/s
128 KiB	rand. read	9875 KiB/s	44.4 MiB/s	50.6 MiB/s	56.3 MiB/s	58.6 MiB/s	55.5 MiB/s
	rand. write	3381 KiB/s	13.2 MiB/s	17.5 MiB/s	21.6 MiB/s	24.4 MiB/s	25.1 MiB/s
1 MiB	rand. read	53.7 MiB/s	209 MiB/s	243 MiB/s	254 MiB/s	265 MiB/s	250 MiB/s
	rand. write	22.5 MiB/s	85.4 MiB/s	114 MiB/s	126 MiB/s	140 MiB/s	140 MiB/s
4 MiB	rand. read	115 MiB/s	359 MiB/s	398 MiB/s	404 MiB/s	409 MiB/s	406 MiB/s
	rand. write	68.6 MiB/s	250 MiB/s	298 MiB/s	333 MiB/s	348 MiB/s	349 MiB/s
16 MiB	rand. read	218 MiB/s	367 MiB/s	400 MiB/s	419 MiB/s	429 MiB/s	419 MiB/s
	rand. write	119 MiB/s	346 MiB/s	329 MiB/s	363 MiB/s	372 MiB/s	369 MiB/s

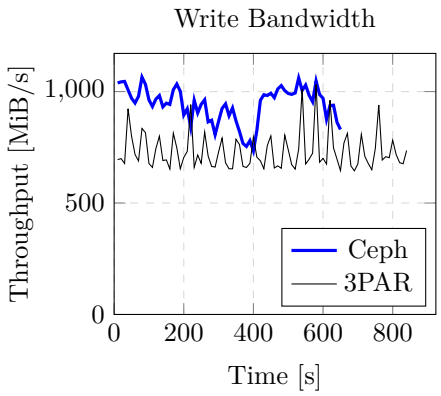
(b) 3PAR virtual volume.

Block size	Operation	I/O Depth					
		1	16	32	64	128	256
4 KiB	rand. read	23.6 MiB/s	208 MiB/s	475 MiB/s	532 MiB/s	536 MiB/s	474 MiB/s
	rand. write	9602 KiB/s	21.0 MiB/s	21.0 MiB/s	20.3 MiB/s	20.0 MiB/s	18.5 MiB/s
128 KiB	rand. read	245 MiB/s	1571 MiB/s	1571 MiB/s	1573 MiB/s	1573 MiB/s	1573 MiB/s
	rand. write	131 MiB/s	165 MiB/s	180 MiB/s	183 MiB/s	182 MiB/s	181 MiB/s
1 MiB	rand. read	406 MiB/s	1532 MiB/s	1543 MiB/s	1531 MiB/s	1531 MiB/s	1574 MiB/s
	rand. write	239 MiB/s	426 MiB/s	435 MiB/s	430 MiB/s	441 MiB/s	412 MiB/s
4 MiB	rand. read	451 MiB/s	1501 MiB/s	1504 MiB/s	1506 MiB/s	1506 MiB/s	1567 MiB/s
	rand. write	250 MiB/s	526 MiB/s	540 MiB/s	545 MiB/s	544 MiB/s	530 MiB/s
16 MiB	rand. read	497 MiB/s	1493 MiB/s	1489 MiB/s	1500 MiB/s	1500 MiB/s	1528 MiB/s
	rand. write	240 MiB/s	551 MiB/s	551 MiB/s	582 MiB/s	578 MiB/s	580 MiB/s

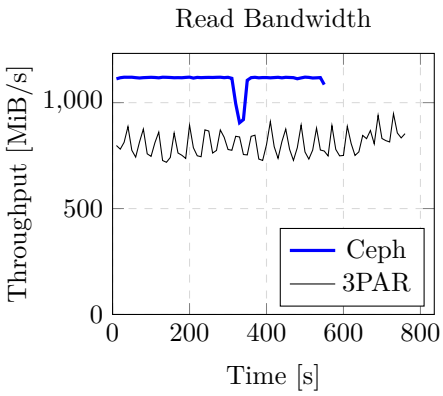
B.5 Sequential 4 MiB Access Plots

This section contains throughput plots of sequential Ceph and 3PAR access of a block size of 4 MiB.

B.5.1 RAID 0 and RAID 1

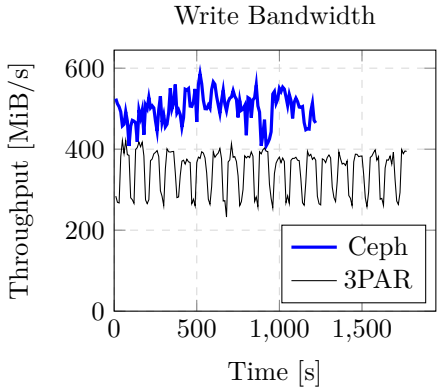


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

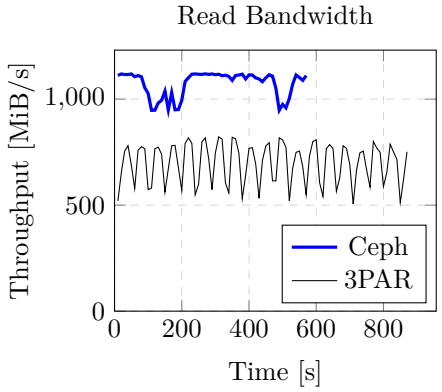


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.1: RAID 0 (1 replica) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

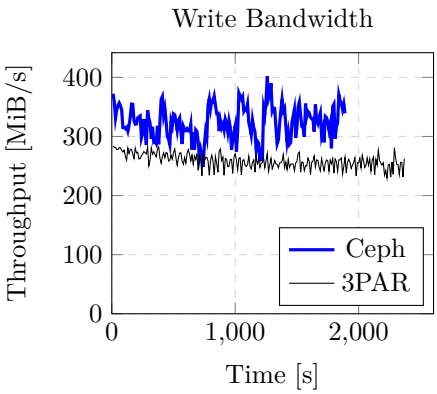


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

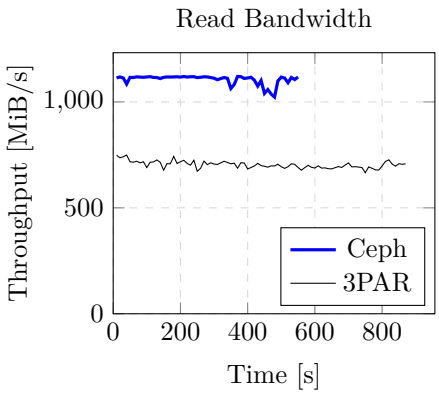


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.2: RAID 1 (2 replicas) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

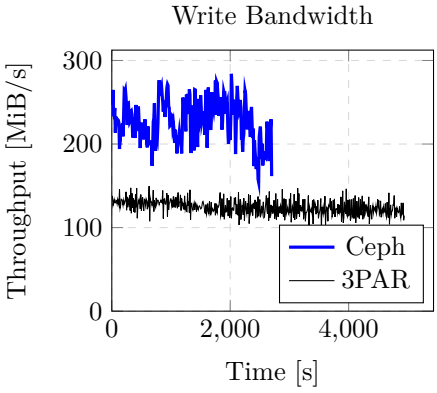


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

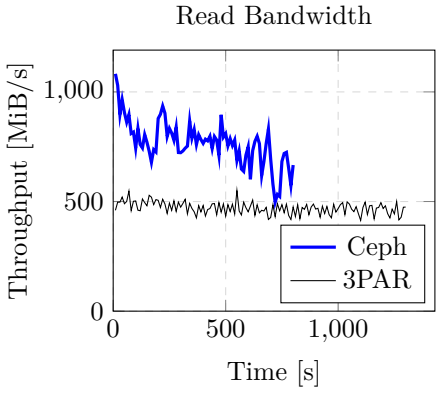


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.3: RAID 1 (3 replicas) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).



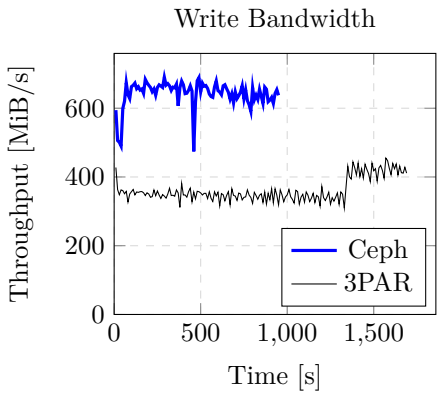
(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.



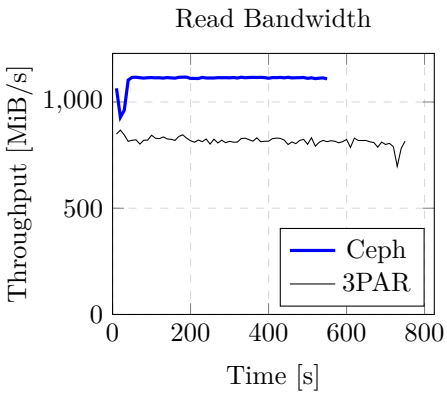
(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.4: RAID 1 (4 replicas) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

B.5.2 RAID 5

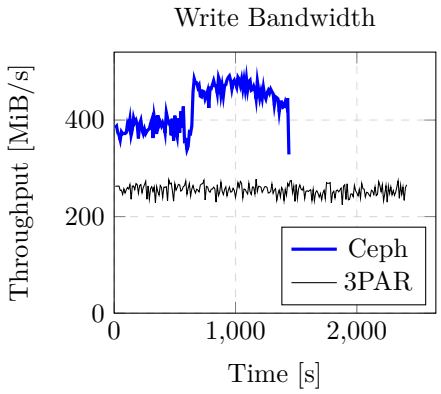


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

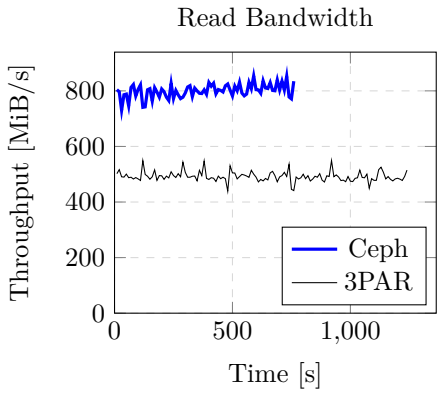


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.5: RAID 5 (2 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

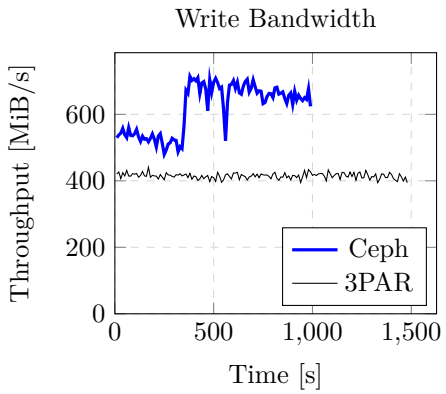


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

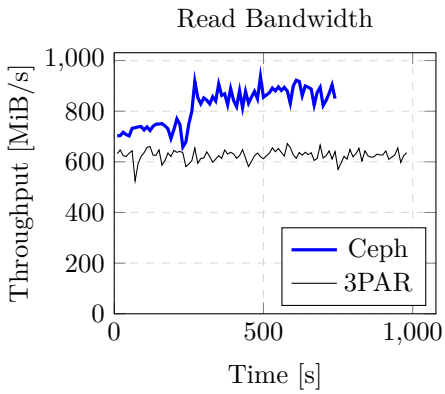


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.6: RAID 5 (3 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

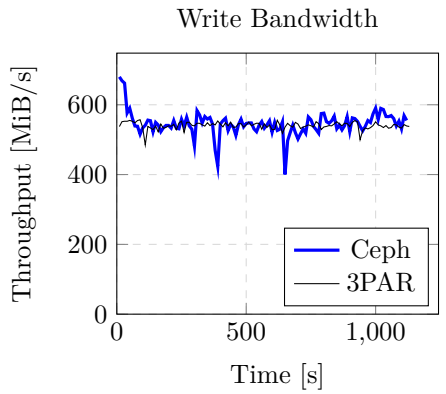


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

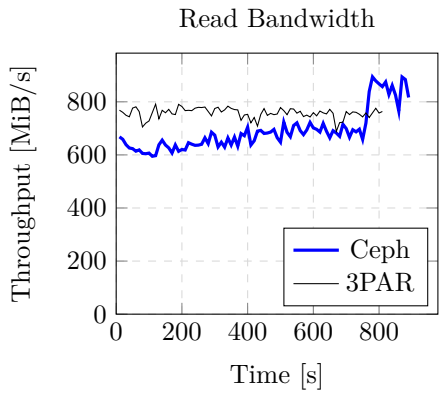


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.7: RAID 5 (4 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

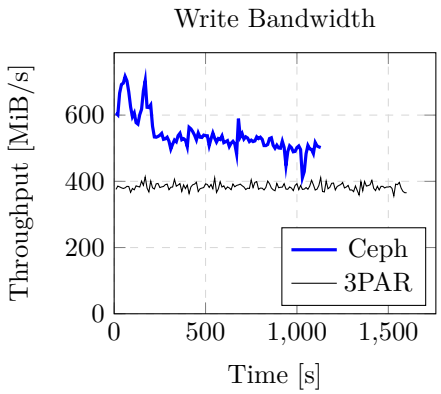


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

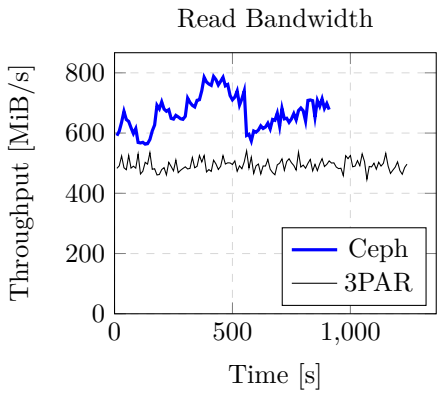


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.8: RAID 5 (5 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

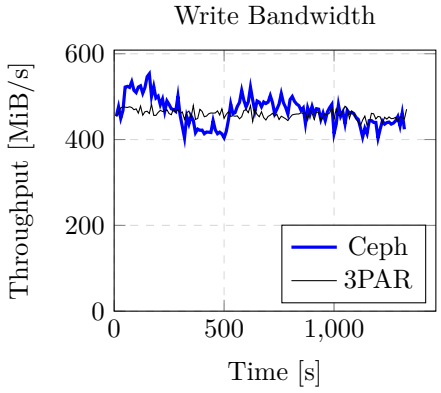


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

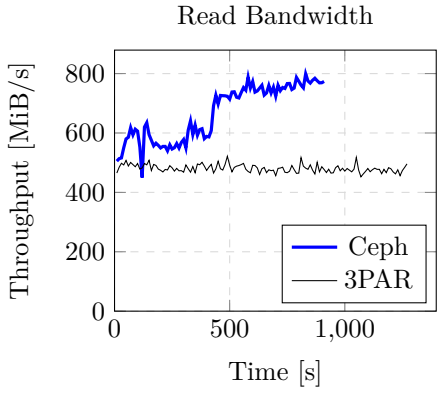


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.9: RAID 5 (6 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

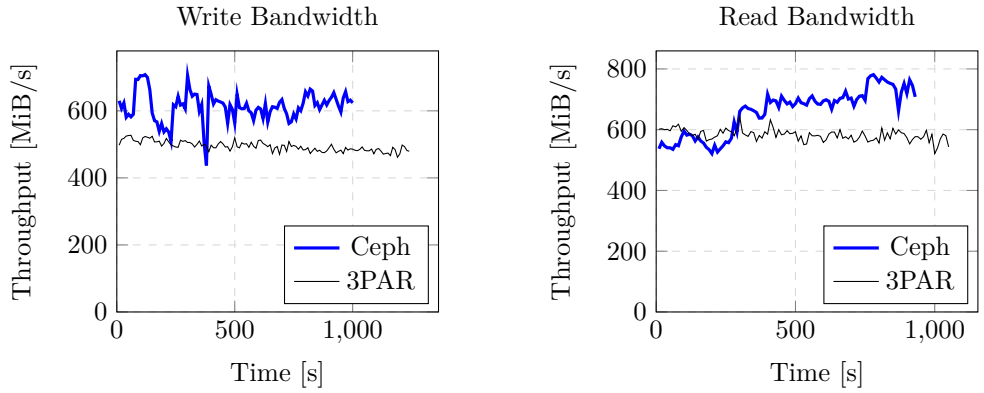


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.



(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.10: RAID 5 (7 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

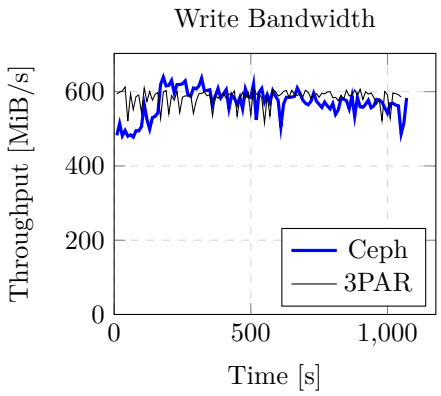


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

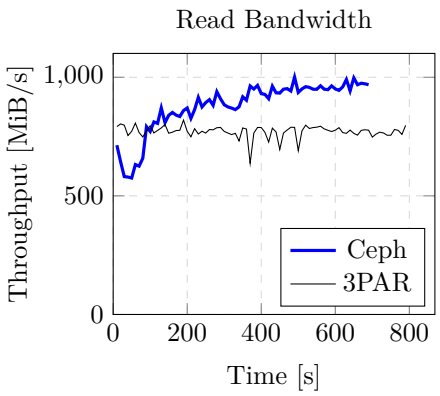
(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.11: RAID 5 (8 data, 1 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

B.5.3 RAID 6

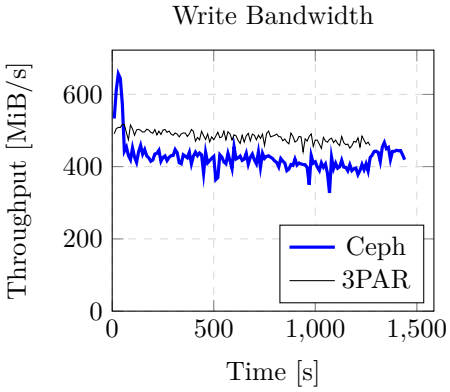


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

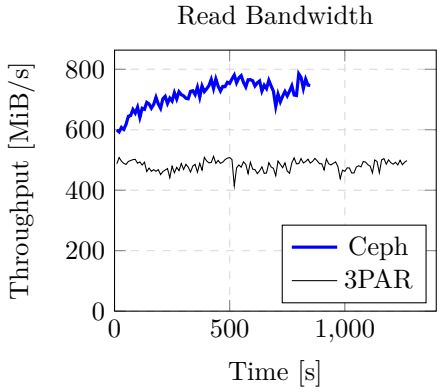


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.12: RAID 6 (4 data, 2 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

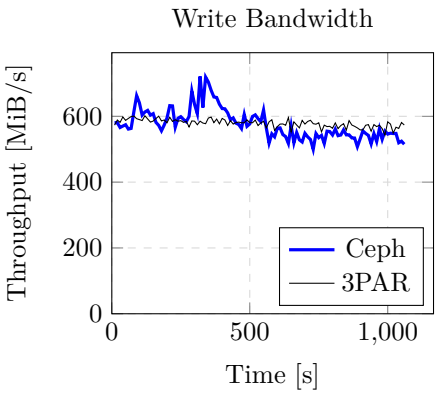


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

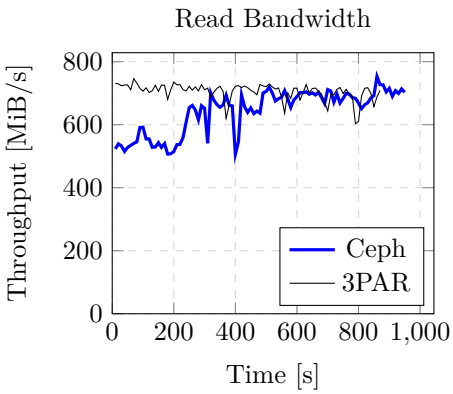


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.13: RAID 6 (6 data, 2 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

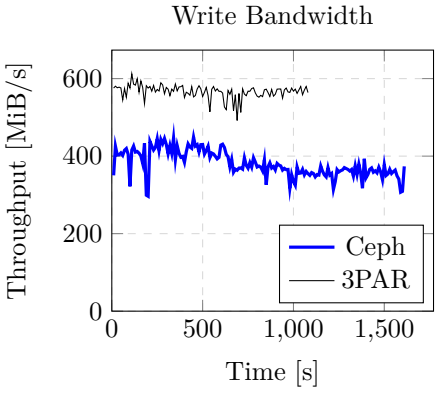


(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.

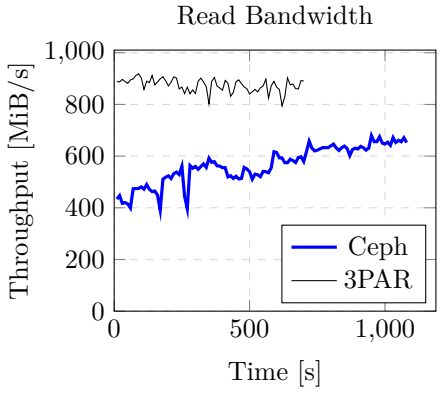


(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.14: RAID 6 (8 data, 2 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).



(a) Sequential write. Block size of 4 MiB, I/O depth of 128 requests.



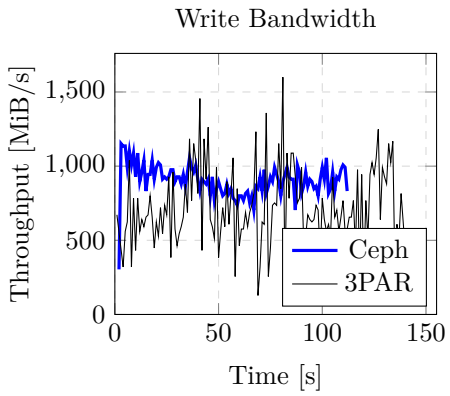
(b) Sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.15: RAID 6 (10 data, 2 parity) volume with a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

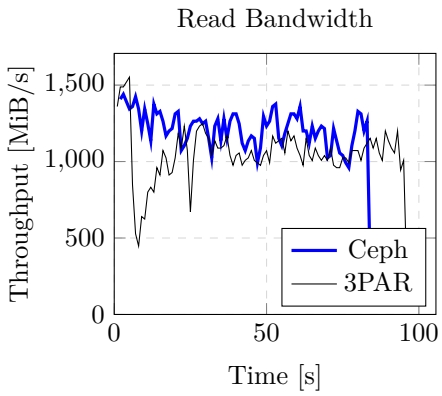
B.6 Sequential 16 MiB Access Plots

This section contains throughput plots of sequential Ceph and 3PAR access of a block size of 16 MiB.

B.6.1 RAID 0 and RAID 1

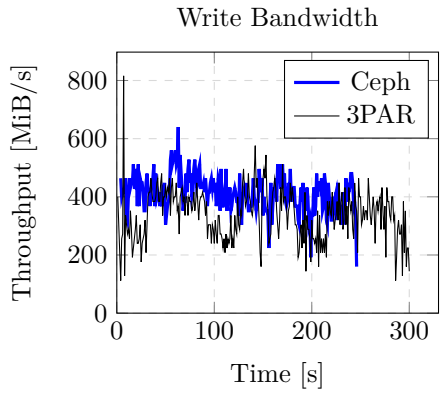


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

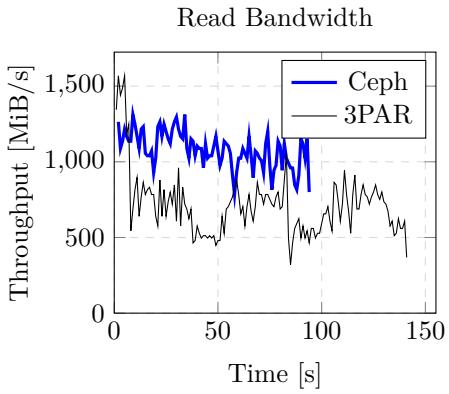


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.16: RAID 0 (1 data) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

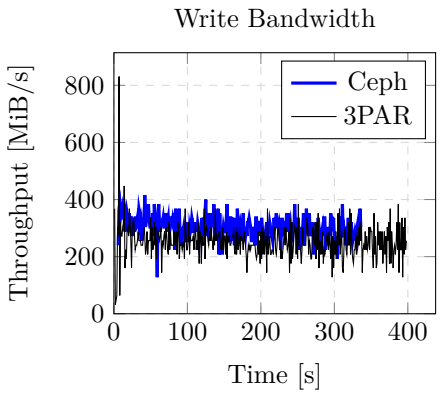


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

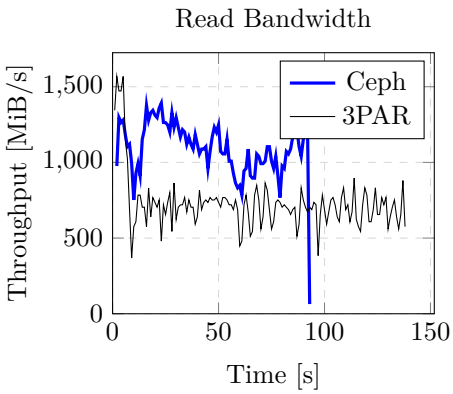


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.17: RAID 1 (2 data) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

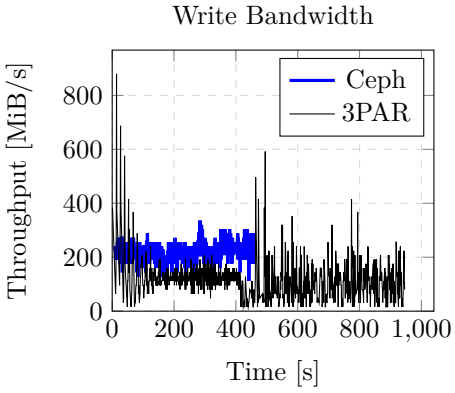


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

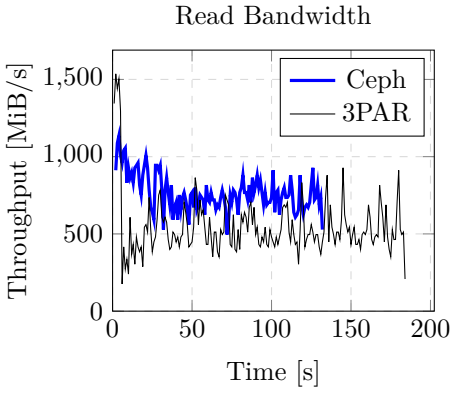


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.18: RAID 1 (3 data) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).



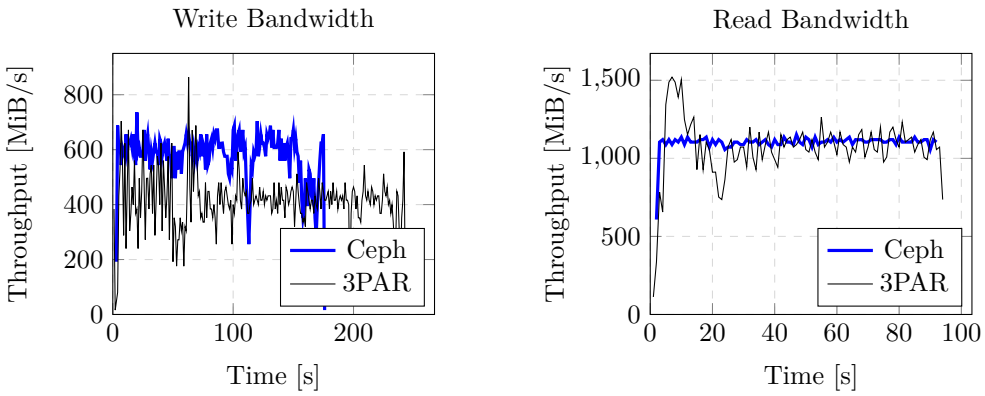
(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.



(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.19: RAID 1 (4 data) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

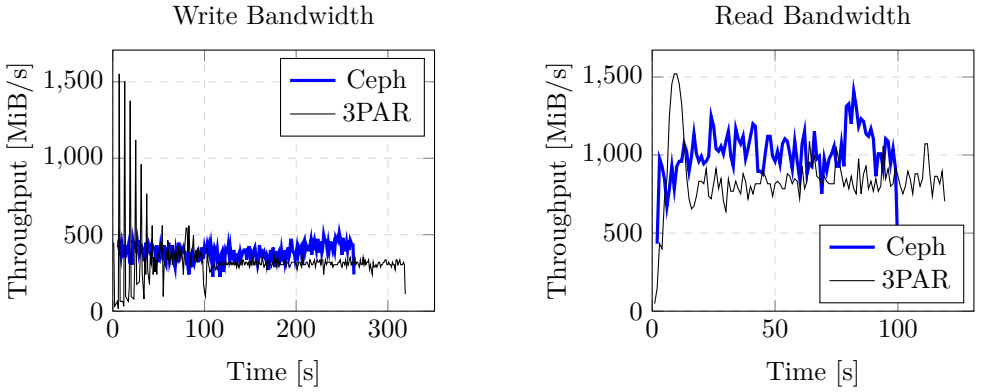
B.6.2 RAID 5



(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

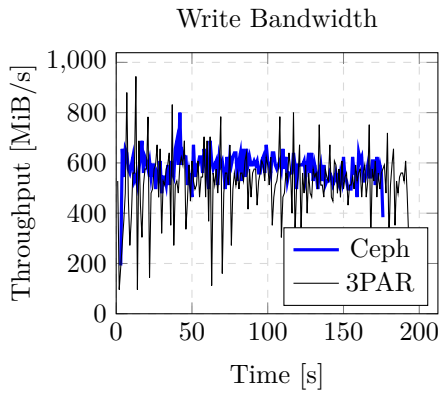
Figure B.20: RAID 5 (2 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).



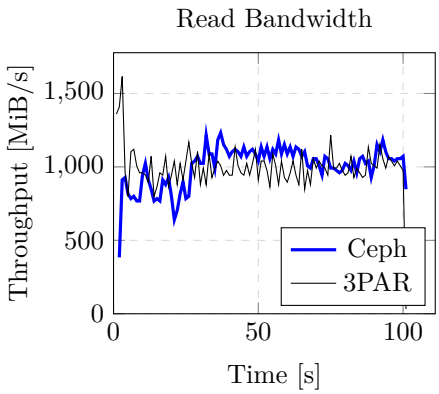
(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.21: RAID 5 (3 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

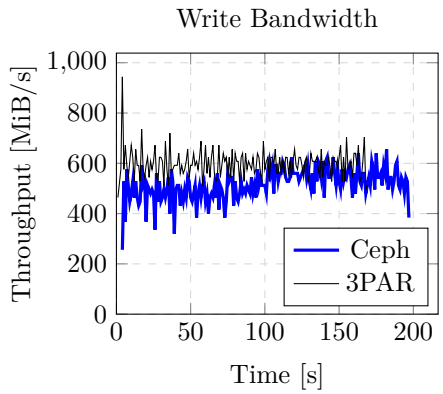


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

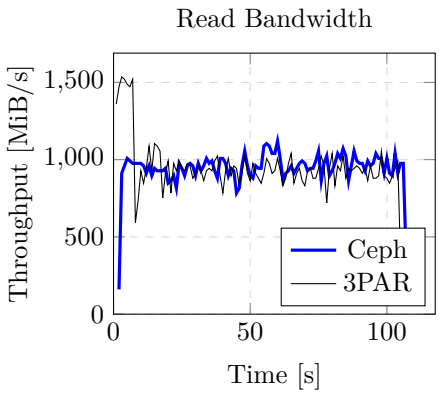


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.22: RAID 5 (4 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

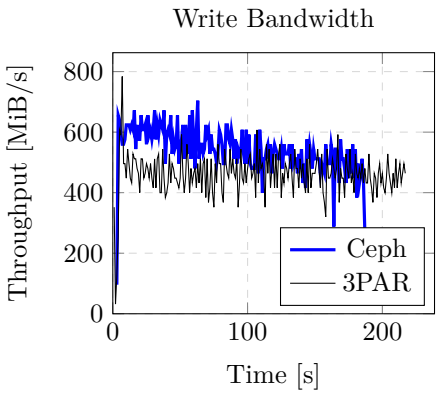


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

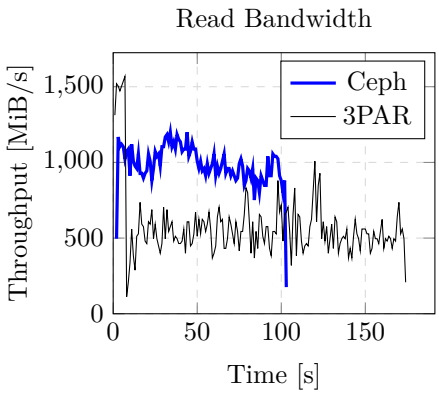


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.23: RAID 5 (5 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

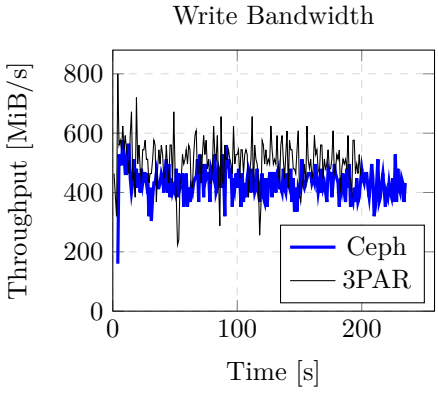


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

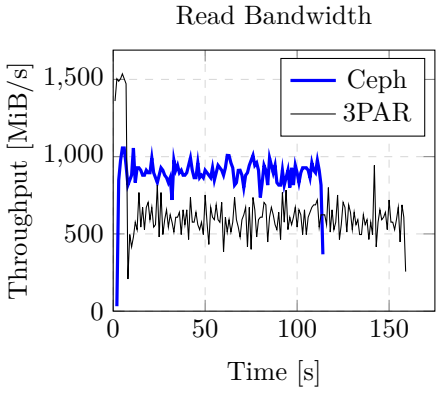


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.24: RAID 5 (6 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

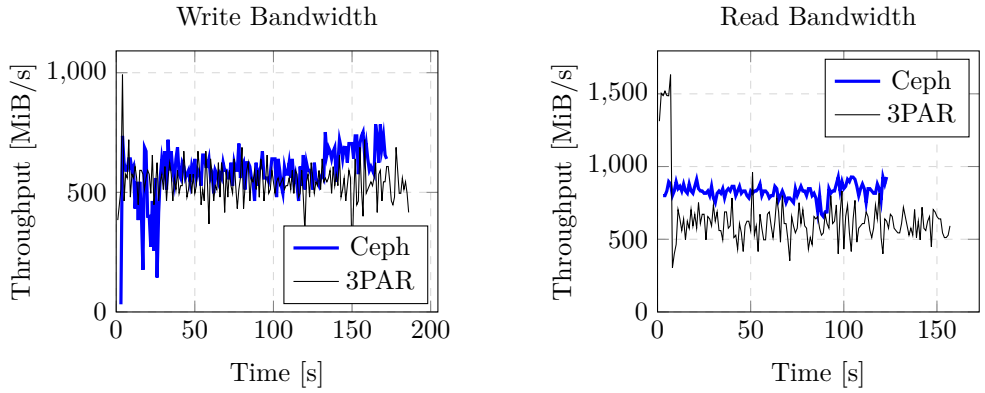


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.



(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.25: RAID 5 (7 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

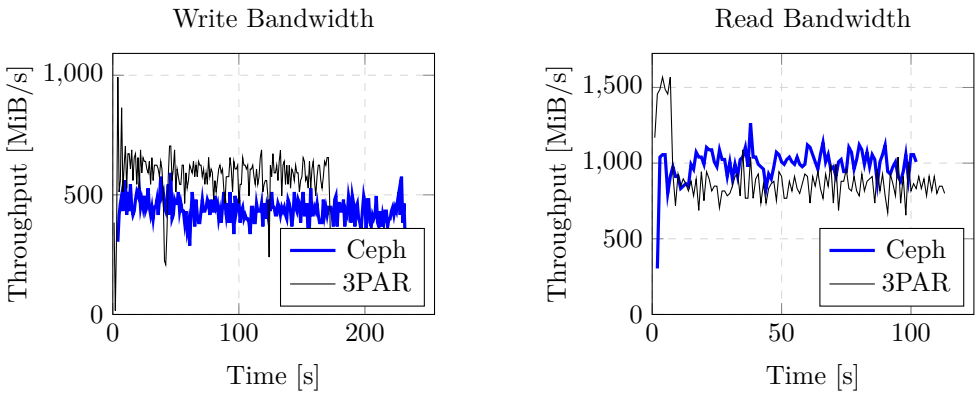


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.26: RAID 5 (8 data, 1 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

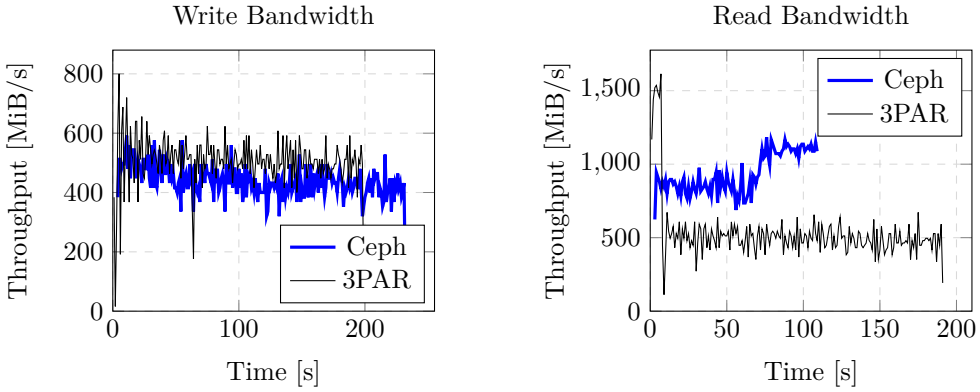
B.6.3 RAID 6



(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

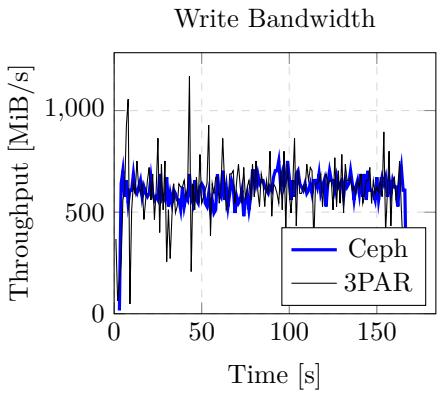
Figure B.27: RAID 6 (4 data, 2 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).



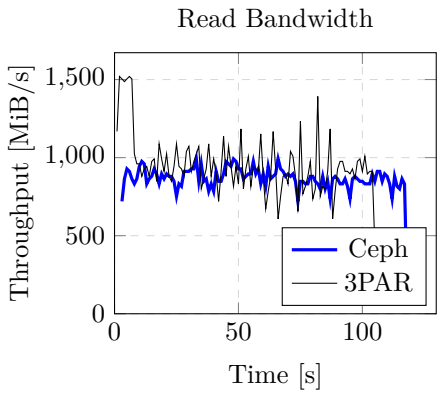
(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.28: RAID 6 (6 data, 2 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

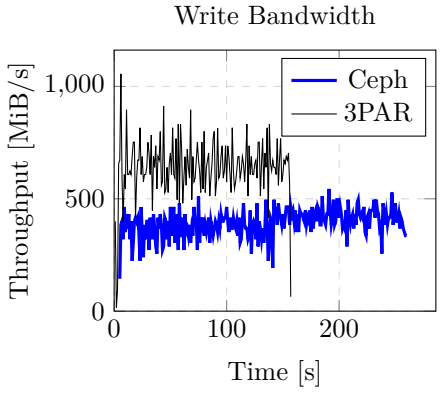


(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.

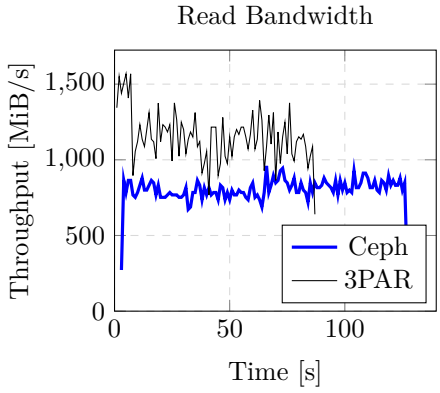


(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.29: RAID 6 (8 data, 2 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).



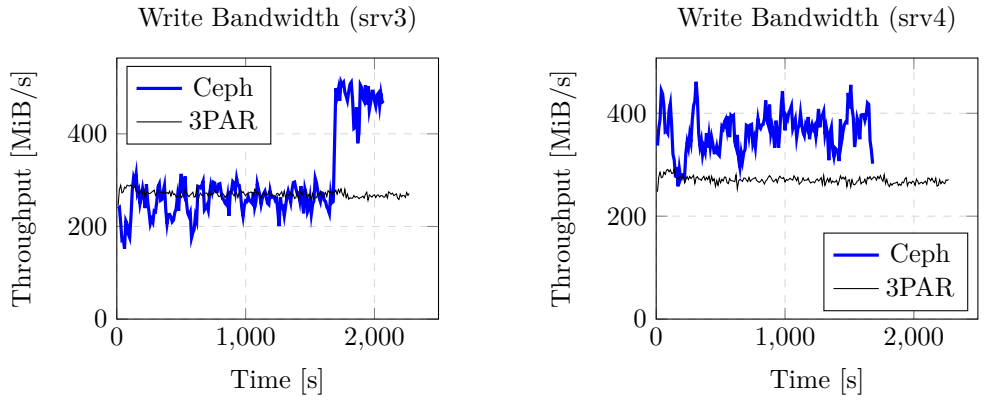
(a) Sequential write. Block size of 16 MiB, I/O depth of 128 requests.



(b) Sequential read. Block size of 16 MiB, I/O depth of 128 requests.

Figure B.30: RAID 6 (10 data, 2 parity) volume with a capacity of 100 GiB. Sequential access throughput (*fio* version 3.13).

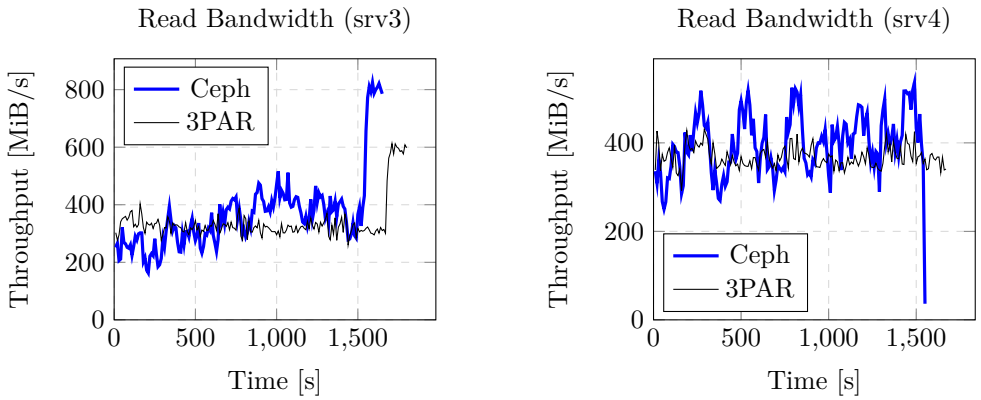
B.7 Multiple Hosts



(a) RAID 6 sequential write. Block size of 4 MiB, I/O depth of 128 requests.

(b) RAID 5 sequential write. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.31: Two volumes were simultaneously written to from two different client servers: RAID 6 (6 data, 2 parity) volume from srv3, RAID 5 (5 data, 1 parity) volume from srv4. Both volumes had a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

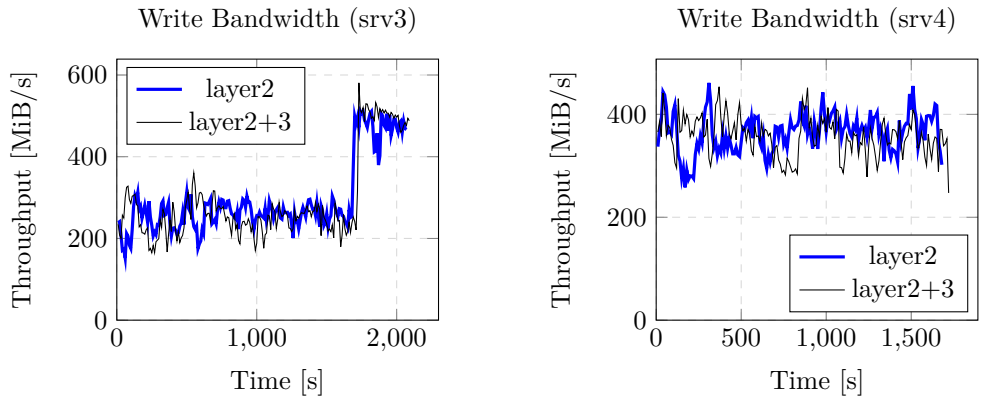


(a) RAID 6 sequential read. Block size of 4 MiB, I/O depth of 128 requests.

(b) RAID 5 sequential read. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.32: Two volumes were simultaneously read from from two different servers: RAID 6 (6 data, 2 parity) volume from srv3, RAID 5 (5 data, 1 parity) volume from srv4. Both volumes had a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

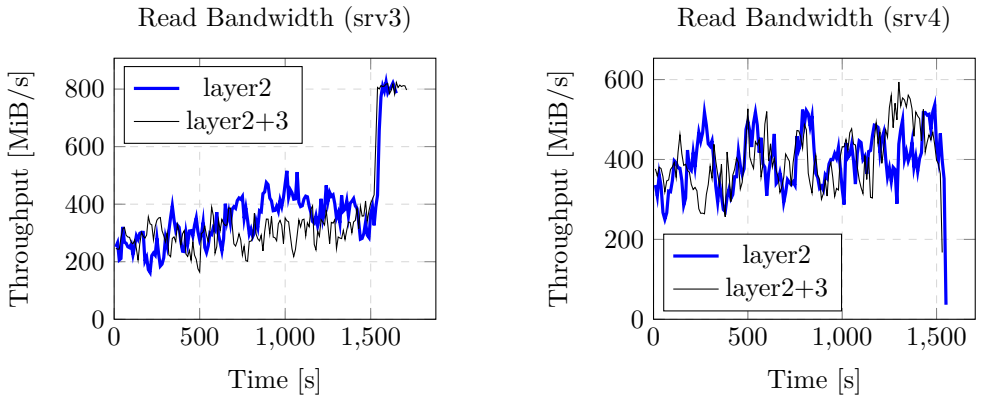
B.8 Effect of `xmit_hash_policy` Setting



(a) RAID 6 sequential write to Ceph. Block size of 4 MiB, I/O depth of 128 requests.

(b) RAID 5 sequential write to Ceph. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.33: Different *xmit_hash_policy* settings on bonded network interfaces on all servers. Two volumes were simultaneously written to from two different client servers: RAID 6 (6 data, 2 parity) volume from srv3, RAID 5 (5 data, 1 parity) volume from srv4. Both volumes had a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).



(a) RAID 6 sequential read from Ceph. Block size of 4 MiB, I/O depth of 128 requests.

(b) RAID 5 sequential read from Ceph. Block size of 4 MiB, I/O depth of 128 requests.

Figure B.34: Different *xmit_hash_policy* settings on bonded network interfaces on all servers. Two volumes were simultaneously read from from two different servers: RAID 6 (6 data, 2 parity) volume from srv3, RAID 5 (5 data, 1 parity) volume from srv4. Both volumes had a capacity of 600 GiB. Sequential access throughput (*fio* version 3.13).

Appendix C

Contents of Enclosed SD Card

- `attachments` ..... a directory with various files collected during the tests
  - `benchmark-logs` ..... an assorted raw benchmark data directory
  - `configuration-notes` ..... an assorted configuration notes directory
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `BP_Sebek_David_2019.pdf` ..... the thesis text in PDF format