Degree project

UNIVERSITY OF SKÖVDE

CLOUD-NATIVE STORAGE SOLUTIONS FOR KUBERNETES

A Performance Comparison

Bachelor Degree Project in Informatics

First Cycle 30 credits

Spring Term 2023

Student: Filip Andersson

Supervisor: Johan Zaxmy

Examiner: Thomas Fischer

ABSTRACT

Kubernetes is a container orchestration system that has been rising in popularity in recent years. The modular nature of Kubernetes allows the usage of different storage solutions, and for cloud environments, cloud-native distributed storage solutions may be attractive due to their redundant nature. There are many tools for cloud-native distributed storage available on the market today with differing features and performance. Choosing the correct one for an organisation can be difficult.

Organisations utilising Kubernetes in cloud environments would like to be as performance efficient as possible to save on costs and resources. This study aims to offer a benchmark and analysis for some of the most popular tools, to help organisations choose the 'best' solution for their operational needs, from a performance perspective.

The benchmarks compare three cloud-native distributed storage solutions, OpenEBS, Portworx, and Rook-Ceph on both Amazon Elastic Kubernetes Service (EKS) and Azure Kubernetes Service (AKS). For a baseline comparison, the study will also benchmark the cloud providers own solutions; Azure Disk Storage, and Amazon Elastic Block Storage. The study compares these solutions from three key metrics; bandwidth, latency, and IOPS, in both read and write performance.

Keywords: Kubernetes, Distributed Storage, Storage Performance, Benchmark, Cloud-Native, Bandwidth, Latency, IOPS

Table of Contents

1 Introduction

Kubernetes is a container orchestration system that has been rising in popularity in recent years. According to a survey by Red Hat, over 90% of all respondent businesses use Kubernetes for container orchestration and over 85% use it in deployment (Red Hat, 2022). One of the key features of Kubernetes is the way it orchestrates clustering to allow a highly available environment. The modular nature of Kubernetes allows the usage of different storage solutions, and for cloud environments, cloud-native distributed storage solutions may be attractive due to their redundant nature. There are many tools for cloud-native distributed storage available on the market today with differing features and performance. Choosing the correct one for an organisation can be difficult.

Organisations utilising Kubernetes in cloud environments would like to be as performance efficient as possible to save on costs and resources. This study aims to offer a benchmark and analysis for some of the most popular tools, to help organisations choose the 'best' solution for their operational needs, from a performance perspective.

The benchmarks compare three cloud-native distributed storage solutions, OpenEBS, Portworx, and Rook-Ceph on both Amazon Elastic Kubernetes Service (EKS) and Azure Kubernetes Service (AKS). The study compares these solutions from three key metrics; bandwidth, latency, and IOPS, in both read and write performance.

This report and work is largely based on the works of Mercl & Pavlik (2019), and aims to extend the relevancy of their benchmarks and methodology. This report takes inspiration from other benchmarks of storage solutions or Kubernetes, such as Cloudian (2023), Mehndiratta (2020), Mutai (2019), and Pereira Ferreira & Sinnott (2019).

This report will give a background into Kubernetes and distributed storage solutions, provide a description on how the benchmarks were conducted and how the experiment design was constructed, describe how the infrastructure and environment was set up, and lastly give an analysis and comparison of the final results from the benchmark.

2 Background

This chapter will explain the necessary background to familiarise oneself with Kubernetes, cloud environments and distributed storage, and the relationship between these components.

2.1 Kubernetes

Kubernetes, also known as K8s, is an open-source container orchestration system for automating deployment, scaling, and management of software. Kubernetes was originally created by Google based on the design of the cluster management software Borg. Google open-sourced the project in 2014, and the project is now being developed by the Cloud Native Computing Foundation (Kubernetes, 2022).

2014 was a breakthrough year for Kubernetes, with companies like Microsoft, Red Hat, IBM, Docker joining the Kubernetes community (Moravcik et al., 2022). Today, Kubernetes supports many organisations with its large and rapidly evolving ecosystem. Kubernetes can now be considered as the main tool available for container orchestration and management.

Kubernetes can be seen as a framework to run distributed systems resiliently. Kubernetes historically used Docker as their container backend, using their dockershim implementation. Nowadays they have migrated to Containerd as the default container runtime, but allows other runtimes to be used.

Kubernetes is not a PaaS (Platform as a Service) system, but provides many common associated features such as scaling, load balancing, app deployment, and lets users integrate logging, monitoring and alerting. It is not monolithic, meaning the default solutions are optional and pluggable to the users demands and needs.

2.2 Cloud environments

Kubernetes can either be run locally or commonly in cloud environments. While Kubernetes is still a relatively young software, there are many options available on what solution and platform to choose. Kubernetes in the cloud can either be run using a managed or unmanaged solution (BLUESHOE, 2021). Unmanaged, meaning that you have to do the installation and maintenance of Kubernetes, this is usually done using virtual machines as nodes. Managed, meaning the cloud provider provides the Kubernetes installation for you. According to a survey published in 2021, the most popular managed Kubernetes available were Amazon Elastic Kubernetes Service (EKS), and Azure Kubernetes Service (AKS) (The Cloud Native Computing Foundation, 2021).

Amazon Elastic Kubernetes Service is the managed service for using Kubernetes on AWS. It integrates AWS features and services such as Amazon Elastic Container Registry (ECR), load balancing, automatic scaling, high availability, and authentication (Amazon Web Services, 2023c).

Azure Kubernetes Service is the service for deploying a managed Kubernetes cluster in Azure. It offloads operational overhead to Azure, such as health monitoring and

maintenance. It also provides features such as networking, Azure Active Directory integration, and monitoring (Microsoft, 2023b).

2.3 Distributed storage solutions

Kubernetes containers generally operate on a stateless design, but in some cases a persistent storage may be wanted for databases, configuration or file storage. Because of the highly available nature of Kubernetes, choosing a distributed storage solution can be attractive due to it allowing the storage to be available even in case of a failure on one or more of the machines. There are a number of available cloud-native distributed storage solutions on the market today, with some popular platform independent choices being OpenEBS, Rook, Portworx, Gluster (The Cloud Native Computing Foundation, 2019, p. 201). Alternatively one could use the cloud providers own storage solutions, such as Azure Disk Storage, Amazon Elastic Block Storage, or Google Persistent Disk.

Azure Disk Storage is Microsoft's proprietary block storage for Azure Virtual Machines. The key feature of Azure Disk Storage is the built in integration with Azure (Microsoft, 2023a).

Amazon Elastic Block Store (EBS) is Amazon's proprietary block storage for Amazon Elastic Compute Cloud. The Amazon EBS Container Storage Interface (CSI) allows Amazon Elastic Kubernetes Service to take advantage of Amazon EBS Volumes (Amazon Web Services, 2023b).

OpenEBS is a Kubernetes native storage solution to easily turn storage on worker nodes into Kubernetes persistent volumes. It was originally created by DataCore, formerly MayaData, but the project is now being developed by the Cloud Native Computing Foundation (OpenEBS, 2021). Some of the key features of OpenEBS is its ease of use, consistency across all Kubernetes distributions and the fact that it is open source and free.

Portworx is a software defined storage overlay for container orchestration systems such as; Kubernetes, Red Hat Openshift, Docker Swarm, and more. Portworx is targeted towards enterprises and was created by Pure Storage. What makes Portworx stand out among the storage tools available, is their commercial features such as enterprise backup and support (Portworx, 2023b). While most other tools are open source, Portworx is a closed source solution.

Rook is a tool for cloud-native open source storage orchestration, it takes the storage provider Ceph, and provides integration for Kubernetes. Ceph is an open source distributed storage system, providing unified object, block, and file storage. Ceph is a well established software developed by the Ceph Foundation, an under-organisation to the Linux Foundation, with member organisations such as IBM, Canonical, Intel, among many more (Rook, 2023; The Ceph Foundation, 2023). Rook is developed by the Linux Foundation.

2.4 Disk Benchmarking

Benchmarks used for performance evaluations should be representative of applications that run on actual systems (Eeckhout, 2006). In the past, computer performance was evaluated by the speed of ADD or MULTIPLY instructions, however with the advancement of computers, evaluation of more complex operations is necessary. Today, the workloads may vary greatly, and it is not easy to create representative benchmarks. A great number of different benchmarks are available on the market today, of different types, for different types of workloads.

Flexible I/O tester (FIO) is an established software for generating IO workloads. FIO is made to be a simple to use tool, with emphasis on being flexible in the sense that it can be made to simulate the IO workloads you want. FIO was created by Jenx Axboe and is co-maintained with Vincent Fu (Axboe, 2012/2022).

For disk benchmarking on Kubernetes there are a couple of different choices available on the market. Dbench is one such tool, it tests Kubernetes disk volumes with different workloads using FIO. It performs nine test cases, benchmarking latency, bandwidth, and IOPS, in both read and write performance. Dbench was created by Lee Liu, from the organisation LogDNA, with contribution from Alexis Turpin (Liu, 2018/2023).

2.5 Previous works

Following the rise in popularity of Kubernetes and the trending research interest in cloud environments, a large number of benchmarks and evaluations have been made comparing different aspects and features of Kubernetes storage (Cloudian, 2023; Mehndiratta, 2020; Mercl & Pavlik, 2019; Mutai, 2019). However the research into performance of cloud-native distributed storage solutions, in combination with cloud environments, is limited.

Mercl & Pavlik (2019) conducted a benchmark using four different solutions on Azure Kubernetes Service (AKS); Portworx, OpenEBS, Rook-Ceph, and GlusterFS. For baseline comparison they also benchmarked two solutions using AKS built in features. Their study was performed using benchmarking tool dbench, with the default parameter values, evaluating the bandwidth, latency and IOPS of the storage solutions. Their conclusion was that Portworx could be considered the fastest storage solution. Rook-Ceph and OpenEBS were viable options, but OpenEBS could not be considered production ready. Rook-Ceph was good for hardware clusters, but considered too complex for public cloud solutions.

Perira Ferreira & Sinnott (2019) did a benchmark comparing and evaluating performance of cloud-native managed Kubernetes infrastructures. The research compared the following container metrics; CPU, memory, disk and network, of Amazon Elastic Kubernetes Service (EKS), Microsoft Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE), to a baseline unmanaged deployment on NeCTAR Research Cloud. Their study concluded that there was not a single solution that could be considered optimal, as it depends on the demands of the workload. The study also concluded the selection of adequate cloud resources was a key aspect of achieving the best performance.

3 Problem description

This chapter will describe the aim of the research, the motivation behind and what the goal is.

3.1 Research Question

The aim of the research could be formulated as the following question:

- From a performance perspective, what is the 'best' cloud-native distributed storage solution for Kubernetes in cloud environments?

3.2 Motivation and Research Aim

There are multiple solutions available on the market for using cloud-native distributed storage solutions for Kubernetes. Choosing the 'best' one can be challenging as they have differing features and are based on different technology.

To answer the research question, the study aims to compare and evaluate the performance of some of the most popular cloud-native distributed storage solutions. The choice of storage solutions to be compared, design of experiment, and methodology used, are based from a study by Mercl & Pavlik (2019). In large, this study will be replicating the previous study, but there are a couple of identified ways in which the relevancy of that study could be extended. The key points for extending the paper can be summarised as the following:

- The research does not account for performance variation between different platforms as the experiment was solely done on Azure Kubernetes Services.
- The research does not account for performance fluctuation affected by the currently available resources and load on the cloud providers infrastructure.
- The research was conducted in 2019 and may be therefore considered outdated, due to the rapid development of cloud environments, Kubernetes platforming, and storage solutions.

The results of this study aims to offer a metric and analysis that can be used for helping when choosing what the 'best' solution would be from a performance perspective. For geographical purposes this study is targeted towards enterprises and consumers in northern Europe.

3.3 Delimitations

Mercl & Pavlik (2019) benchmarked GlusterFS using Heketi for RESTful integration. However Heketi is now in 'deep maintenance mode' and not being actively developed anymore. As the creators of Heketi discourage the usage of the tooling, GlusterFS/Heketi will therefore not be evaluated in this research (Heketi, 2015/2023).

4 Method

This chapter will describe what method will be used in the research and the motivation behind the choice. This chapter will also describe the methodologies used and how they will be implemented.

Experimental methodology was deemed the most appropriate due to it providing metrics that can be measured for objective and reliable results. Choice of experimental methodology was based on previous work with similar aims of evaluating performance of storage solutions. In particular the works of Mercl & Pavlik (2019) and Pereira Ferreira & Sinnott (2019), where they benchmarked cloud-native storage solutions for Kubernetes, and container performance for Kubernetes services respectively.

4.1 Methodological Approach

As this report is largely based on the work by Mercl & Pavlik (2019), it will use the same methodological approach as they have used. The methodological approach proposed by Mercl & Pavlik (2019) is rather simple and follows a four step implementation and test procedure. In addition to the implementation and test procedure, the Kubernetes infrastructure needs to be prepared, and a test procedure needs to be established.

1. Configuration of external third-party storage services (if any),
2. Configuration of storage back-end on Kubernetes cluster side (creating storage class, persistent volume and/or persistent volume claims),
3. Deployment of test pod for testing and mounting volume to this pod,
4. Automatic test procedure.

4.1.1 Infrastructure

The Kubernetes infrastructure consists of a provisioned cluster using three virtual machines with a one terabyte empty disk for each instance. Mercl & Pavlik (2019) deployed the cluster solely on Azure Kubernetes Services, this study however will be using Amazon Elastic Kubernetes Services in addition, to compare the results between the platforms.

4.1.2 Test procedure

The test procedure follows the same as Mercl & Pavlik (2019), and consists of a load test using a Kubernetes disk benchmark tool, dbench (Liu, 2018/2023). Dbench generates I/O workloads using the tool Flexible I/O Tester (FIO) with multiple test cases. The test cases benchmarks the following variables:

- Random read and write bandwidth
- Random read and write IOPS
- Read and write latency
- Sequential read and write bandwidth
- Mixed read and write IOPS

4.2 Methodological Principles

For reproducibility and to improve validity, reliability and accuracy, this study will also follow the methodological principles proposed by Papadopoulos et al. (2021). These principles suggest how the experiment should be designed, what information should be made available for the reader, and how to analyse the results. As the methodology proposed in the work of Papadopoulos et al. (2021) is merely a set of principles, following this methodology does not conflict with following the approach from Mercl & Pavlik (2019).

Papadopoulos et al. (2021) recommends the following eight principles for benchmarks on cloud computing platforms:

1. **Repeated experiments (statistical)** Decide how many repetitions of the same configuration, based on the identified sources factors of variability and then quantify the confidence in the final results.
2. **Workload and configuration coverage** Experiments should be conducted using different configurations using randomised parameter values based on realistic probabilistic distributions or using historical data.
3. **Experimental setup description** Descriptions of hardware, software and environment setups, and parameters must be provided. Any information about the configuration should be included, such as operating system or software versions.
4. **Open access artefact** Artefacts should be publicly available to the scientific community. Digital objects created by the authors to be used in the experiment or generated as a result, should be included. For example these can be scripts used, input datasets, or raw data collected.
5. **Probabilistic result description of measured performance** Report characterizations of empirical distributions. This includes the aggregated values, but also their variations and the confidence in these values. The aggregations and summarizations methods have to be carefully selected to appropriately present the data based on distribution of data points.
6. **Statistical evaluation** Provide statistical evaluation of significance of results. The likelihood and representatives are essential in establishing the validity of claims derived from the conclusions.
7. **Measurement units** Report the unit of measurement for any reported quantity, to be able to accurately present results, compare with other approaches and analyse the correctness.
8. **Cost** Report the following about the experiment.
    a. Cost model used or assumed for experiment
    b. Accounted resource usage (per second)
    c. Charged cost according to the model

4.3 Validity

Wohlin et al. (2012) identifies four types of validity threat types that need to be taken into account to ensure adequate validity in the results: conclusion, internal, construct and external validity. Each type has a number of threats associated that may impact the validity of the experiment.

The identified conclusion validity threats are as follows; Low statistical power, which is the ability to find patterns in the data and ensure that the correct conclusion is drawn. Reliability of measures, which is how to measure a phenomenon to ensure the same outcome. Random irrelevancies in experimental settings, which in this study is how the result may be affected by random latency differences or spikes in cloud infrastructure

load. All of these threats will be remediated by running multiple tests at differing points in time.

Internal validity threats are only applicable to studies involving human subjects as they refer to handling of control groups and social threats.

The identified construct validity threats are as follows; Inadequate preoperational explication of constructs, refers to the importance of clearly defined constructs, before they are translated into measures or treatments. In this study it refers to the definition of 'performance' and 'best', and is remediated by referring to previous literature.

The identified external validity threats are as follows; Interaction of setting and treatment, meaning the effect of the study not being representative. This is ensured by using up-to-date tools and that the tools are considered to be industry standard. Interaction of history and treatment, is how the result may be affected by time of day or date. For this experiment the main variable is the available cloud resources and infrastructure load at a time, and is remediated by running tests at different points of time.

5 Experiment Design

This chapter will describe the infrastructure, information about the setup, including versioning and specifications. In accordance with principle 4 of Papadopoulos et al. (2021), the configuration files for setting up the cluster, setting up storage solutions, and deploying the test is available as a data set (Andersson, 2023).

5.1 Infrastructure

The tests are performed on Azure Kubernetes Service and Amazon Elastic Kubernetes Service using a three node cluster with an empty one terabyte disk on each node. (Mercl & Pavlik, 2019) does not specify what machine and disk types were used on Azure in their study. However using general purpose machine and disk types seem to be the most applicable, to increase impartiality of results.

Perira Ferreira & Sinnott (2019) conducted a study comparing the performance of managed kubernetes services and an unmanaged provider. Their study concluded that the selection of adequate resources was a key aspect to achieving good performance. In addition to resource requirements, their reasoning for choice of machine types was to utilise general purpose VMs to ensure impartiality of results. This study bases the machine types on the work of Pereira Ferreira & Sinnott (2019). For Azure the machine type Standard D2s v3 was chosen. For AWS, t3.large, was chosen instead of t2.large as it is now deprecated in favour of the newer version.

	Azure	AWS
Machine type	Standard D2s v3	t3.large
Physical CPU	Intel Xeon CPU E5-2673 v3	Intel Xeon Skylake 8175M or Intel Xeon Cascade Lake 8259CL
CPU specifications	12 Core 24 Thread @ 2.40GHz	24 Core 48 Thread @ 2.50GHz
vCPU cores	2	2
Memory	8 GiB	8 GiB
		Turbo CPU clock speed of up to 3.1 GHz

Table 6.2. Machine type specifications - overview

For geographical proximity, all tests were deployed on North European servers, located in Sweden, for both AWS and Azure.

The experiments were designed to be as similar as possible between the platforms. The storage solutions follow the same versioning between Azure and AWS, however for Kubernetes; Azure uses version 1.24.10 and AWS uses a modified version 1.24.11.

5.1.1 Azure Kubernetes Service

The Azure infrastructure was deployed using the provisioning tool Terraform, with scripts based from HashiCorp (HashiCorp, n.d.-a). The configuration files specify variables such as the machine types, cluster size, server region, and Azure resource group. The modified configuration file is available as a data set (Andersson, 2023). The Azure cluster was deployed with the following specifications:

As previously mentioned, for the machine type, Standard D2s v3, was used. The machine features an Intel Xeon CPU E5-2673 v3, with 2 vCPU cores and 8 GiB memory. The machine type is targeted towards balanced workloads, ideal for testing and development, small to medium databases, and low to medium traffic web servers (Microsoft, 2023c).

The benchmark disk used was Premium SSD LRS (P30). The max IOPS is 5000, and max throughput is 200 MBps.

The cluster was deployed using Kubernetes version v1.24.10, with a free tier, meaning no associated costs.

5.1.2 Amazon Elastic Kubernetes Service

The AWS infrastructure was deployed using the provisioning tool Terraform, with scripts based from HashiCorp (HashiCorp, n.d.-b). The configuration files specify variables such as the machine types, cluster size, server region, and AWS security group. The modified configuration file is available as a data set (Andersson, 2023). The AWS cluster was deployed with the following specifications:

As previously mentioned, for the machine type, t3.large was chosen. The machine features Intel Xeon Scalable processors, either Skylake 8175M or Cascade Lake 8259CL, with 2 vCPU cores and 8 GiB memory. The machine type is targeted towards general purpose, with a balance of computation, memory, and networking resources, ideal for applications such as web servers and code repositories (Amazon Web Services, 2023a).

The benchmark disk used was General Purpose SSDs (gp2), which utilises scaling performance per disk size. For a 1 terabyte disk, the max IOPS is 3000, and max throughput is 250 MiB/s.

The cluster was deployed using Kubernetes version v1.24.11-eks-a59e1f0. The costs associated with deploying the infrastructure using the above configuration were according to the following model:

- $0.01 per GB - DataTransfer-Regional-Bytes in EU (Stockholm)
- $0.1045 per GB - General Purpose SSD (gp2) provisioned storage in EU (Stockholm)
- $0.0864 per hour - Linux t3.large instance
- $0.10 per hour - Amazon EKS cluster usage in EU (Stockholm)

5.2 Setup

For relevancy the most recent stable version was used for each storage solution at the time of conducting experiments. The same versioning was used for the experiments on Azure Kubernetes Service and Amazon Elastic Kubernetes Service, to increase the consistency between the platforms.

In addition to the three storage solutions, this study is comparing them to the infrastructure providers specific solutions. The cloud provider solutions provide a baseline comparison. Azure uses Azure Disk Storage, and AWS has their Amazon EBS. The provider specific solutions do not require any additional setup, apart from the infrastructure.

OpenEBS requires iSCSI support enabled on the host machines. On Azure this is handled by the initialization of the operator. AWS however requires an additional external initiator to enable the features. OpenEBS can be set up using a number of different engines, these benchmarks were done using cStor engine with version 3.4.0. OpenEBS was set up using the default guide (OpenEBS, 2021).

Rook-Ceph was set up using the default guide with version 1.9.2 (Rook, 2023). See data set for configurations file used (Andersson, 2023).

Portworx was set up using the default guide with version 2.13 (Portworx, 2023b). See data set for configurations file used (Andersson, 2023).

5.3 Experimental tests

5.3.1 Dbench

The test procedure consists of a load test using a Kubernetes disk benchmark tool, dbench (Liu, 2018/2023). Dbench operates by running the following nine fio (Flexible I/O tester) tests, using the below parameters:

- Random Read IOPS & Random Write IOPS
    - Blocksize 4 kilobyte
    - IOdepth 64
- Random Read Bandwidth & Random Write Bandwidth
    - Blocksize 128 kilobyte
    - IOdepth 64
- Random Read Latency & Random Write Latency
    - Blocksize 4 kilobyte
    - IOdepth 4
- Read Sequential Speed & Write Sequential Speed
    - Blocksize 1 megabyte
    - IOdepth 16
    - Numjobs 4
- Random ReadWrite Mixed IOPS
    - Blocksize 4 kilobyte
    - IOdepth 64
    - Read 75% / 25% Write

The mixed read and write IOPS is a combined singular test, while the others are separated by read and write testings. Each test run has a ramp time of two seconds before recording the average performance over a 15 second period. The sequential speed tests operate by running four consecutive jobs and reporting the average values over all jobs.

5.3.2 Test design

According to principle 1 of Papadopoulos et al. (2021) the number of repetitions should be based on identified sources of variability, and have to account for random variability, required accuracy, and acceptable probability. Based on the sources of validity threats in Chapter 4.4, the selection of repetitions and test occasions can be determined to have to follow the following requirements;

1. **Multiple repetitions** To account for random latency differences, spikes in cloud infrastructure load, or other anomalies.
2. **Multiple occasions** At different days, including both weekdays and weekends.
3. **Differing points of time** During and outside office hours to account for the normal daily cloud infrastructure load fluctuation patterns.
4. **Avoiding major performance fluctuations** from seasonal events such as holidays or major events, such as sports, where the server infrastructure load may be expected to be abnormal.

According to principle 2 of Papadopoulos et al. (2021), workload and configuration coverage should be based on realistic probabilistic distributions. The test design uses dbench with the default parameters. This only partially fulfils principle 2 since the experiment uses multiple workloads, however workload coverage is not considered, and not based on randomised parameter values.

One major variable that could affect the results is the current infrastructure load and available cloud resources, and neither Azure or AWS publicises that data. From cloud usage patterns we could predict the current infrastructure load. A study showed the data purchase fluctuation, i.e. the reserved resources, to be between 2-6%, and the actual data usage fluctuation to be between 15-20% for data centres (Kilcioglu et al., 2017). However the cloud infrastructure load may not necessarily reflect the available cloud resources as the performance scheduling of Azure and AWS is not public. The above requirements are designed to account for the effect of performance fluctuations from such events.

Initially we determined the number of occasions to be four, with five repetitions at each occasion, totaling 20 repetitions per solution. If the results after running a number of tests show a considerably high or low variability at a time, the number of occasions may be lowered or increased post hoc, with a minimum of three occasions.

6 Results

This chapter will present the results and give a description of the results after running the experiments. In accordance with principle 5 of Papadopoulos et al. (2021), the results are reported using a table with averaged values and box plots, to report the aggregated values based on the distribution of data, including the variation. Line represents the highest and lowest value, the box represents the 1st and 3rd quartile, and the line in the box represents the median value. The measurement units are reported according to principle 7 of Papadopoulos et al. (2021).

6.1 Azure Kubernetes Service

Table below shows the average values for all repetitions and occasions on Azure. For results at each repetition, see data set (Andersson, 2023).

	Azure Disk Storage	Portworx	OpenEBS	Rook-Ceph
Random read bandwidth (MiB/s)	186.3	668.25	17.88	256.55
Random write bandwidth (MiB/s)	122.7	57.1	38.25	86.425
Random read IOPS	3991.75	35745	2389.9	3364.35
Random write IOPS	975.25	6029.05	1537.55	2782.8
Read latency (µsec)	1007.52	257.18	2528.44	3121.93
Write latency (µsec)	4157.31	756.25	3024.92	7076.00
Sequential read (MiB/s)	194.5	805.1	9.99	261.85
Sequential write (MiB/s)	192.65	59.55	47.05	86.67
Mixed read IOPS	1703	18800	1652.1	2783.5
Mixed write IOPS	569.25	6235.9	551.3	928.95

Table 6.1. Average values of all repetitions and occasions - Azure

6.1.1 Random read and write bandwidth

Portworx shows a considerably higher random read bandwidth compared to the second best performing solution, Rook-Ceph, with more than double the speed. Random write bandwidth shows less of a difference between solutions, however with Azure Disk Storage outperforming the other storage solutions. OpenEBS showed the lowest random read and write bandwidth, with read speed being considerably lower compared to the other solutions.

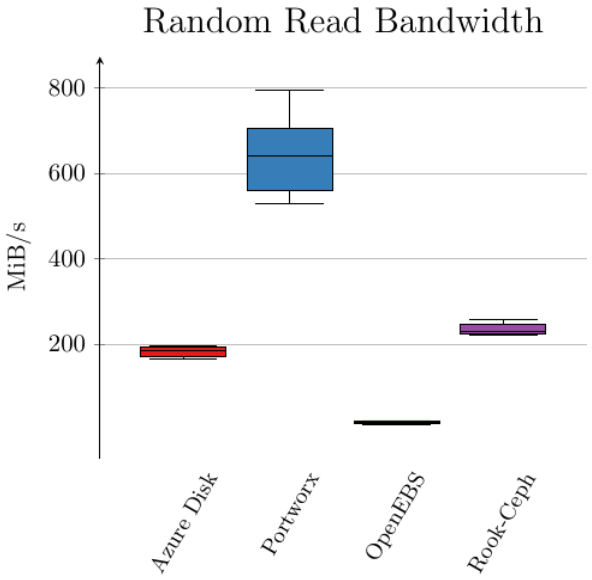

Fig 6.1. Box plot displaying the random read bandwidth in mebibyte per second on Azure.

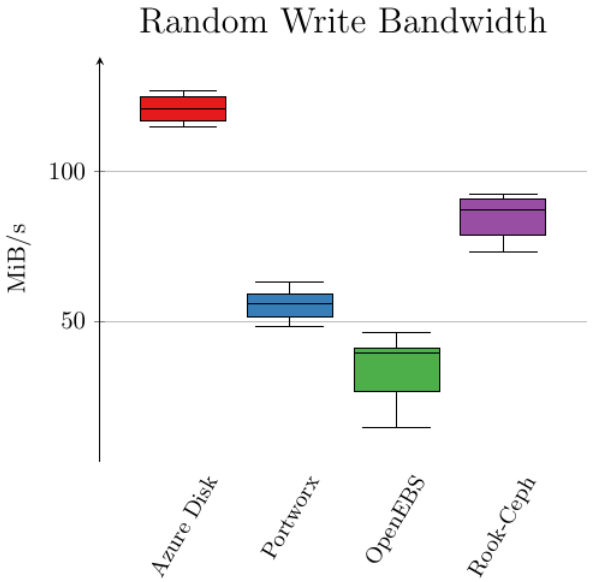

Fig 6.2. Box plot displaying the random write bandwidth in mebibyte per second on Azure.

6.1.2 Random read and write IOPS

Portworx showed a considerably higher random read and write IOPS compared to the other solutions, however with a high coefficient of variability (CV) for read, at 20,08%, compared to second highest CV, OpenEBS, at 10,52%. Read IOPS was similar for the other solution, while Rook Ceph outperformed OpenEBS and Azure Disk Storage in write IOPS.

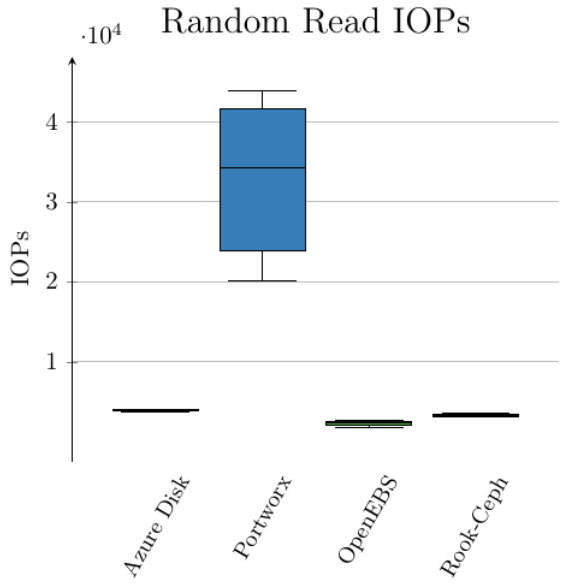

Fig 6.3. Box plot displaying the random read I/O per second on Azure.

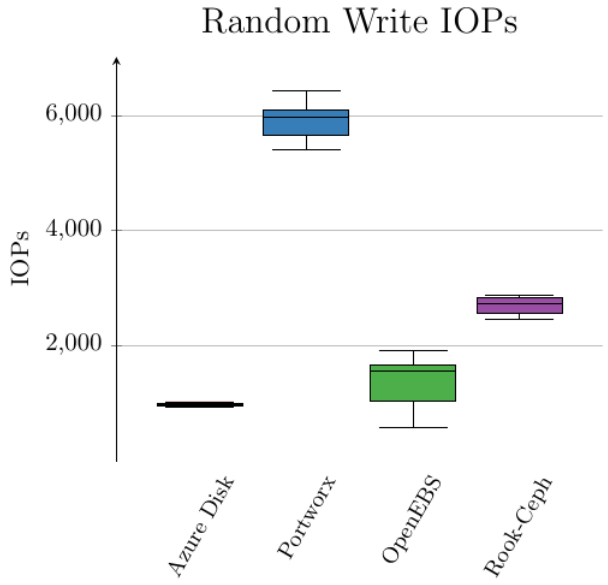

Fig 6.4. Box plot displaying the random write I/O per second on Azure.

6.1.3 Read and write latency

Portworx showed a considerably lower read and write latency compared to the other solutions. Rook Ceph had the highest latency in both read and write performance.

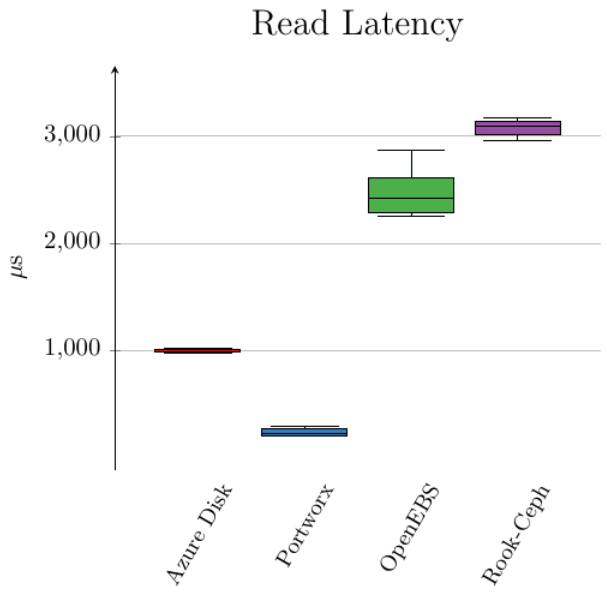

Fig 6.5. Box plot displaying the read latency in microseconds on Azure.

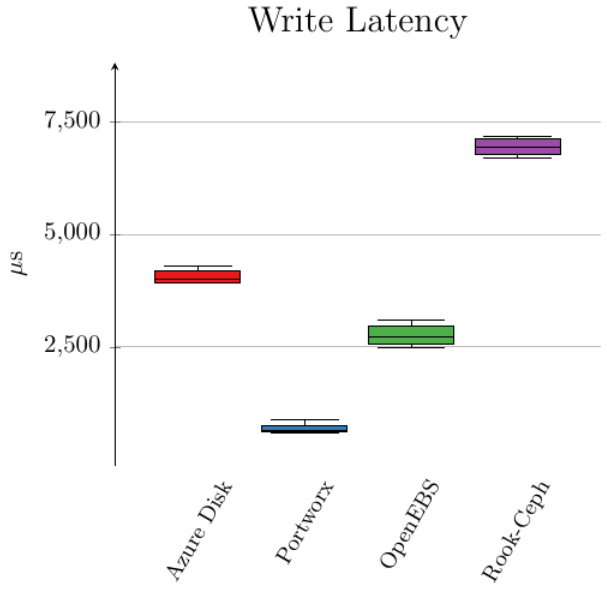

Fig 6.6. Box plot displaying the write latency in microseconds on Azure.

6.1.4 Sequential read and write

Portworx showed a higher sequential read speed, compared to the other solutions. Write performance was similar between Portworx, at 255,5 MiB/s, Rook Ceph, at 252 MiB/s, and Amazon EBS following, at 246 MiB/s. OpenEBS was considerably slower in both random read and write bandwidth.

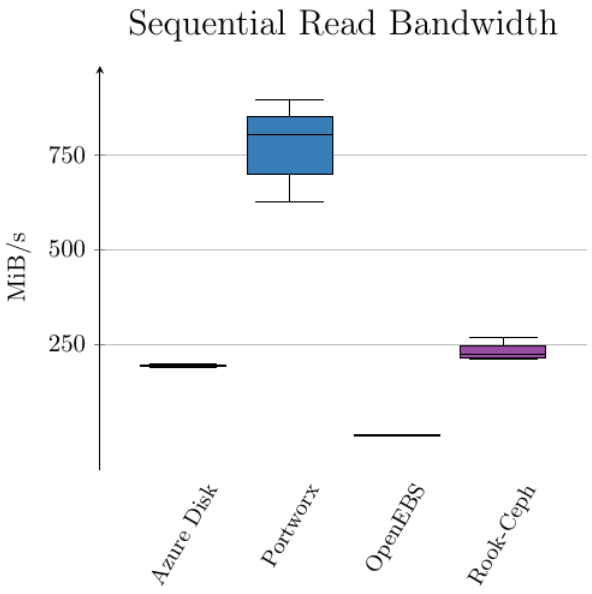

Fig 6.7. Box plot displaying the sequential read bandwidth in mebibyte per second on Azure.

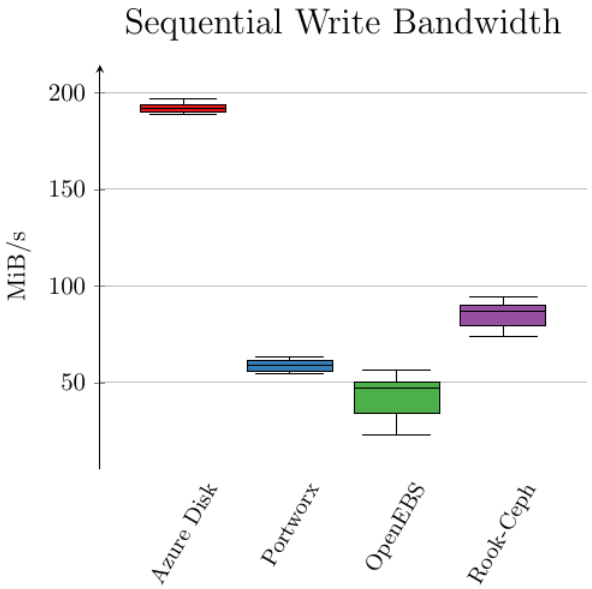

Fig 6.8. Box plot displaying the sequential write bandwidth in mebibyte per second on Azure.

6.1.5 Mixed read and write IOPS

Portworx showed a considerably higher read and write IOPS compared to the other solutions, followed by Rook Ceph. Performance between Azure Disk Storage and OpenEBS was comparative in both read and write IOPS.

Fig 6.9. Box plot displaying the mixed read and write I/O per second on Azure.

6.2 Amazon Elastic Kubernetes Service

Table below shows the average values for all repetitions and occasions on Azure. For results at each repetition, see data set (Andersson, 2023).

	Amazon EBS	Portworx	OpenEBS	Rook-Ceph
Random read bandwidth (MiB/s)	241.85	673.65	48.58	193.4
Random write bandwidth (MiB/s)	239.4	230.75	42.675	198.35
Random read IOPS	2961.35	42820	3879.45	6971.9
Random write IOPS	2968.35	4598.5	1496.9	2660.95
Read latency (µsec)	1364.51	263.08	1558.47	1360.70
Write latency (µsec)	1364.79	1643.00	3431.50	6812.00
Sequential read (MiB/s)	246.35	878.45	34.68	577.15
Sequential write (MiB/s)	245.2	254.9	52.635	252.1
Mixed read IOPS	2230.8	15450	2039.95	3585.1
Mixed write IOPS	739.55	5160.15	682.4	1192.9

Table 6.2. Average values of all repetitions and occasions - AWS

6.2.1 Random read and write bandwidth

Portworx shows a considerably higher read speed, compared to the other solutions. Write performance was similar between Amazon EBS, at 241,5 MiB/s, and Portworx, at 230,5 MiB/s. Rook Ceph follows, at 200,5 MiB/s, however with high variation. OpenEBS was the slowest in both random read and write bandwidth.

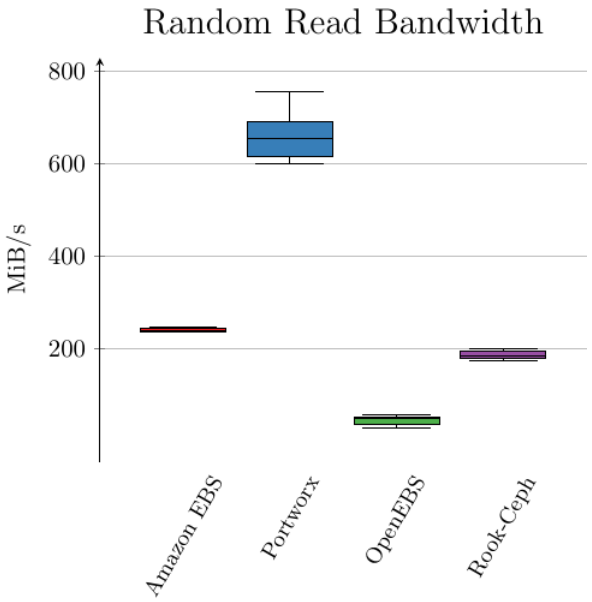

Fig 6.10. Box plot displaying the random read bandwidth in mebibyte per second on AWS.

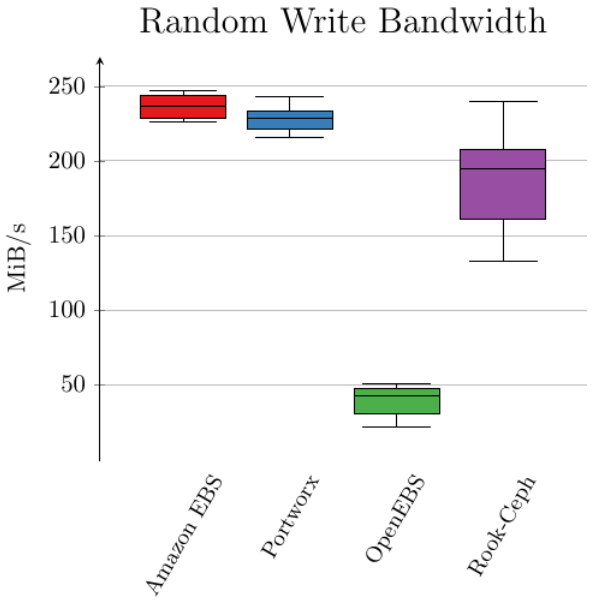

Fig 6.11. Box plot displaying the random write bandwidth in mebibyte per second on AWS.

6.2.2 Random read and write IOPS

Portworx showed a considerably higher random read and write IOPS compared to the other solutions, however with a high coefficient of variability (CV) for read, at 19,45%. Rook Ceph outperformed OpenEBS and Azure Disk Storage in read IOPS, while Azure Disk Storage outperformed OpenEBS and Rook Ceph in write IOPS.

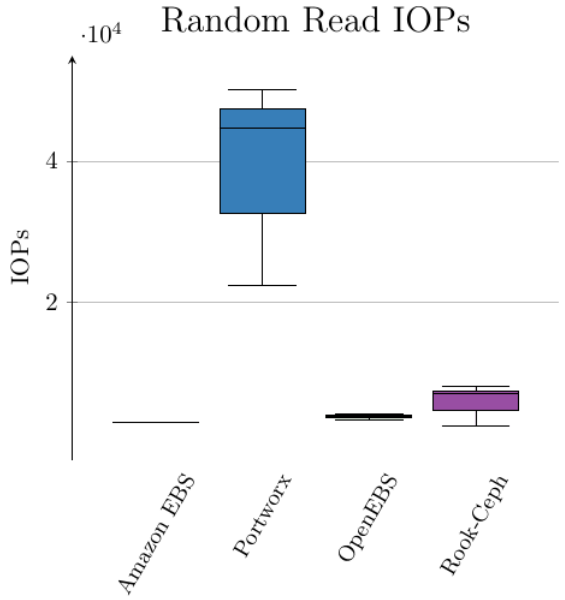

Fig 6.12. Box plot displaying the random read I/O per second on AWS.

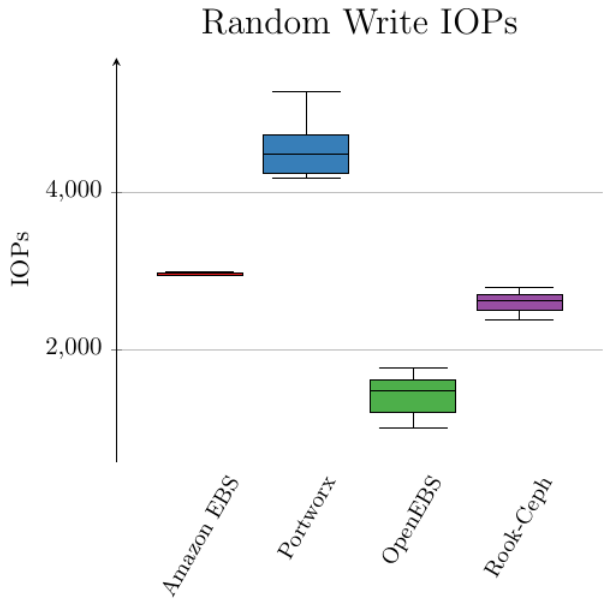

Fig 6.13. Box plot displaying the random write I/O per second on AWS.

6.2.3 Read and write latency

Portworx considerably outperformed the others in read latency, with the other solutions performing comparatively. Amazon EBS and Portworx showed similar write latency followed by OpenEBS, then Rook-Ceph.

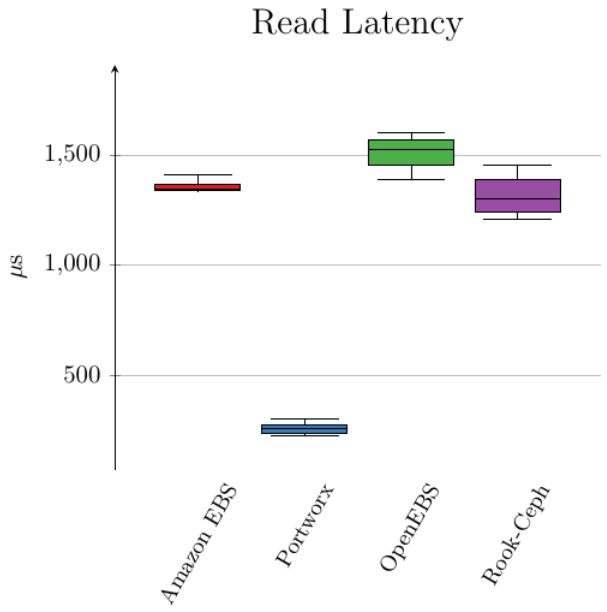

Fig 6.14. Box plot displaying the read latency in microseconds on AWS.

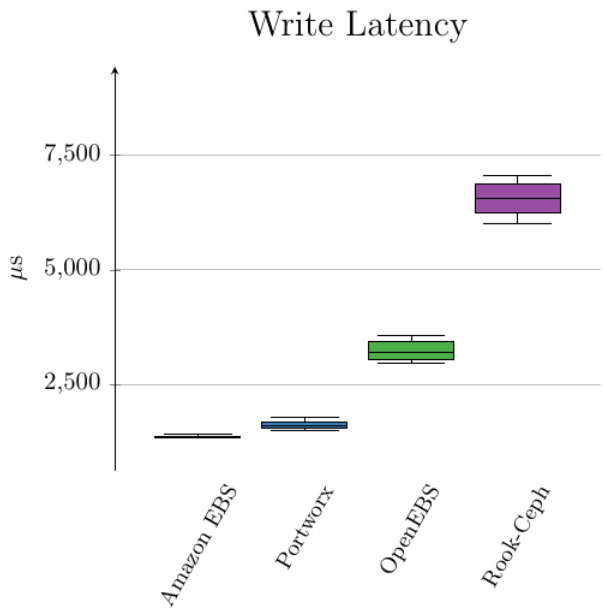

Fig 6.15. Box plot displaying the write latency in microseconds on AWS.

6.2.4 Sequential read and write

Portworx showed a higher sequential read speed, compared to the other solutions. Write performance was similar between Portworx, at 255,5 MiB/s, Rook Ceph, at 252 MiB/s, and Amazon EBS following, at 246 MiB/s. OpenEBS was considerably slower in both random read and write bandwidth.

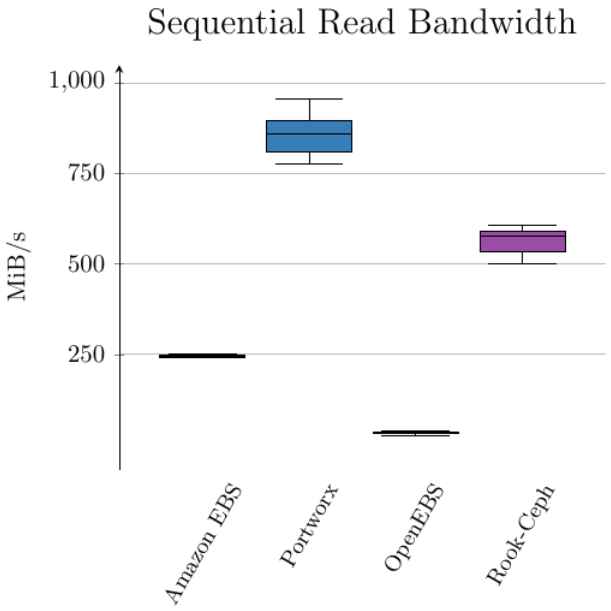

Fig 6.16. Box plot displaying the sequential read bandwidth in mebibyte per second on AWS.

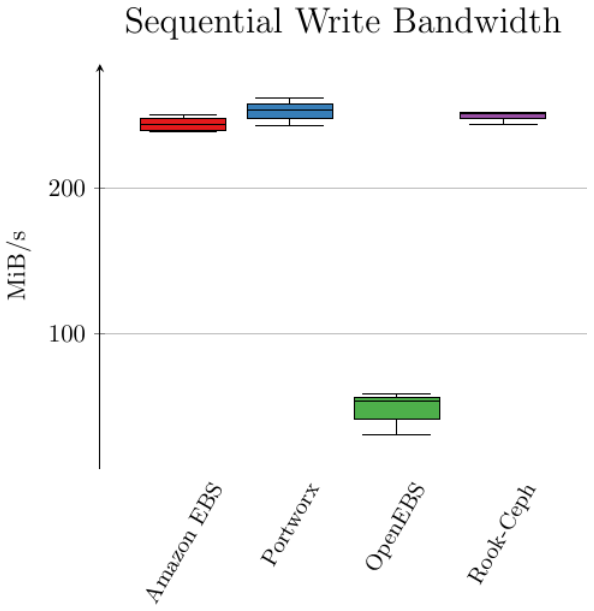

Fig 6.17. Box plot displaying the sequential write bandwidth in mebibyte per second on AWS.

6.2.5 Mixed read and write IOPS

Portworx showed a considerably higher read and write IOPS compared to the other solutions, followed by Rook Ceph. Performance between Amazon EBS and OpenEBS was comparative in both read and write IOPS.

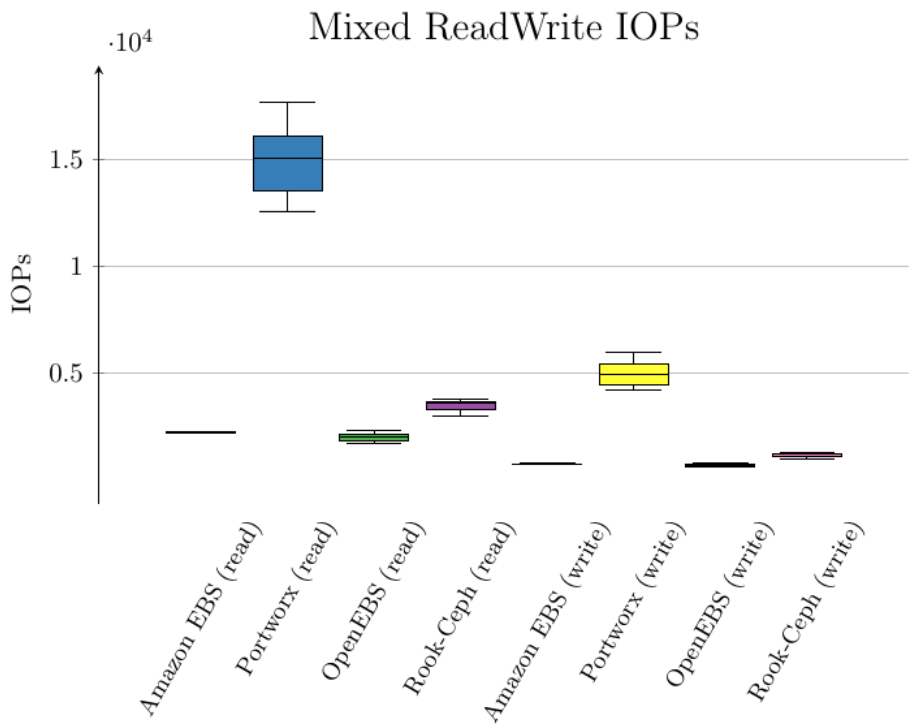

Fig 6.18. Box plot displaying the mixed read and write I/O per second on AWS.

7 Discussion

This chapter will give an analysis of the results, by comparing the solutions, between platforms, and towards the previous results by Mercl & Pavlik. This chapter will also give a discussion on the potential ethical and societal impact as a result of this study.

7.1 Analysis

All the tests were done on four occasions per solution, totaling 20 repetitions. There were however a few tests that showed higher variability, for example random read IOPS for Portworx. The focus of this study, however, is to determine the best performing solution, and for that purpose the results show clear trends, and therefore it was determined additional repetitions were not necessary.

Amazon EBS showed the lowest variance for results between each run and occasion. This may be considered expected behaviour since using the integrated storage solution has the advantage of not introducing overhead, which means less complexity that could impact the results.

Azure Disk Storage showed higher variance compared to Amazon EBS. However the variance was still lower compared to that of Portworx, OpenEBS, and Rook Ceph, reaffirming the advantage of no overhead.

Portworx showed the best read performance on both AWS and Azure by far. On Azure, Portworx showed almost nine times higher read IOPS compared to second best performing, Azure Disk Storage. According to Portworx documentation, FIO tests using the direct option should be respected and therefore not use host cache (Portworx, 2023a). However the results suggest read caching has had a significant impact as it greatly exceeds what should be a max IOPS of 5000 on Azure and 3000 on AWS.

Portworx showed lower random read IOPS on both AWS and Azure at the first run on each occasion, which was a consistent pattern shown on all occasions. This phenomenon was however not reproduced on any other storage solution. This may be, once again, due to read caching in some way impacting the results.

OpenEBS consistently showed the worst performance in terms of bandwidth, and among the worst in IOPS and latency. Rook Ceph did not excel in any category but showed stable performance, outperforming OpenEBS in IOPS and bandwidth, however with worse latency.

The Azure results showed similar patterns as the results from Mercl & Pavlik (2019). Portworx consistently showed the best read bandwidth, latency and IOPS. The other solutions showed similar patterns in performance with slight differences at times. This may be due to a number of different factors such as different versions of solutions and infrastructure. Furthermore we don't know the exact machine and disk type used in their testing and therefore we cannot reproduce a one-to-one comparison.

The results between AWS and Azure cannot be compared one-to-one as they are using different hardware, disk specifications, Kubernetes version, and operating system. However we can analyse the trends and general observations to compare the results between the platforms. For both AWS and Azure, Portworx showed significant read speed, high IOPS and low latency.

One difference in results between Azure and AWS was for sequential write speed, where, on AWS, the average speed between Amazon EBS, at 245.2 MiB/s, Portworx, at 254.9 MiB/s, and Rook Ceph, at 252.1 MiB/s, was similar with only a 3.96% variance between Amazon EBS and Rook Ceph. However on Azure, Azure Disk Storage greatly outperformed the other solutions, at 192.65 MiB/s, followed by Rook at 86.67 MiB/s, and Portworx at 59.55 MiB/s, meaning a variance greater than three times between Portworx and Azure Disk Storage. OpenEBS however, showed lower performance on both platforms, at 55.4 MiB/s on AWS, and 49.3 MiB/s on Azure.

7.2 Ethical Impact

Portworx is a closed source commercial solution and as a result there are potential ethical issues arised. Software and hardware developers have been known to, at times, manipulate benchmarks and tests by artificially inflating the results, with some prominent cases being Samsung lawsuit after getting caught cheating phone benchmarks (McCarthy, 2019), or the multiple accusations of Samsung cheating TV benchmarks (*Samsung Once Again Caught Cheating on Benchmarks*, 2022). Further, Eeckhout (2017) points out that misuse and abuse has happened in usage of benchmarking software and in interpretation of results. Even if the developers do not deliberately cheat a benchmark, the non-transparent nature of closed source software means we do not necessarily know how the technology works, and therefore cannot know how the results reflect real life performance.

During testing, a much higher read performance was discovered for Portworx, compared to the other solutions. We are not aware of the reason for this, but we suspect it may be due to how their caching works. Due to the previously mentioned non-transparency surrounding closed source software it is harder to investigate the exact reason for why this is, it may be due to a number of other different reasons. However, there is a certain lack of trust and ensurance that is missing compared to open source software.

A common ethical issue when doing experiments is how to properly handle human subjects. This study does not involve any human subjects and therefore avoids any such issues.

7.3 Societal Impact

There are two main societal impacts identified for this study; which is how the results can be used for publicity for a cloud provider or storage solution, and how it may be used for saving money, resources and time.

Portworx is a commercial solution and they would have a monetary reason to advertise their product as the best solution available. The high read performance reported in this study could be used for publicity. The results of this study may or may not reflect performance in production environments and could potentially show a misconstrued and biased view. This is a common problem for any type of benchmark involving commercial elements, as there are multiple ways of conducting such an experiment, which could produce a wide variety of results depending on the cases. It is important

from the experimenters side to acknowledge this issue and aim to be as unbiased as possible.

The same issue arises for the cloud providers, while this study does not do a one-to-one comparison between AWS and Azure, the benchmark and its results could still be used for making such a comparison. The results may consequently impact the publicity and image of the company, by showing a bias towards one of the providers.

The other societal impact; saving money, resources and time, is mainly due to how the results from the benchmarks could help for choosing the most performance optimised solution and therefore save in terms of computation resources or time. This study does not provide an analysis into cost saving, resource usage or time saving, and as such only offers a partial derivative from the performance.

8 Conclusion

To answer the research question and decide which storage solution would be 'best' from a performance perspective, we evaluated the performance from three key metrics; bandwidth, latency, and IOPS, in both read and write performance.

Based on the data from the experiments, Portworx was the best performing solution in terms of read performance as it, often considerably, outperformed the other solutions on both AWS and Azure in all categories. However the validity of the read performance for Portworx cannot be determined. Portworx should respect the direct option in FIO and therefore not use host caching, but the results suggest caching made a significant impact.

For write performance, it is harder to determine from the results what the best performing storage solution would be. If we disregard Azure Disk Storage and Amazon EBS, we can conclude that Portworx was the best performing in terms of latency and IOPS, on both AWS and Azure. For bandwidth, Rook Ceph outperformed Portworx and OpenEBS on Azure. However, on AWS, Portworx was the fastest.

Portworx showed the highest write IOPS by far on both Azure and AWS. Portworx also showed the lowest write latency on Azure. On AWS, Portworx had the lowest write latency among the storage solutions, but was slightly beaten by Amazon EBS. On AWS Amazon EBS, Portworx and Rook Ceph tied sequential write speed, Amazon EBS and Portworx tied in random write bandwidth, followed by Rook Ceph.

The results from this study can only be considered representative of performance in a simulated environment. It is important to take into account other aspects than just performance when choosing a storage solution in a production environment. The integrated Azure Disk Storage and Amazon EBS showed high performance in all metrics and has the advantage of not complicating the infrastructure or introducing overhead. Portworx was generally the fastest performing, however it is a commercial solution, Rook with Ceph could be considered a better solution if one wishes to use free and open source software.

The focus in this study is on comparing the storage solutions from a performance perspective, and offering a conclusion for what the 'best' one would be. When selecting a storage solution in a production environment, other criterias may be of importance. Some things to take into account when selecting a storage solution may be pricing, compliance with laws such as GDPR, integration with existing tools, or features such as encryption, backup capabilities, simplicity of design and setup, or snapshotting. Using Azure disk storage or Amazon EBS instead of a distributed storage solution can also be beneficial from a performance perspective due to not requiring any overhead or setup.

8.1 Future work

This report was targeted towards general purpose workloads, however both Azure and AWS offer machine and disk types that are targeted towards other use cases. For example Azure offers their Ls-series machines that are targeted storage optimised

(Microsoft, 2023d). It would be interesting to see how the storage solutions would perform using other types of machines and disk types, and how the results would differ.

This study only compares some of the most popular storage solutions. Cloud computing and Kubernetes are technologies that are being rapidly developed. The results from the benchmark done in this paper may differ if replicated in the future, as newer versions of the storage solutions and Kubernetes infrastructure come out. There are also many more solutions on the market today which would be of interest to also compare in a more substantive benchmark.

Furthermore, the tests were solely done on northern european servers for geographical reasons, and can therefore only be considered applicable for this region. There may be performance fluctuations between different server regions, and the resource usage and availability patterns may also vary. It would be of interest to compare the results between server regions and analyse if and how the results may differ.

Lastly, as this report is based on the work of Mercl & Pavlik (2019), dbench with the default parameter values was used. There are two identified problems with this; firstly, this only partially fulfils principle 2 of Papadopoulos et al. (2021), Workload and configuration coverage. Secondly, by default, dbench runs multiple FIO tests during a 15 second period each, with a ramp time of 2 seconds. For accurate disk benchmarking, testing during a longer time period, with a workload largely exceeding the host memory, should be conducted. A more comprehensive and relevant study should use a larger workload coverage based on realistic probabilistic distributions.

References

Amazon Web Services. (2023a). *Amazon EC2 Instance Types—Amazon Web Services*. Amazon Web Services, Inc. https://aws.amazon.com/ec2/instance-types/

Amazon Web Services. (2023b). *High-Performance Block Storage – Amazon EBS – Amazon Web Services*. Amazon Web Services, Inc. https://aws.amazon.com/ebs/

Amazon Web Services. (2023c). *Kubernetes on AWS*. Amazon Web Services, Inc. https://aws.amazon.com/kubernetes/

Andersson, F. (2023). CLOUD-NATIVE STORAGE SOLUTIONS FOR KUBERNETES : A performance Comparison (Version 1.0) [Data set]. Digitala Vetenskapliga Arkivet. https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-22757

Axboe, J. (2022). *Flexible I/O Tester* [C]. https://github.com/axboe/fio (Original work published 2012)

BLUESHOE. (2021). *Managed vs. Unmanaged Kubernetes*. https://www.blueshoe.io/blog/managed-vs-unmanaged-kubernetes/

Cloudian. (2023). Kubernetes Storage Solutions: Top 4 Solutions & How to Choose. *Cloudian*. https://cloudian.com/guides/kubernetes-storage/kubernetes-storage-solutions-top-4-solutions-how-to-choose/

Eeckhout, L. K. J., Lieven (Ed.). (2006). *Performance Evaluation and Benchmarking*. CRC Press. https://doi.org/10.1201/9781315220505

HashiCorp. (n.d.-a). *Provision an AKS Cluster (Azure) | Terraform | HashiCorp Developer*. Provision an AKS Cluster (Azure) | Terraform | HashiCorp Developer. Retrieved April 12, 2023, from https://developer.hashicorp.com/terraform/tutorials/kubernetes/aks

HashiCorp. (n.d.-b). *Provision an EKS Cluster (AWS) | Terraform | HashiCorp Developer*. Provision an EKS Cluster (AWS) | Terraform | HashiCorp Developer. Retrieved April 12, 2023, from https://developer.hashicorp.com/terraform/tutorials/kubernetes/eks

Heketi. (2023). *Heketi* [Go]. Heketi. https://github.com/heketi/heketi (Original work published 2015)

Kilcioglu, C., Rao, J. M., Kannan, A., & McAfee, R. P. (2017). Usage Patterns and the Economics of the Public Cloud. *Proceedings of the 26th International Conference on World Wide Web*, 83–91. https://doi.org/10.1145/3038912.3052707

Kubernetes. (2022). *Production-Grade Container Orchestration*. https://kubernetes.io/docs

Liu, L. (2023). *Dbench* [Shell]. https://github.com/leeliu/dbench (Original work published 2018)

McCarthy, K. (2019). *Ever own a Galaxy S4? Congrats, you're $10 richer as Samsung agrees payout over dodgy speed tests*. https://www.theregister.com/2019/09/30/samsung_benchmarking_settlement/

Mehndiratta, H. (2020). *Comparing Top Storage Solutions for Kubernetes*. https://kubevious.io/blog/post/comparing-top-storage-solutions-for-kubernetes

Mercl, L., & Pavlik, J. (2019). Public Cloud Kubernetes Storage Performance Analysis.

In N. T. Nguyen, R. Chbeir, E. Exposito, P. Aniorté, & B. Trawiński (Eds.), *Computational Collective Intelligence* (pp. 649–660). Springer International Publishing. https://doi.org/10.1007/978-3-030-28374-2_56
Microsoft. (2023a). *Azure Disk Storage – Block Storage*. https://azure.microsoft.com/en-us/products/storage/disks
Microsoft. (2023b). *Cloud Computing Services*. https://azure.microsoft.com/en-us
Microsoft. (2023c). *Sizes for virtual machines in Azure*. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes
Microsoft. (2023d). *Virtual Machine series*. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/
Moravcik, M., Kontsek, M., Segec, P., & Cymbalak, D. (2022). Kubernetes—Evolution of virtualization. *2022 20th International Conference on Emerging ELearning Technologies and Applications (ICETA)*, 454–459. https://doi.org/10.1109/ICETA57911.2022.9974681
Mutai, J. (2019). *Best Storage Solutions for Kubernetes & Docker Containers | ComputingForGeeks*. https://computingforgeeks.com/storage-solutions-for-kubernetes-and-docker/
OpenEBS. (2021). *Kubernetes storage simplified*. https://openebs.io/
Papadopoulos, A. V., Versluis, L., Bauer, A., Herbst, N., Kistowski, J. von, Ali-Eldin, A., Abad, C. L., Amaral, J. N., Tůma, P., & Iosup, A. (2021). Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering*, *47*(8), 1528–1543. https://doi.org/10.1109/TSE.2019.2927908
Pereira Ferreira, A., & Sinnott, R. (2019). A Performance Evaluation of Containers Running on Managed Kubernetes Services. *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 199–208. https://doi.org/10.1109/CloudCom.2019.00038
Portworx. (2023a). *FIO Performance*. Portworx Documentation. https://docs.portworx.com/operations/operate-other/performance-and-tuning/fio/
Portworx. (2023b). *Persistent Storage for containers & data services for stateful containers*. https://portworx.com/
Red Hat. (2022). *State of Kubernetes security report 2022*.
Rook. (2023). *Open-Source, Cloud-Native Storage for Kubernetes*. https://rook.io/
Samsung once again caught cheating on benchmarks. (2022, June 16). Android Authority. https://www.androidauthority.com/samsung-tv-benchmark-cheating-3177061/
The Ceph Foundation. (2023). *Ceph.io*. https://ceph.io/en/
The Cloud Native Computing Foundation. (2019). *CNCF Survey Report*. https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf
The Cloud Native Computing Foundation. (2021). *CNCF Survey Report*. https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. https://doi.org/10.1007/978-3-642-29044-2

Appendix A - Dbench.yaml

The following is a modified yaml for deploying five test sequences consecutively. The file is based on (Liu, 2018/2023). A different image had to be specified, as the one originally used is deprecated, functionally the image is the same, just uploaded by another user. To use the yaml for deploying tests, 'storageClassName' has to be changed to match the storage class name of the specified storage solution. The file can be applied thereafter.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: dbench-pv-claim
spec:
  storageClassName: <storageClassName>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1000Gi
---
apiVersion: batch/v1
kind: Job
metadata:
  name: dbench
spec:
  parallelism: 1
  completions: 5
  template:
    spec:
      containers:
      - name: dbench
        image: zayashv/dbench:latest
        imagePullPolicy: Always
        env:
          - name: DBENCH_MOUNTPOINT
            value: /data
        volumeMounts:
        - name: dbench-pv
          mountPath: /data
      restartPolicy: Never
      volumes:
      - name: dbench-pv
        persistentVolumeClaim:
          claimName: dbench-pv-claim
  backoffLimit: 4
```

Appendix B - Azure results Disk Storage

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 21 12:38:29 2023

Apr 21 12:38:29 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	166	168	195	178	194
Random write bandwidth (MiB/s)	115	118	127	119	127
Random read IOPS	3683	3682	4078	3945	4076
Random write IOPS	924	923	1014	945	1014
Read latency (µsec)	1108.19	1107.71	980.08	1015.16	977.61
Write latency (µsec)	4764.32	4657.35	3936.94	4275.63	3942.05
Sequential read (MiB/s)	195	194	194	193	194
Sequential write (MiB/s)	192	189	194	190	194
Mixed read IOPS	1506	1539	1774	1646	1777
Mixed write IOPS	509	519	594	546	594

Test occasion #2 - Apr 21 18:54:08 2023

Apr 21 18:54:08 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	178	195	197	191	177
Random write bandwidth (MiB/s)	118	127	125	126	118
Random read IOPS	3932	4071	4072	4063	3927
Random write IOPS	942	1014	991	999	941

Read latency (µsec)	1018.15	982.64	983.31	988.08	1020.3
Write latency (µsec)	4320	3930	4010	4000	4410
Sequential read (MiB/s)	194	194	197	193	195
Sequential write (MiB/s)	189	194	197	192	189
Mixed read IOPS	1633	1766	1798	1743	1629
Mixed write IOPS	549	594	589	587	547

Test occasion #3 - Apr 22 11:24:57 2023

Apr 22 11:24:57 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	193	183	197	197	182
Random write bandwidth (MiB/s)	123	123	127	124	122
Random read IOPS	4063	4028	4062	4079	4019
Random write IOPS	982	977	995	985	965
Read latency (µsec)	987.73	999.99	981.87	981.4	1010.66
Write latency (µsec)	4060	4160	3930	3940	4230
Sequential read (MiB/s)	196	193	194	197	195
Sequential write (MiB/s)	194	191	194	196	192
Mixed read IOPS	1759	1710	1781	1786	1654
Mixed write IOPS	580	565	594	590	559

Test occasion #4 - Apr 23 08:03:06 2023

Apr 23 08:03:06 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	178	195	177	194	191

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	119	127	119	127	123
Random read IOPS	3940	4055	3948	4083	4029
Random write IOPS	944	1014	944	1014	978
Read latency (µsec)	1033.97	981.67	1016.68	986.47	988.78
Write latency (µsec)	4370	3930	4260	3930	4090
Sequential read (MiB/s)	196	195	190	195	196
Sequential write (MiB/s)	193	195	190	194	194
Mixed read IOPS	1639	1760	1656	1786	1718
Mixed write IOPS	548	599	543	593	586

Appendix C - Azure results Portworx

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 21 13:19:39 2023

Apr 21 13:19:39 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	717	757	749	728	795
Random write bandwidth (MiB/s)	57.4	56.1	60.4	56.5	48.2
Random read IOPS	26100	43800	42100	43900	40500
Random write IOPS	6025	5928	6356	6278	5405
Read latency (µsec)	243.57	234.26	242.88	248.23	263.34
Write latency (µsec)	600.3	602.42	613.42	600.89	864.28
Sequential read (MiB/s)	893	896	892	889	850
Sequential write (MiB/s)	56.8	57.3	61.7	58.7	60.7
Mixed read IOPS	20200	20400	21800	20100	15900
Mixed write IOPS	6718	6742	7240	6983	5263

Test occasion #2 - Apr 21 19:27:18 2023

Apr 21 19:27:18 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	533	558	575	561	595
Random write bandwidth (MiB/s)	55.1	52.5	51.4	52.6	55.6
Random read IOPS	20100	28500	27700	27600	29400
Random write IOPS	6052	6155	5899	6002	5946

Read latency (µsec)	386.64	364.91	345.28	350.17	336.87
Write latency (µsec)	1063.68	1031.7	1078.04	1130.61	898.81
Sequential read (MiB/s)	668	662	708	625	649
Sequential write (MiB/s)	54.6	57.7	54.8	54.6	57.6
Mixed read IOPS	14200	13800	13400	13200	15400
Mixed write IOPS	4735	4332	4444	4417	5152

Test occasion #3 - Apr 22 13:25:58 2023

Apr 22 13:25:58 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	687	714	719	701	714
Random write bandwidth (MiB/s)	63.1	60.6	61.6	57.4	60.7
Random read IOPS	25400	42700	42800	42300	42100
Random write IOPS	5819	6362	5931	6039	6197
Read latency (µsec)	204.72	212.36	198.4	204.47	205.83
Write latency (µsec)	637.11	629.2	637.56	629.12	646.75
Sequential read (MiB/s)	875	859	820	859	854
Sequential write (MiB/s)	61.6	62.4	61.7	62.1	63.4
Mixed read IOPS	21800	20800	20600	21900	20900
Mixed write IOPS	7224	6946	6849	7250	6887

Test occasion #4 - Apr 23 10:21:43 2023

Apr 23 10:21:43 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	528	705	680	674	675

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	52.3	62.2	61.3	58.5	58.5
Random read IOPS	24900	41500	42200	39600	41700
Random write IOPS	5535	6001	6093	6124	6434
Read latency (µsec)	266.78	209.37	207.91	214.7	202.83
Write latency (µsec)	874.92	625.57	690.72	630.57	639.41
Sequential read (MiB/s)	791	826	847	832	807
Sequential write (MiB/s)	60.7	58.4	62.3	61.1	62.8
Mixed read IOPS	15900	21800	20700	21500	21700
Mixed write IOPS	5317	7034	6856	7120	7209

Appendix D - Azure results OpenEBS

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 21 17:39:22 2023

Apr 21 17:39:22 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	10.6	16.2	17.9	20.6	18.4
Random write bandwidth (MiB/s)	14.4	42.4	42.7	42.2	40.4
Random read IOPS	1804	2216	2378	2623	2482
Random write IOPS	1064	1431	1698	1507	1603
Read latency (µsec)	2840.59	2706.32	2416.58	2250.64	2496.45
Write latency (µsec)	4187.3	3088.19	2858.34	2478.32	3063.07
Sequential read (MiB/s)	12.3	10.6	10.2	9.731	9.586
Sequential write (MiB/s)	49.9	51.5	50.9	49.9	49.9
Mixed read IOPS	1519	1608	1674	1733	1613
Mixed write IOPS	484	531	570	591	535

Test occasion #2 - Apr 21 22:28:56 2023

Apr 21 22:28:56 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	12.2	17.9	18.4	19.4	19.2
Random write bandwidth (MiB/s)	22.9	42.4	39.9	39.8	39.8
Random read IOPS	1729	2467	2399	2525	2502
Random write IOPS	559	1775	1527	1604	1703

Read latency (µsec)	2820.14	2576.71	2336.48	2268.51	2321.76
Write latency (µsec)	4267.27	2772.86	3100.18	3026.28	2757.66
Sequential read (MiB/s)	7.694	9.791	9.675	9.932	9.306
Sequential write (MiB/s)	22.8	48.5	45.9	41.6	48.7
Mixed read IOPS	1171	1660	1734	1731	1676
Mixed write IOPS	391	565	579	568	556

Test occasion #3 - Apr 22 11:59:36 2023

Apr 22 11:59:36 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	12.8	18.8	19.3	20.9	20.7
Random write bandwidth (MiB/s)	24.5	42.9	44.5	45.8	46.4
Random read IOPS	2011	2338	2612	2639	2623
Random write IOPS	1072	1598	1866	1888	1900
Read latency (µsec)	2872.64	2770.35	2515.1	2327.65	2252.92
Write latency (µsec)	3911.5	3477.41	2652.72	2623.16	2679.03
Sequential read (MiB/s)	10.8	10.6	10.6	9.97	10.4
Sequential write (MiB/s)	53.1	50.2	56.3	53.1	50.7
Mixed read IOPS	1568	1604	1785	1822	1779
Mixed write IOPS	525	547	599	614	596

Test occasion #4 - Apr 23 11:19:24 2023

Apr 23 11:19:24 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	18.3	18.6	18.3	19.9	19.2

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	39.2	39.5	39.3	38.5	37.5
Random read IOPS	2343	2525	2496	2572	2514
Random write IOPS	1482	1706	1536	1492	1740
Read latency (µsec)	2810.09	2557	2529.9	2439.8	2459.25
Write latency (µsec)	3106.06	2582.69	2784.98	2525.06	2556.41
Sequential read (MiB/s)	9.83	9.13	9.268	9.959	10.4
Sequential write (MiB/s)	41.6	45.8	46.6	42.3	41.7
Mixed read IOPS	1532	1704	1697	1716	1716
Mixed write IOPS	503	569	569	569	565

Appendix E - Azure results Rook-Ceph

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 21 15:16:35 2023

Apr 21 15:16:35 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	234	228	234	231	222
Random write bandwidth (MiB/s)	91.5	91.6	88.8	87.9	92.4
Random read IOPS	3478	3395	3539	3235	3077
Random write IOPS	3181	2853	2700	2785	2710
Read latency (µsec)	3139.08	3074.7	3144.74	3095.53	3122.31
Write latency (µsec)	6850	6780	7170	6690	6790
Sequential read (MiB/s)	225	233	218	219	211
Sequential write (MiB/s)	88.9	89.1	87.1	90.7	89.3
Mixed read IOPS	2882	2994	2571	2605	2954
Mixed write IOPS	966	991	860	859	996

Test occasion #2 - Apr 21 21:25:54 2023

Apr 21 21:25:54 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	234	227	233	234	223
Random write bandwidth (MiB/s)	90.6	91.7	89.3	90.2	91.4
Random read IOPS	3345	3315	3598	3262	3204
Random write IOPS	2827	2483	2452	2536	2518

Read latency (µsec)	3207.79	3343.65	3157.59	3264.48	3243.67
Write latency (µsec)	7440	8070	7320	7100	6870
Sequential read (MiB/s)	226	231	219	222	212
Sequential write (MiB/s)	90.5	91.1	87.2	92.6	90.1
Mixed read IOPS	2690	2974	2448	2610	2853
Mixed write IOPS	908	993	822	884	958

Test occasion #3 - Apr 22 16:27:33 2023

Apr 22 16:27:33 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	341	332	349	331	330
Random write bandwidth (MiB/s)	74.7	73.9	74.2	75.6	73.2
Random read IOPS	3500	3313	3574	3179	3451
Random write IOPS	3034	2779	2777	2603	2735
Read latency (µsec)	3072.51	2997.94	2983.48	2962.57	2981.72
Write latency (µsec)	7230	7150	7350	7040	7040
Sequential read (MiB/s)	387	384	392	382	370
Sequential write (MiB/s)	77.1	76.5	75.4	73.7	76.3
Mixed read IOPS	2904	3035	2515	2705	2848
Mixed write IOPS	956	1004	843	905	953

Test occasion #4 - Apr 23 09:26:41 2023

Apr 23 09:26:41 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	234	227	234	232	221

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	91.4	91.2	87.1	89.6	92.2
Random read IOPS	3477	3145	3585	3253	3362
Random write IOPS	3248	2950	2903	2822	2760
Read latency (µsec)	3108.72	3229.86	3065.68	3133.38	3109.26
Write latency (µsec)	6830	6910	7060	7040	6790
Sequential read (MiB/s)	226	232	219	219	210
Sequential write (MiB/s)	92.4	92.7	94.2	89.9	88.6
Mixed read IOPS	2860	3075	2535	2667	2945
Mixed write IOPS	943	1020	837	890	991

Appendix F - AWS results Disk Storage

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 18 13:51:03 2023

Apr 18 13:51:03 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	240	246	237	237	236
Random write bandwidth (MiB/s)	240	246	232	231	233
Random read IOPS	2956	2977	2943	2941	2938
Random write IOPS	2959	2979	2944	2959	2960
Read latency (µsec)	1358.02	1341.97	1388.89	1398.68	1381.83
Write latency (µsec)	1358.48	1342.11	1389.33	1398.45	1383.06
Sequential read (MiB/s)	246	250	242	243	243
Sequential write (MiB/s)	245	249	241	240	241
Mixed read IOPS	2225	2238	2229	2217	2212
Mixed write IOPS	741	743	728	740	748

Test occasion #2 - Apr 19 11:22:58 2023

Apr 19 11:22:58 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	237	243	237	245	242
Random write bandwidth (MiB/s)	231	242	226	245	241
Random read IOPS	2944	2967	2940	2974	2963
Random write IOPS	2943	2970	2938	2976	2969

Read latency (µsec)	1388.71	1352.03	1408.39	1342.65	1353.1
Write latency (µsec)	1389.01	1352.02	1409.36	1342.99	1353.31
Sequential read (MiB/s)	243	247	243	249	246
Sequential write (MiB/s)	240	246	240	248	246
Mixed read IOPS	2218	2236	2226	2243	2238
Mixed write IOPS	741	735	729	735	733

Test occasion #3 - Apr 20 06:03:49 2023

Apr 20 06:03:49 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	238	245	246	247	247
Random write bandwidth (MiB/s)	239	245	246	247	247
Random read IOPS	2950	2973	2976	2980	2980
Random write IOPS	2969	2979	2986	2985	2986
Read latency (µsec)	1372.17	1345.51	1343.35	1340.42	1340.7
Write latency (µsec)	1372.7	1345.76	1343.36	1340.45	1340.49
Sequential read (MiB/s)	244	248	249	250	250
Sequential write (MiB/s)	242	248	249	250	250
Mixed read IOPS	2215	2235	2233	2236	2246
Mixed write IOPS	747	742	747	747	737

Test occasion #4 - Apr 22 12:49:33 2023

Apr 22 12:49:33 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	237	247	247	246	237

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	228	247	247	246	229
Random read IOPS	2940	2980	2983	2979	2943
Random write IOPS	2971	2984	2985	2979	2946
Read latency (µsec)	1409.71	1340.78	1339.11	1341.23	1402.87
Write latency (µsec)	1410.2	1340.75	1338.75	1341.22	1403.98
Sequential read (MiB/s)	243	250	250	249	242
Sequential write (MiB/s)	239	250	250	249	241
Mixed read IOPS	2214	2254	2230	2245	2226
Mixed write IOPS	742	728	755	736	737

Appendix G - AWS results Portworx

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 18 15:53:56 2023

Apr 18 15:53:56 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	672	683	733	751	731
Random write bandwidth (MiB/s)	229	237	239	216	218
Random read IOPS	26700	48400	49300	45600	48100
Random write IOPS	4624	4798	4663	4188	4192
Read latency (µsec)	245.7	246.78	251.49	264.11	267.3
Write latency (µsec)	1530	1590	1590	1730	1730
Sequential read (MiB/s)	946	914	956	892	930
Sequential write (MiB/s)	260	259	247	254	243
Mixed read IOPS	16400	15100	15400	14700	13500
Mixed write IOPS	5423	5291	5097	4689	4468

Test occasion #2 - Apr 19 13:28:21 2023

Apr 19 13:28:21 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	616	706	710	644	686
Random write bandwidth (MiB/s)	228	230	243	242	241
Random read IOPS	26600	45900	48700	48500	47900
Random write IOPS	4307	4394	4544	5277	4725

Read latency (µsec)	272.91	223.86	245.28	246.64	246.12
Write latency (µsec)	1740	1770	1550	1500	1570
Sequential read (MiB/s)	842	859	776	876	879
Sequential write (MiB/s)	248	251	262	261	260
Mixed read IOPS	12700	13700	15700	15600	17100
Mixed write IOPS	4204	4558	5206	5189	5952

Test occasion #3 - Apr 20 10:27:32 2023

Apr 20 10:27:32 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	663	646	689	756	681
Random write bandwidth (MiB/s)	233	235	236	231	235
Random read IOPS	27700	50200	47100	49700	49700
Random write IOPS	4288	4858	4807	4175	4760
Read latency (µsec)	267.76	265.62	260.01	254.17	277.75
Write latency (µsec)	1740	1630	1650	1750	1690
Sequential read (MiB/s)	915	913	895	935	868
Sequential write (MiB/s)	258	257	254	253	250
Mixed read IOPS	15200	16800	17200	13900	16800
Mixed write IOPS	5054	5595	5693	4601	5575

Test occasion #4 - Apr 22 13:16:00 2023

Apr 22 13:16:00 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	615	633	627	632	599

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	216	229	227	226	224
Random read IOPS	22300	44200	43100	43600	43100
Random write IOPS	4235	4643	4851	4664	4977
Read latency (µsec)	303.01	276.71	281.19	281.39	283.8
Write latency (µsec)	1760	1600	1570	1580	1590
Sequential read (MiB/s)	809	845	842	837	840
Sequential write (MiB/s)	257	257	260	254	253
Mixed read IOPS	12600	16300	16500	16100	17700
Mixed write IOPS	4207	5380	5495	5628	5898

Appendix H - AWS results OpenEBS

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 18 17:10:31 2023

Apr 18 17:10:31 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	50.2	51.9	53.4	54.2	54.2
Random write bandwidth (MiB/s)	47.6	39.3	49.4	45.3	47.9
Random read IOPS	3937	3615	3880	3906	3758
Random write IOPS	1676	1328	1487	1699	1293
Read latency (µsec)	1407.62	1573.74	1449.54	1392.09	1531.05
Write latency (µsec)	3130	3600	3410	3060	3690
Sequential read (MiB/s)	35.6	37.4	35.9	37.5	35.6
Sequential write (MiB/s)	55.3	55.4	55.4	56.5	54.3
Mixed read IOPS	2177	1962	2130	2287	1958
Mixed write IOPS	732	639	706	772	644

Test occasion #2 - Apr 19 10:12:30 2023

Apr 19 10:12:30 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	30.3	41.1	50.5	53.2	55.8
Random write bandwidth (MiB/s)	26.6	34.1	47.5	46.9	46.9
Random read IOPS	3337	3490	3950	4130	3885
Random write IOPS	1126	998	1433	1637	1573

Read latency (µsec)	1512.46	1757.63	1663.57	1583.61	1613.86
Write latency (µsec)	4590	3560	3100	3110	3220
Sequential read (MiB/s)	24.8	35.4	37.2	37.9	35.8
Sequential write (MiB/s)	30.7	55.6	54.6	51.6	56.6
Mixed read IOPS	1719	1911	2026	2009	1975
Mixed write IOPS	584	638	680	674	671

Test occasion #3 - Apr 20 14:34:35 2023

Apr 20 14:34:35 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	27.6	45.1	51.2	53.7	52.2
Random write bandwidth (MiB/s)	21.5	24.8	50.4	47.1	49.1
Random read IOPS	3751	3931	4141	4162	4065
Random write IOPS	1108	1760	1722	1576	1669
Read latency (µsec)	1523.86	1609.53	1547.88	1520.91	1524.11
Write latency (µsec)	4430	3530	3020	2950	3310
Sequential read (MiB/s)	26.4	35.7	36.4	26.9	35.5
Sequential write (MiB/s)	32.9	49.8	57.9	56.9	51.7
Mixed read IOPS	1928	2096	2146	2220	2082
Mixed write IOPS	628	711	702	750	697

Test occasion #4 - Apr 21 13:48:24 2023

Apr 21 13:48:24 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	41.8	46.9	50.5	54.7	53.1

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	47.2	37.7	46.2	48.8	49.2
Random read IOPS	3694	3911	3894	4092	4060
Random write IOPS	1424	1760	1469	1690	1510
Read latency (µsec)	1747.26	1590.95	1599.97	1521.51	1498.34
Write latency (µsec)	3100	3310	3750	3300	3460
Sequential read (MiB/s)	35.8	36.9	36.1	35.6	35.2
Sequential write (MiB/s)	55.4	56.5	50.8	56.2	58.6
Mixed read IOPS	2067	2151	1785	2075	2095
Mixed write IOPS	687	728	604	699	702

Appendix I - AWS results Rook-Ceph

Time displayed is the start of the first run, following runs are started sequentially after completion of previous one. All times are in UTC.

Test occasion #1 - Apr 18 14:34:18 2023

Apr 18 14:34:18 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	186	203	195	179	211
Random write bandwidth (MiB/s)	198	211	208	175	195
Random read IOPS	6876	7320	7224	6847	7612
Random write IOPS	2666	2587	2734	2789	2581
Read latency (µsec)	1458.5	1400.58	1375.99	1457.3	1348.19
Write latency (µsec)	7300	7120	7030	7340	6630
Sequential read (MiB/s)	604	554	582	591	607
Sequential write (MiB/s)	255	252	252	252	259
Mixed read IOPS	3701	3537	3657	3797	3546
Mixed write IOPS	1230	1181	1206	1279	1177

Test occasion #2 - Apr 19 12:10:03 2023

Apr 19 12:10:03 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	186	196	186	181	244
Random write bandwidth (MiB/s)	190	183	199	140	133
Random read IOPS	6857	7331	7212	2360	8029
Random write IOPS	2633	2628	2778	2726	2379

Read latency (µsec)	1528.44	1404.48	1473.92	1638.77	1479.96
Write latency (µsec)	8700	7300	6880	7000	6550
Sequential read (MiB/s)	595	536	500	587	602
Sequential write (MiB/s)	258	252	244	251	252
Mixed read IOPS	3601	3458	2976	3631	3561
Mixed write IOPS	1196	1137	992	1224	1178

Test occasion #3 - Apr 20 06:32:07 2023

Apr 20 06:32:07 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read bandwidth (MiB/s)	186	195	183	176	206
Random write bandwidth (MiB/s)	227	240	234	224	229
Random read IOPS	6811	7350	7212	6870	7648
Random write IOPS	2636	2644	2760	2755	2585
Read latency (µsec)	1280.23	1254.15	1251.23	1209.21	1212.54
Write latency (µsec)	6420	6170	6780	6460	6020
Sequential read (MiB/s)	590	540	572	581	593
Sequential write (MiB/s)	252	251	251	250	253
Mixed read IOPS	3676	3552	3676	3619	3558
Mixed write IOPS	1236	1168	1214	1209	1178

Test occasion #4 - Apr 22 15:14:14 2023

Apr 22 15:14:14 2023	Run #1	Run #2	Run #3	Run #4	Run #5
Random read	188	199	184	174	210

bandwidth (MiB/s)					
Random write bandwidth (MiB/s)	209	207	193	170	202
Random read IOPS	6843	7298	7254	6851	7633
Random write IOPS	2620	2615	2794	2752	2557
Read latency (µsec)	1299.13	1305.51	1287.19	1281.91	1266.69
Write latency (µsec)	6460	6420	6680	6690	6290
Sequential read (MiB/s)	599	552	580	583	595
Sequential write (MiB/s)	252	252	251	252	251
Mixed read IOPS	3690	3521	3650	3745	3550
Mixed write IOPS	1233	1180	1215	1254	1171