Matching distributed file systems with application workloads

Stefan Meyer
MSC CS

Thesis submitted for the degree of Doctor of Philosophy

NATIONAL UNIVERSITY OF IRELAND, CORK

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BOOLE CENTRE FOR RESEARCH IN INFORMATICS

9 September 2017

Head of Department:	Prof. Cormac J. Sreenan
Supervisor:	Prof. John P. Morrison

Research supported by IRCSET and Intel Ireland Ltd.

Contents

List of Figures

List of Tables

I, Stefan Meyer, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Stefan Meyer

Abstract

Modern storage systems have a large number of configurable parameters, distributed over many layers of abstraction. The number of combinations of these parameters, that can be altered to create an instance of such a system, is enormous. In practise, many of these parameters are never altered; instead default values, intended to support generic workloads and access patterns, are used. As systems become larger and evolve to support different workloads, the appropriateness of using default parameters in this way comes into question. This thesis examines the implications of changing some of these parameters and explores the effects these changes have on performance. As part of that work multiple contributions have been made, including the creation of a structured method to create and evaluate different storage configurations, choosing appropriate access sizes for the evaluation, picking representative cloud workloads and capturing storage traces for further analysis, extraction of the workload storage characteristics, creating logical partitions of the distributed file system used for the optimization, the creation of heterogeneous storage pools within the homogeneous system and the mapping and evaluation of the chosen workloads to the examined configurations.

Acknowledgements

Firstly, I would like to thank Prof. John Patrick Morrison to accept my application of the project and his support and guidance during the entire duration of the thesis.

Furthermore I would particularly like to thank Mr. Brian Clayton for his support and technical guidance throughout the duration of this dissertation.

A special thanks I would like to address to Dr. Ruairi O'Reilly for being a good friend and companion during our time in the CUC and afterwards.

I would like to thank Dr. David O'Shea for his friendship and his mathematical advice.

A special thanks I would like to address to Dr. Dapeng Dong for his friendship and advice during our Master and PhD studies.

I would like to also thank the staff and members of The Centre for Unified Computing at the University College Cork for their support for the period of the project.

I would like to thank the examiners Prof. George A. Gravvanis and Dr. John Herbert for reviewing my dissertation and providing constructive comments to improve my dissertation.

I would like to thank Intel Ireland Ltd. and in particular Mr. Joe Buttler, Dr. Giovani Octavio Gomez Estrada and Mr. Kieran Mulqueen for making this project possible and their support throughout the dissertation.

Similarly, I would like to thank the Irish Research Council, who, in combination with Intel Ireland Ltd., made this research possible by supporting it under the Enterprise Partnership Grant EPSPG/2012/480.

A very special thank you is addressed to my parents Susanne and Thomas and my sister Lisa for their continuous support during my studies.

I would like to give a lot of thanks to Dr. Huanhuan Xiong for her support, help and guidance she provided.

Chapter 1

Introduction

In a general purpose cloud system efficiencies are yet to be had from supporting diverse applications and their requirements within a storage system used for a cloud system. Supporting such diverse requirements poses a significant challenge in a storage system that supports fine grained configuration of a variety of parameters.

Currently, there are many different open source cloud management platforms available that can be deployed on premise to be used as a private cloud, such as OpenStack, Apache CloudStack, Eucalyptus and many more. Among these, OpenStack is the most popular and is widely supported by the industry and the community to drive development and innovation.

The workloads that are deployed to cloud systems are very diverse and differ in their storage requirements. While one workload might use mostly small sequential read accesses, another might use large sequential writes, small random read accesses or a combination of multiple access sizes and patterns simultaneously. These different workload storage requirements are difficult to capture with a single storage system configuration.

The storage systems currently used in cloud systems have to support a multitude of storage services within that system. Within OpenStack, three distinct storage systems with different characteristics exist. The image service serves virtual machine images to the compute hosts. The block storage service is used to provide block devices to virtual machines as a means of persistent storage that can be used by standard file system commands and tools. The object storage service is used to provide data storage with a RESTful API. These three storage services have different requirements and can be provided by dedicated storage systems for each service or by a shared system.

Using a dedicated system for each storage service is a costly approach, since multiple systems have to be purchased, configured and maintained. Furthermore, changes to the capacity or adaptation to new workloads is difficult and often requires hardware changes. Using a shared storage system for the storage services can reduce the overhead of maintaining three different systems. The single system would be under one administrative domain and could thus create logical partitions for each service. Furthermore, the larger scale of a shared storage system could result in increased performance since data would be distributed across a larger number of storage devices, speeding up data transfers. Creating a configuration within that system that best supports a given application, is a difficult task, since configuration parameters are often not linked to their respective impact on storage and subsequent workload performance.

In this work, the tuning of a distributed file system for cloud workloads is attempted, as depicted in Figure 1.1. The cloud system used in this work is OpenStack. A detailed description of the components and relationships within OpenStack are presented in Section 1.1. OpenStack supports many storage systems to serve as the backend for its storage services, of which a collection is presented here. The storage system used in this work is the open source distributed file system Ceph, which is highly complex and offers many degrees of freedom for optimization. A detailed description of Ceph and its components is presented in Section 1.2. The integration of OpenStack and Ceph as the storage backend for the different OpenStack storage services is described in Section 1.3. Related work and state of the art is described in Section 1.4.

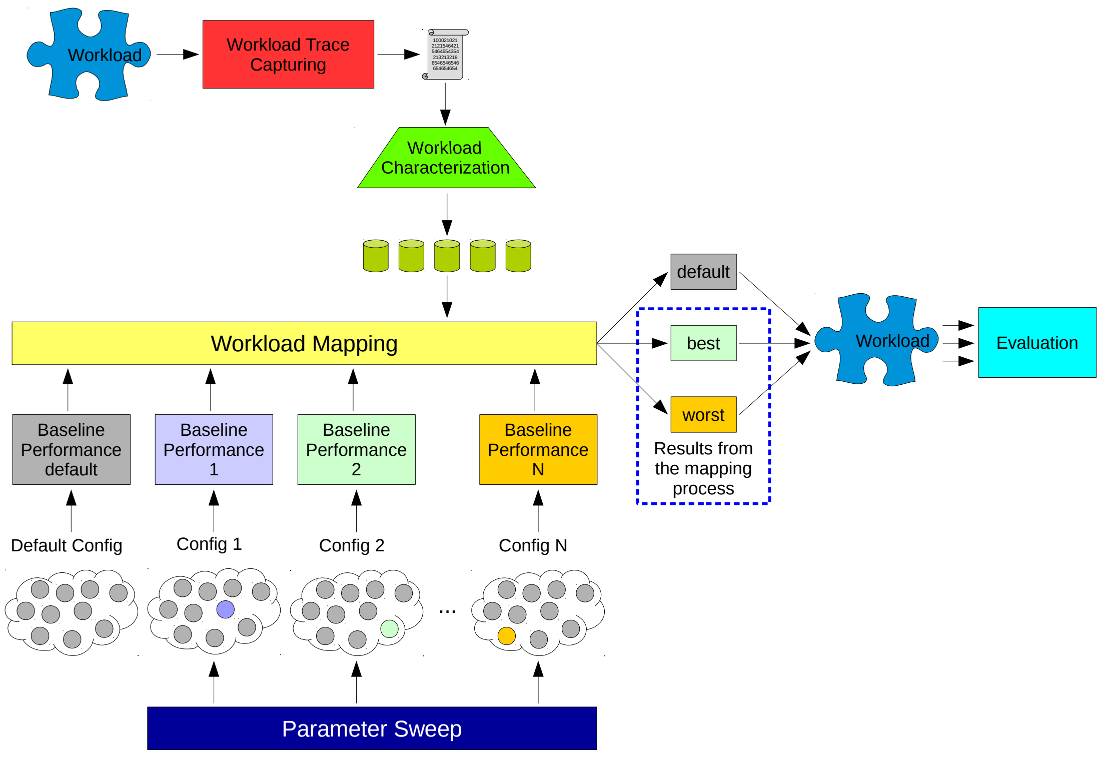


Figure 1.1: Overview of the proposed improvement process.

The improvement process proposed here will be presented in Chapter 2. It presents a structured procedure to construct alternative storage configurations, as opposed to the *ad hoc* tuning process used in the literature. A parameter sweep is used to determine the effect of certain configuration changes, thus resulting in the creation of N configurations and associated baseline performances. The baseline evaluation of these configurations is determined for specific access sizes and patterns. Workload characterization and subsequent mapping of identified characteristics to appropriate storage configurations is made and the appropriateness of the mapping process is validated. In Section 2.3, cloud use cases and appropriate representative workloads are presented. To facilitate improvements the Ceph system is broken into a number of logical partitions, as presented in Section 2.4.

In Chapter 3, the empirical studies are described. They include the creation of a testbed using different server types, storage devices and creation of a system architecture that considers the environment, available hardware and requirements of the deployed systems. In Section 3.2, the system initialization, composed of the testing system and the initialization of virtual machines, used in the empirical experiments, is described. In Section 3.3, cluster configurations are created and in the subsequent section tested against 20 access size and access pattern combinations.

In Chapter 4, the workload characterization process is presented. A storage tracing tool is described in detail in Section 4.2; and in Section 4.3, five representative cloud workloads are subsequently traced and analysed for their respective access sizes and patterns.

In Chapter 5 the information of the workload characterization is used to create a mapping between the access sizes and patterns of the workloads to alternative storage configurations. The predicted best and worst alternative configurations are subsequently empirically evaluated and compared against the default configuration to determine the performance improvements and disimprovements of those configurations and the accuracy of the mapping process.

The conclusions and future work are presented in Chapter 6.

1.1 OpenStack

OpenStack is an open source cloud management platform that can be used to create an Infrastructure-as-a-Service (IaaS) cloud system. It is developed by the community, consisting of independent developers and large companies, such as Intel, Dell and RedHat. OpenStack is capable of scaling from small scale deployments to large deployments spanning thousands of hosts. The capabilities of OpenStack meet the requirements of both private and public cloud providers.

OpenStack is used by many large multinationals, such as Volkswagen, Walmart and Bloomberg. It is also used by public institutions, such as Her Majesty's Revenue and Customs (HMRC) and Postal Savings Bank of China. It is also wide used in research institutions, such as CERN (over 190000 CPU cores), Harvard University and the University of Cambridge.

OpenStack consists of many different interrelated components, each providing an open APIs, many of which can be extended to allow for innovation but keeping compatibility by building on top of the core API calls. All OpenStack components can be managed through the OpenStack Horizon web interface, command line tools and SDKs that support the APIs.

OpenStack is composed of 18 individual components. Each of these components provides specific capabilities. The most often used components of OpenStack are Nova, Keystone, Glance, Neutron, Horizon and Cinder [1]. These components, together with OpenStack Swift, are the core components of OpenStack. The logical relationships are depicted in Figure 1.2.

While OpenStack is the most widely used open source cloud management platform, other systems, such as Apache CloudStack, Eucalyptus and oVirt, exist and offer similar components and functionalities.

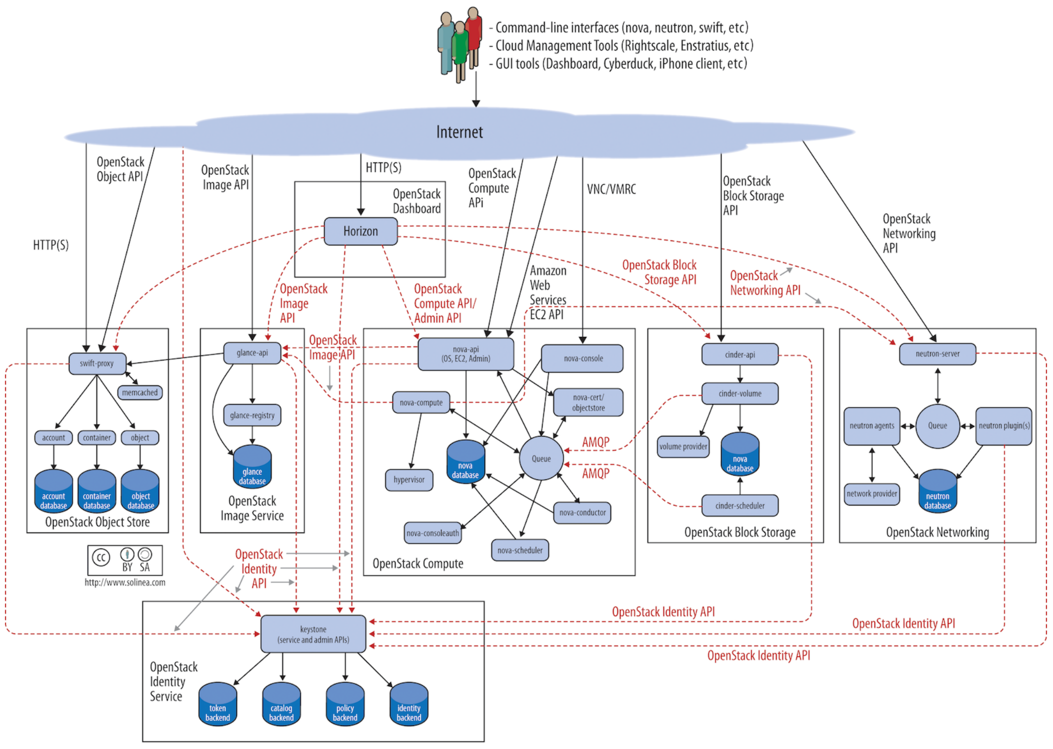


Figure 1.2: Logical architecture of the core OpenStack components.

1.2 Ceph

Ceph [2] is an open source distributed file system. It has been designed to support peta-scale storage systems. Such systems are typically grown incrementally and due to

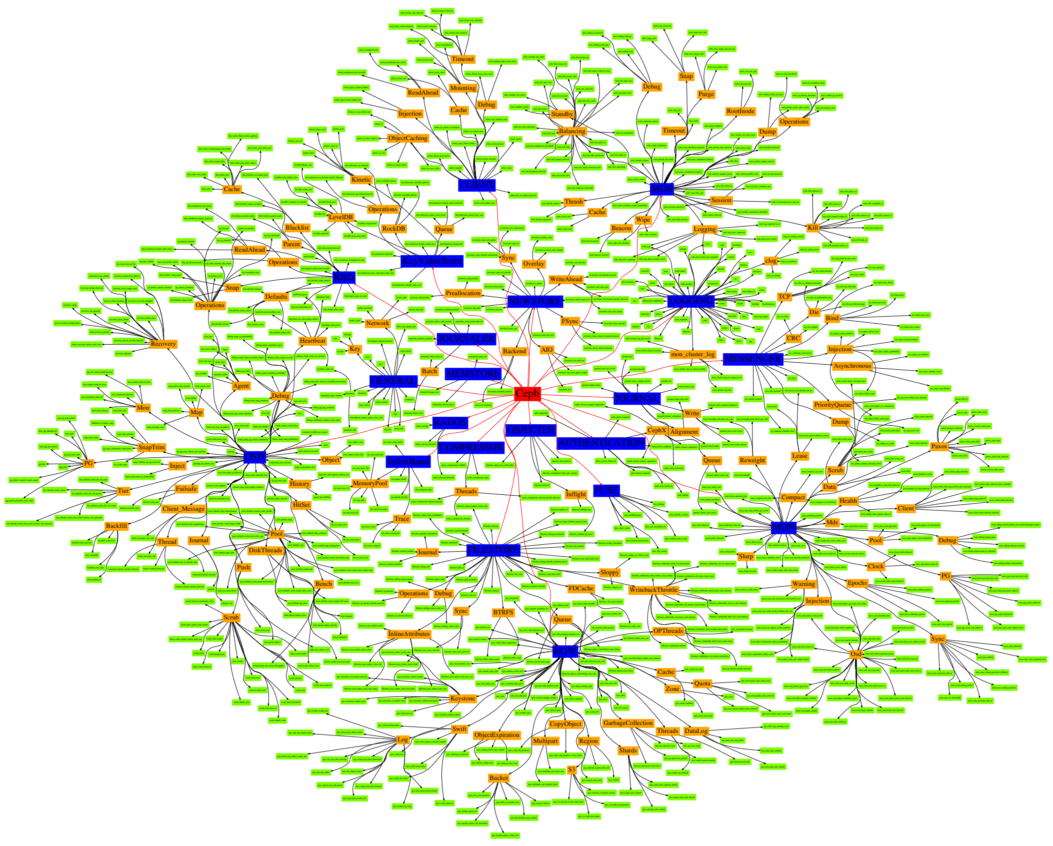


Figure 1.3: Ceph parameter space with **Orange** representing categories and **green** individual parameters.

the large number of components in such systems, the mean time to component failure is short. Moreover, in such systems workloads and workload characteristics constantly change. At the same time, the storage system has to be able to handle thousands of user requests and deliver high throughput [3]. The system replaces the traditional interface to disks or RAID arrays with object storage devices (OSD) that integrate intelligence to handle specific operations locally. Depending on the access interface being used, clients interact directly with the ODSs or with the OSDs in combination with the metadata server to perform operations, such as open and rename. The algorithm used to spread the data over the available OSDs is called CRUSH [4]. It uses placement groups, that are distributed across the available OSDs, to calculate the location of an object to achieve random placement. From a high level, Ceph clients and metadata servers view the object storage cluster, that consists of possibly tens or hundreds of thousands of OSDs, as a single logical object store and namespace. Ceph's Reliable Autonomic Distributed Object Store (RADOS) [5] achieves linear scaling in both capacity and aggregate performance by delegating management of object replication, cluster expansion, failure detection and recovery to OSDs in a distributed fashion. The data objects are stored in logical partitions of the RADOS cluster called pools.

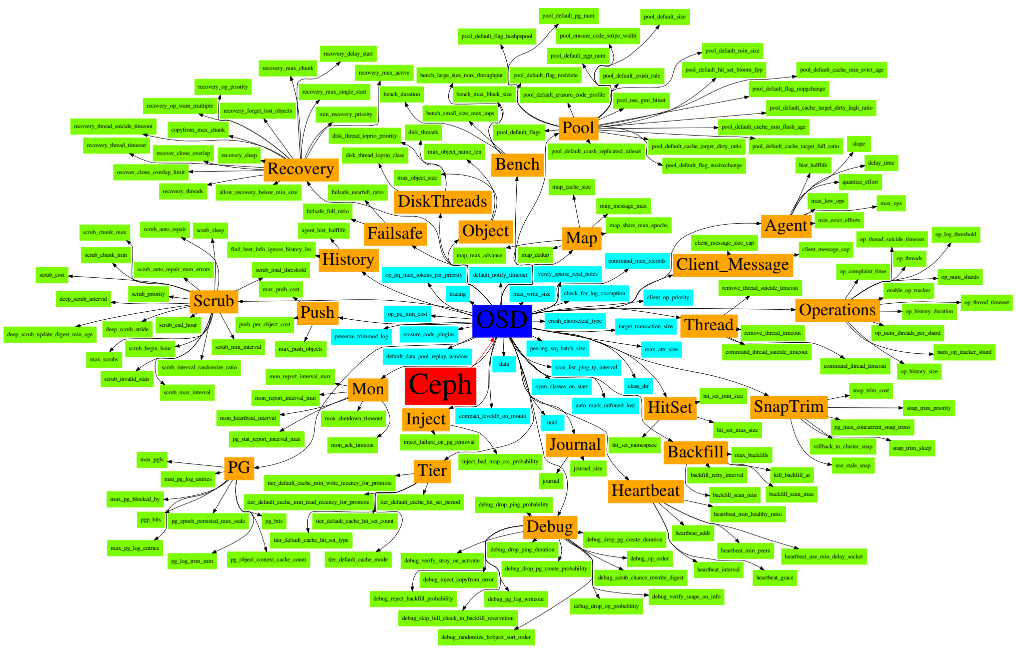


Figure 1.4: Subset of total Ceph parameter space showing only parameters related to the OSD configuration. **Orange** representing categories, **green** their leafs and **cyan** parameters that cannot be grouped.

Ceph is highly configurable. All Ceph components can be configured through the Ceph configuration, that consists of over 800 parameters (see Figure 1.3). Finding an optimal configuration for specific workloads is a difficult task, since the impacts on performance of individual parameters are not documented. When limiting the scope to a single Ceph component, the configuration space shrinks, but even this reduced space might still consist of up to 200 configuration parameters, as depicted in Figure 1.4.

1.2.1 Ceph Storage Architecture

A Ceph storage cluster is composed of several software components, each fulfilling a specific role within the system to provide unique functionalities. The software components are split into distinct storage daemons that can be distributed and do not have to reside within the same host.

From a logical perspective, the object store, RADOS, is the foundation of a Ceph storage cluster. It provides, among others, the distributed object store, high availability, reliability, no single point of failure, self-healing and self-management. Thus, it is the heart of Ceph and holds special importance within the system. The different access interfaces, shown in Figure 1.5, all operate on top of the RADOS layer.

`librados` is a library to access the storage cluster directly using the programming

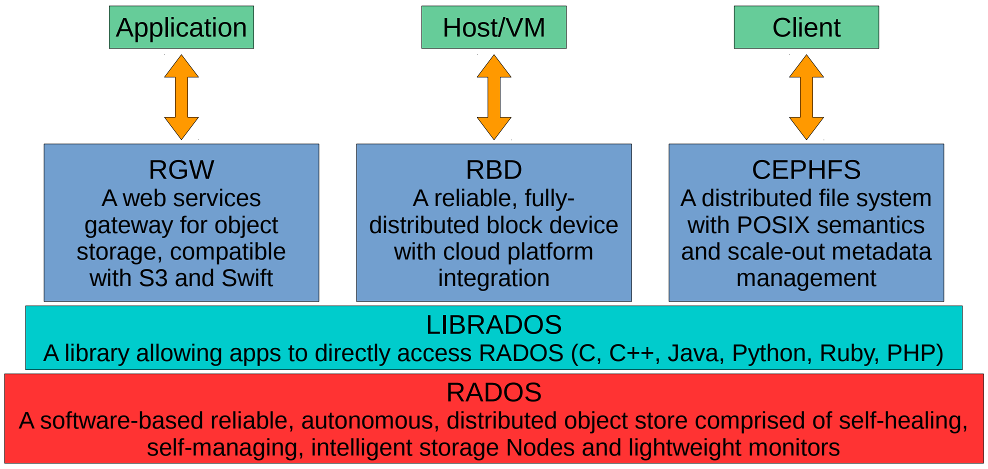

Application
Host/VM
Client
RGW
A web services gateway for object storage, compatible with S3 and Swift
RBD
A reliable, fully-distributed block device with cloud platform integration
CEPHFS
A distributed file system with POSIX semantics and scale-out metadata management
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)
RADOS
A software-based reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage Nodes and lightweight monitors

Figure 1.5: The different interfaces of Ceph [6].

languages Ruby, Java, PHP, Python, C and C++. It provides a native interface to the Ceph storage cluster and is used by other services of Ceph, such as RBD, RGW and CephFS. Furthermore, `librados` gives direct access to RADOS to create custom interfaces to the storage cluster.

The RADOS block device (RBD) interface provides block storage access to the storage cluster. It can be used to directly mount a block device on a client, using the KRBD Linux Kernel implementation, or to provide RADOS block devices as block devices for VMs in a hypervisor, such as QEMU/KVM, using the `librbd` implementation (see Figure 1.6).

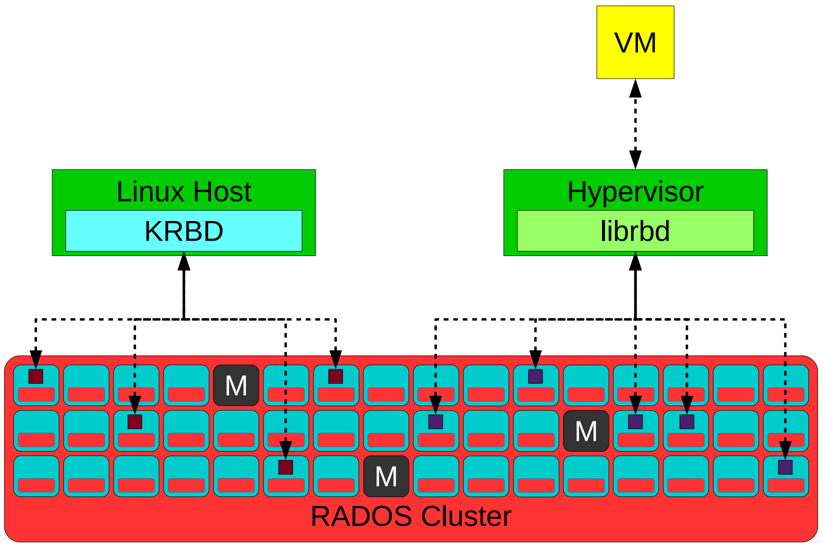


Figure 1.6: Mapping RADOS block devices to a Linux host using the `KRBD` module or to map virtual machine images stored on the Ceph cluster through `librbd` [6].

The RADOS Gateway (RGW) provides a RESTful API interface to clients, as depicted in Figure 1.7. It is compatible with Amazon S3 (Simple Storage Service) and the OpenStack object storage service, Swift. Furthermore, RGW support multi-tenancy and the OpenStack Keystone authentication services.

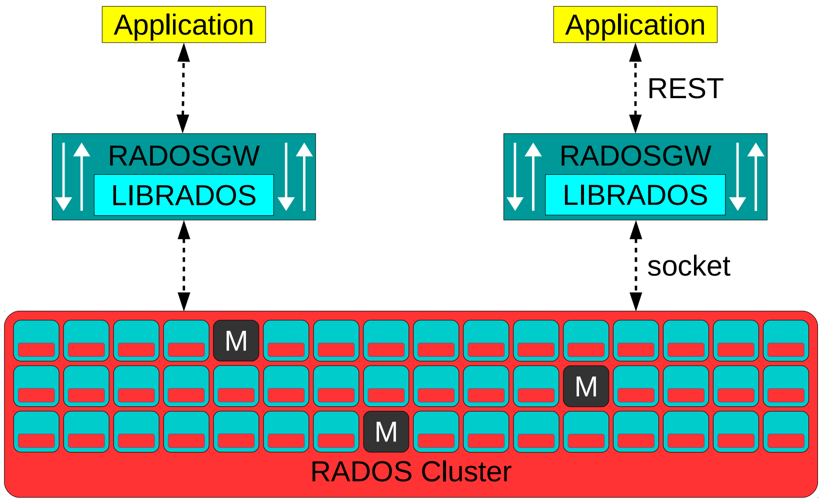


Figure 1.7: REST interfaces offered by the Rados Gateway (RGW) for accessing objects in the cluster by applications [6].

The Ceph File System (CephFS) is a POSIX compliant interface to the RADOS storage cluster. It relies on Ceph Metadata Server(s) (MDS) to keep records of the file hierarchy and associated metadata. It is used to directly mount a pool of the storage cluster on a client.

The Controlled Replication Under Scalable Hashing (CRUSH) algorithm is the essential component of Ceph. It is used to deterministically compute the placement of an object within the RADOS cluster for write and read operations. Unlike traditional distributed file systems, Ceph does not store metadata, but computes it on demand, thus removing the limitations arising from storing metadata in the traditional approach. The algorithm is aware of the underlying topology of the infrastructure, which is used to assure that data is distributed across OSDs and hosts. If appropriately configured, replication can be achieved between racks, server isles or geographical locations.

The CRUSH algorithm is also used to distribute placement groups across the cluster, as depicted in Figure 1.8, in a pseudo-random fashion. This location of the placement groups is stored in the CRUSH map. When an OSD fails, the CRUSH algorithm ensures data integrity by remapping the failed placement group to a different OSD and initiates replication.

During the initialization of a Ceph cluster, a cluster map is created to distribute the data evenly across all OSDs. Changing the weights of specific OSDs for speed or capacity planning reasons will change the cluster map automatically. Changing the cluster map manually to fit user needs and intents is also possible. Situations where manual alteration is necessary include the creation of a tiered storage pool, as described in Section 2.4.3, pinning specific pools to dedicated disk types (SAS, SATA, Flash) or to replicate the infrastructure layout and hierarchy to optimize replication and reliability,

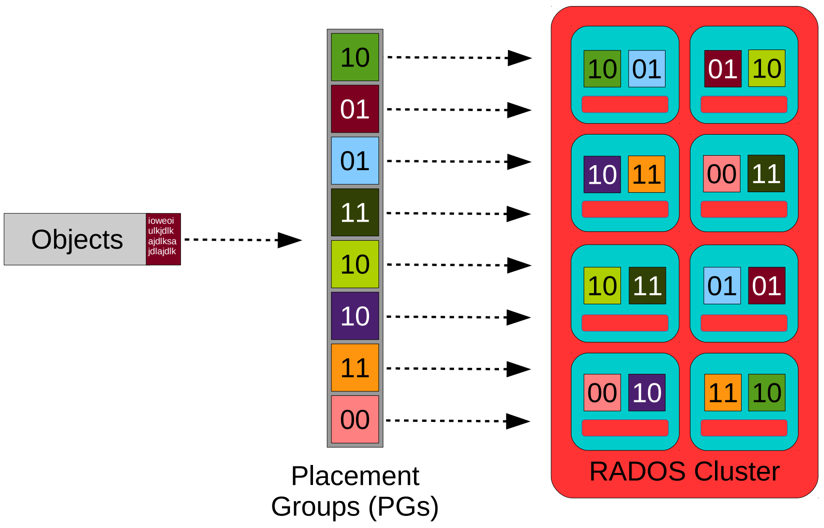


Figure 1.8: The Ceph CRUSH algorithm distributes the placement groups throughout the cluster [6].

ensuring data accessibility in the case of a power or network failure.

The CRUSH map contains records of all OSDs and their respective hosts and the replication and placement rules for pools. The OSDs are grouped in to buckets as part of the hosts. The CRUSH map supports multiple bucket types to support the hierarchical structures described above. These types are:

- type 0 osd
- type 1 host
- type 2 chassis
- type 3 rack
- type 4 row
- type 5 pdu
- type 6 pod
- type 7 room
- type 8 datacenter
- type 9 region
- type 10 root

The CRUSH map can be downloaded as a compiled binary file. It has to be decompiled into a text file before editing. The edited file has to be recompiled before it can be uploaded to the cluster to apply the new CRUSH map. A decompiled CRUSH map is

```
ceph osd getcrushmap -o crushmap-original-compiled
crushtool -d crushmap-original-compiled -o crushmap-original-decompiled
crushtool -c crushmap-original-decompiled -o crushmap-modified-compiled
ceph osd setcrushmap -i crushmap-modified-compiled
```

Listing 1.1: Commands to download, decompile, compile and upload of the Ceph cluster CRUSH map.

presented in Appendix 3.5.1. The commands for downloading, decompiling, compiling and uploading are shown in Listing 1.1.

Pools are directly associated with the placement groups of the CRUSH algorithm. Each pool has a certain number of placement groups, that store the objects of that pool, as depicted in Figure 1.9.

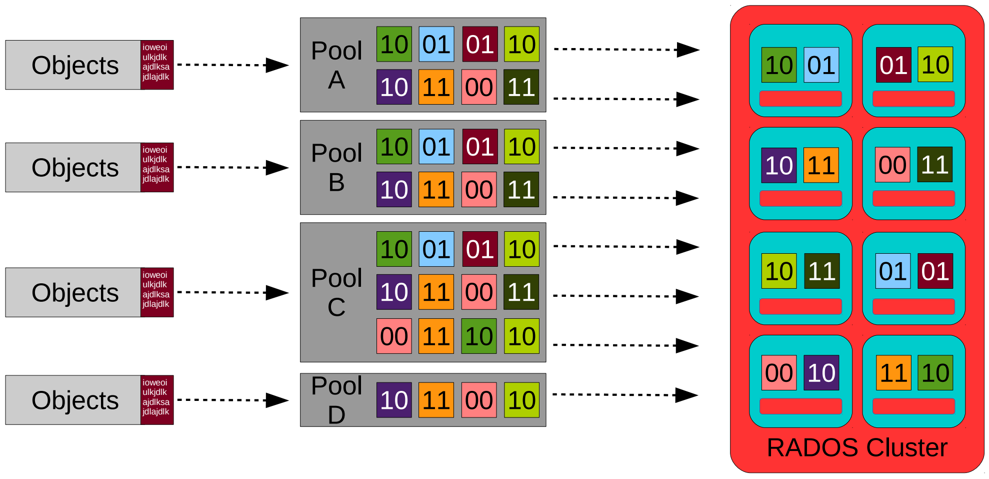


Figure 1.9: Pool placement across the different placement groups [6].

When the RADOS cluster receives a request to write data from a client using one of the above mentioned interfaces, the client uses the CRUSH algorithm to determine the placement group to which the data should be written, as depicted in Figure 1.10. This information is then used by the client to send the data directly to the correct placement group on the OSD, potentially broken up into smaller objects. In case the target Ceph pool is configured with replication, the OSDs replicate the data through the internal network before the write operation is acknowledged.

The software components/daemons used to provide the above mentioned functionalities are described in the following subsections.

1.2.1.1 Ceph Object Storage Device (OSD)

A Ceph object storage device (OSD), depicted in Figure 1.11, stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat. The

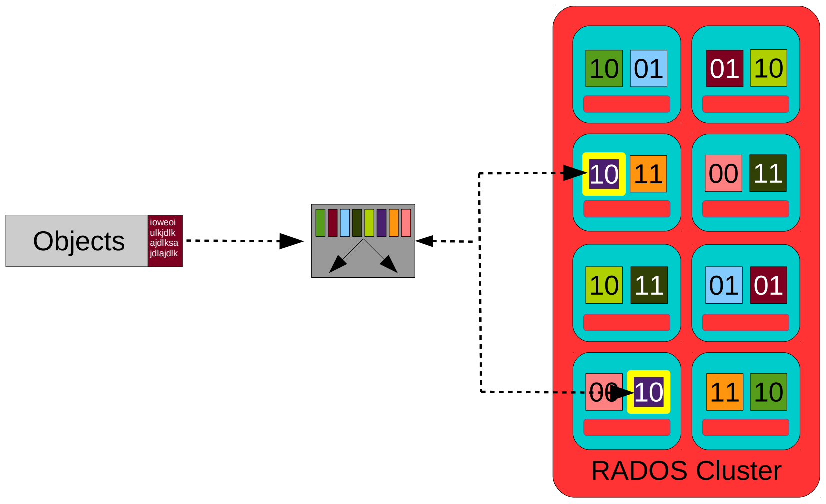


Figure 1.10: The Ceph client is capable to determine, according to the hash of the data in combination with the CRUSH function, where and to which placement group the data has to be written [6].

OSDs implement most of the functionalities of RADOS. The objects stored in the OSDs have a primary copy and potentially multiple secondary copies. If the primary copy is not accessible, due to an OSD failure, clients are able to access the secondary copy instead, which adds to the fault tolerance of the system.

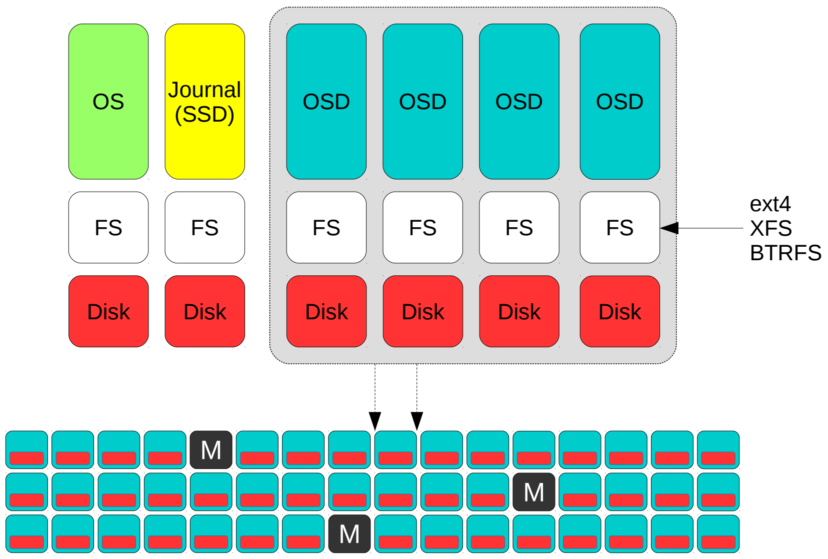


Figure 1.11: Ceph OSD resides on the physical disk on top of the file system [6].

The OSD is deployed on top of a physical storage device. Each OSD requires a data and a journal partition. It is possible to deploy them both to the same device or to separate them. It is recommended to use an SSD to store the OSD journal. Due to the generally higher performance of SSDs, it is possible to use one SSD to store multiple journals rather than using a dedicated SSD for each hard drive, but using the same device for the OSD data and journal is also supported. Depending on the workload, using a different device for the journal can significantly increase performance, since the operation is written to the journal before it is written to the data partition of the OSD, as depicted in Figure 1.12.

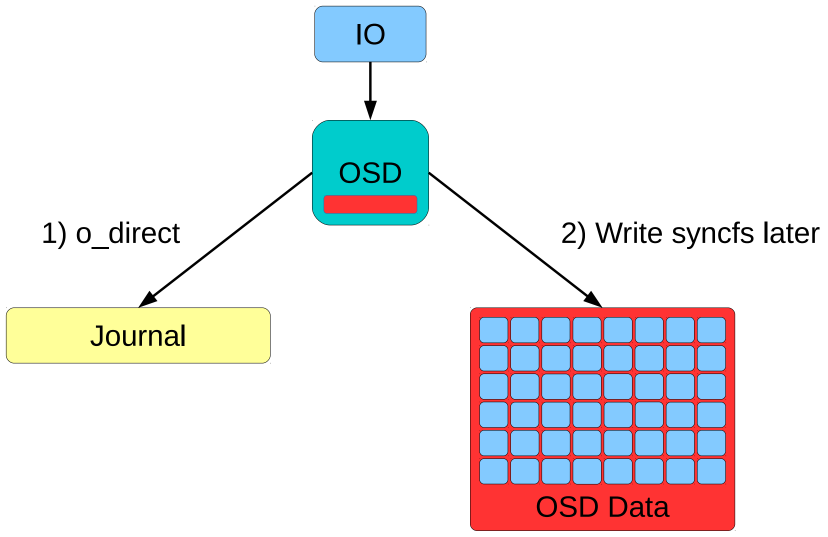


Figure 1.12: IO path within Ceph. Data is written to the journal before it is written to the OSD [6].

The OSD can not use the physical device directly. It requires a file system on the partitions (data and journal). The file systems supported on the OSDs are XFS, BTRFS and ext4. XFS is the recommended file system for production deployments, while BTRFS is not considered to be production ready. While ext4 is supported by Ceph and considered production ready, it has limitations that prevent the construction of large scale clusters (e.g., limited capacity for XATTRs).

Generally, Ceph deployments use the replication mechanism of RADOS. However, creating replicas for each object consumes a considerable amount of storage. A storage cluster with a raw capacity of 300TB is only capable of storing 100TB with a replication count set to 3. Therefore, Ceph supports erasure coding of the pools. In this operating mode, CRUSH is only used to distribute the data across the OSDs, while the OSDs are hosted on RAID arrays to provide the necessary redundancy. This can decrease the overhead in the system from 300% to 50% (when using RAID 6 with 4 data and 2 parity drives), but increases the cost of recovery from drive failures.

1.2.1.2 Ceph Monitor (MON)

A Ceph Monitor (MON) monitors the health of the RADOS cluster. It maintains maps of the cluster state, including the monitor map (location and status of all monitors), the OSD map (location and state of all monitors), the Placement Group (PG) map (location and state of all PGs), the CRUSH map (CRUSH rules and OSD hierarchy) and the MDS map (location, state and map epoch of all MDSs). Furthermore, it maintains a history of each state change of these maps.

The Ceph MONs do not serve data to the clients. They only periodically serve updated maps to clients and other cluster nodes. Thus, the MON daemon is fairly lightweight and does not require excessive amounts of resources. Typically, a Ceph storage cluster contains multiple monitors to add redundancy and reliability by forming a quorum between the MONs to provide a consensus for distributed decision making. This requires at least half of the total number of MONs to be available to prevent uncertain states. Furthermore, a minimum of 3 MONs is required in a production cluster.

1.2.1.3 Ceph Metadata Server (MDS)

The Ceph Metadata Server (MDS), depicted in Figure 1.13, stores metadata on behalf of the Ceph File System (CephFS). Ceph Block Devices and Ceph Object Storage do not use the MDS. Ceph MDSs enables users to mount a POSIX file system of any size and to execute basic commands, such as `ls` and `find`, without placing an enormous burden on the Ceph Storage Cluster. It contains a smart caching algorithm to reduce read and write operations to the cluster. The metadata information is captured in a dynamic subtree with one MDS being responsible for a single piece of metadata. Dynamic subtree partitioning allows for distributing the metadata across multiple MDSs that handle metadata information of a particular part of the tree. Furthermore, this approach allows for quick recovery from failed nodes, joining and leaving of daemons (scaling) and rebalancing.

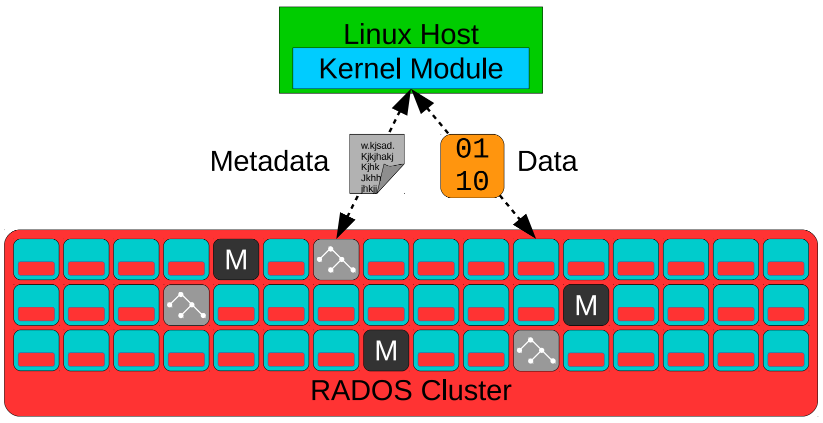


Figure 1.13: Metadata server saves the attributes used in the CephFS file system, directly used by the clients [6].

1.3 OpenStack in combination with Ceph

Ceph integrates well with the different storage services of OpenStack. A detailed description of OpenStack and its services is presented in Appendix A. As depicted in Figure 1.14, many of the OpenStack services can directly use the RADOS cluster as a storage backend. The OpenStack object storage service, Swift, uses the RGW in combination with OpenStack Keystone for authenticating accesses. The OpenStack image service Glance can use the RBD interface to store VM images on a Ceph pool. The OpenStack block storage service Cinder stores block storage devices on pools and uses the RBD interface. The OpenStack compute service Nova uses the block devices and images of Cinder and Glance directly without any proxy through the RBD interface. While OpenStack Cinder supports multiple storage backends, such as different Ceph pools or multiple storage implementations, such a feature is not supported within OpenStack Glance[1].

Cinder supports multiple backends concurrently. This offers the possibility to create different Cinder Tiers that are connected to different backend systems with varying capabilities and features, such as having one proprietary storage system and a network share set up as the backends, or to use different pools from Ceph or completely different Ceph clusters. This allows for differentiated storage solutions, such as low specification versions with no resilience to failures and high performance solid state drive data stores, at different price levels.

When Ceph is used to provide the storage for the previously mentioned storage services, it keeps them within a single system and administrative domain, rather than deploying dedicated data stores for each OpenStack service. Furthermore, it allows for better resource utilization and capacity planning, since the Ceph cluster capacity can be increased by adding new OSDs when required, rather than overprovisioning the data store for a specific storage service.

With each release of OpenStack, functionalities are added and improved to support new features and, in some cases, direct integration of Ceph interfaces.

1.4 Related Work

The following subsections outline the activities and approaches adopted by other researchers in attempting to improve the Ceph storage system. Much of this work is based on suggested improvements derived from synthetic workloads (these are referred to in the literature as benchmarks, although, they are not used as benchmarks in the strict sense of the word).

[1]https://blueprints.launchpad.net/glance/+spec/multi-store

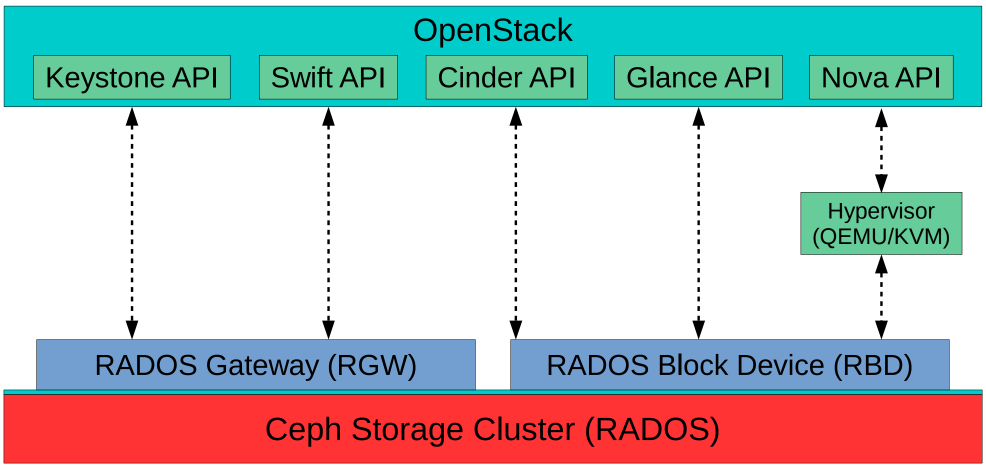


Figure 1.14: Ceph OpenStack integration and the used interfaces [6].

1.4.1 Ceph Performance Testing

The hardware manufacturer, Intel, has initiated the Ceph Performance Portal [7], which aims to track the performance advancements or regressions between the different releases of Ceph. Different approaches to test different components of Ceph are used. For testing the performance of the object storage RADOS Gateway (RGW), the COSBench benchmark is used.

To test the block storage performance of their deployment, fio (Flexible IO) with different access patterns is used. In this scenario 140 VMs are used spread across 4 compute nodes. The storage backend comes in two varieties that differ in CPU and hard drive types. The number of storage nodes (four) and hard drives (10 per node) are identical as is the usage of SSDs for the Ceph journal. The performance of these setups is determined in a default and a tuned configuration. The relative performance degradation or gains are then presented for individual access patterns.

In a parallel effort, UnitedStack share the configuration of their Ceph Deployment that drives their OpenStack deployment [8]. Their hardware configuration is not explained in detail, but their tuning parameters and system performance are in the public domain.

Han [9] keeps the community aware of Ceph and OpenStack developments at RedHat and describes different configurations and storage hardware, such as testing different SSDs for their applicability for Ceph journaling.

While studying the related literature, the author undertook an empirical study of the configurations presented in the foregoing initiatives. The results of this study and the impact of the alternative configurations suggested on performance is presented in Appendix B. It can be seen from the study, that the suggested configurations deliver performance improvements across a limited number of access patterns and sizes only. However, these limitations are often not articulated fully in the literature and most likely derive from the fact that synthetic benchmarks are used over a limited range of access sizes and patterns. The authors study uses real cloud workloads over a larger range of access sizes and patterns and hence reveals the potential limitations of the results reported in the literature.

Wang *et al.* [10] have tested the scalability of Ceph in a high performance computing environment. In their testbed Ceph OSDs were installed on exported LUNs from a high performance RAID array. A total number of 480 hard drives were used (200 SAS, 280 SATA). The LUNs were configured to 8+2 RAID 6 groups and exported to 4 servers over InfiniBand. The clients, 8 servers, were also using InfiniBand.

The experiments performed include the impact of the file system used on the OSDs, networking configuration and a parameter sweep for testing different Ceph configurations. The scaling tests were performed using different client counts with 8 processes on each client. Furthermore, different versions of the Linux Kernel were tested to show their performance over time. In this deployment Ceph was able to perform close to 70% of the raw hardware capability at RADOS level and close to 62% at file system level.

The use of a parameter sweep to test different configurations for impact on performance is highly important. The paper does not reveal what accesses sizes were used to determine the performance difference.

In a more detailed work, Wang *et al.* [11] evaluate the scaling behaviour and optimization potential of Ceph. In this work, optimizations were performed on RADOS and CephFS. To that extend, various optimizations within and outside of Ceph were examined for their effect on performance. A mapping or verification with a non-synthetic workload is not performed. This work shows the effect that configuration optimization has in a large scale system and that Ceph leaves room for optimization.

In general, published peer-reviewed work about Ceph configurations is very limited. While Ceph is often cited in publications, rarely it is used as the subject of the work. The publications that are available are mostly focused on a specific problem, such as the performance in all flash deployments. It is possible that tuning of the file system is performed more in industry, rather than academia, which comes as a surprise, given the large possible configuration space the system offers.

1.4.2 Deployments

Lee *et al.* [12] use Ceph for designing an archival storage Tier 4 for The University of Auckland. The system is supposed to serve as a low cost storage system that is exposed to sequential writes and rare random read accesses, before the data is stored on tape. The authors investigate different server types that meet performance requirements (sequential writes 200MB/s, random reads 80MB/s) and their total cost of ownership (hardware and running cost).

Ceph is used in the evaluation testbed with the default configuration and the replication count set to 1, since it only serves as an intermediate storage Tier. Using a replication count of 1 is not recommended in production systems due to high risk of data loss in the event of hardware failure, but is understandable when aiming to minimize cost for an intermediate storage Tier.

The different hardware configurations are evaluated by exposing the system to the well known and understood workload. Other workloads are not important for the use case and are therefore not tested. Such an approach is well suited to determine the performance for a specific workload (traced, analysed and repeatedly occurring), but can not be used to infer the performance of the storage system under other workloads. This makes this approach unsuited for cloud environments, where workloads are diverse and ever changing.

In a previous paper, Lee *et al.* [13] tested different archive workloads that differed in their used file sizes. Furthermore, the impact of changing the file system from ext4 to btrfs in a Ceph deployment was investigated. The Ceph deployment used the default configuration. Configuration changes were not tested.

1.4.3 Cloud Benchmarks

The Yahoo! Cloud Serving Benchmark (YCSB) [14] aims to create a standard benchmark to assist in selecting the correct cloud serving system for workloads such as MapReduce (i.e., Hadoop [15]). These cloud serving systems provide online read-/write access to data. While relational database systems were previously used for such tasks, key-value stores and NOSQL systems, such as Cassandra [16], Voldemort [17], HBase [18], MongoDB [19] and CouchDB [20], are becoming more important as they are able to scale well.

YCSB is an open source tool that can be easily extended to add new workloads or add new datastores as the backend for the workload tests. The benchmark supports two different types of testing tiers. Tier 1 is focussed on performance. It aims to test the tradeoff between latency and throughput to determine the maximum throughput a specific database system can sustain before the database is saturated and throughput does not increase, while latency does. Tier 2 aims to determine the scaling ability of a specific database system. Two types of scaling are supported: scale-up and elastic scaling. Scale-up tests how a database performs when the number of hosts is increased. This approach is a static scaling test. A small cluster is tested against a specific workload and dataset. Then the data is deleted and the cluster is expanded. The larger cluster is then loaded with a larger dataset and the same workload is re-executed. For the elastic speedup the number of database servers is increased while the system is subjected to a specific workload. This test shows the dynamic scaling ability of the system that will have to reconfigure itself to the changed configuration.

The core workloads of YCSB are designed to evaluate different aspects of the system. They contain fundamental kinds of workloads created by web applications, rather then creating a model of a very specific workload, as done by TPC-C [21]. This wide range approach allows for testing of different characteristics of the database systems. Some systems are highly optimised for reads, but not for writes, while others are optimised for inserts and not for updates. To assist in this approach, different access distributions are supported by YCSB, that allow the modelling of different application characteristics.

This benchmark is of high importance to current cloud deployments and the workloads deployed to them (web services and data analytics). The aim of the benchmark is to show performance and scalability differences between different database systems. The performance of the database is directly effected by the available resources (CPU, memory, storage).

Zheng *et al.* [22] have, in collaboration with Intel, developed the Cloud Object Storage Benchmark (COSBench). It is designed to benchmark cloud object storage systems with different access patterns and sizes to reflect the workload such systems face, such as images and video files. The benchmark supports a large variety of object storage systems, such as OpenStack Swift, Amazon EC2 and Ceph.

The benchmark contains two components. The controller is the logic of the benchmark that sends the workload to workers and gathers benchmark metrics, such as OPS/s and 95^{th} percentile latency, and combines them into a benchmark report. The second component is the driver/worker. It executes the workload by sending one of six workload operations (LOGIN, READ, WRITE, REMOVE, INIT, DISPOSE) to the storage system. Furthermore, it implements the authentication methods for the different storage services, such as OpenStack Keystone.

While the benchmark is of high significance for cloud systems, it is limited to testing the object storage service. In Ceph it is implemented by exposing the storage system through the RADOS Gateway (RGW). It uses an Apache webserver and FastCGI. While the RGW is subjected to the performance of the underlying Ceph cluster, it has over 105 parameters that can be modified for adaptation to a use case. In addition, other tuning options are available for Apache itself.

In this work the block storage performance within virtual machines is investigated. The storage service of interest is therefore OpenStack Cinder and the RADOS block device (RBD). COSBench does not support this interface type, therefore it has not been used in the experimental section. For systems that are used for content delivery, COSBench is a very valuable tool to determine performance when exercised by a large number of clients.

1.4.4 Benchmarks

Traeger *et al.* [23] performed a nine year study on file system and storage benchmarks. A total of 106 papers were studied for the benchmarks and configurations used. The benchmarks were categorized into three:

Macrobenchmarks: performance is tested against a particular workload that is meant to represent some real-world workload, though it might not.

Trace Replays: operations which were recorded in a real scenario are replayed (hoping it is representative of real-world workloads).

Microbenchmarks: few (typically one or two) operations are tested to isolate their specific effects within the system.

The importance of benchmark configurations and conditions (warm or cold caches) are stressed as are the benchmark runs. A larger number of runs should result in a better representation of the system performance since a running average can be created, rather than relying on a single run that could be influenced by unexpected system events. The number of runs greatly varies (1-30) between the different reported results. It was recommended that outliers generated during these runs should not be discarded, as systems can not be tested in a sterile environment.

This work is a guide to how benchmarking should be performed and what information should be revealed about the testing procedure. Configuration settings are important for verification and for understanding the performance metrics. Using a larger number of benchmark runs helps to obtain results reflecting the average system performance. While outliers are natural to some extend, due to the nature of testing on a system where system processes are executed in parallel, they don't represent the average system performance. The approach taken in this dissertation borrows heavily from the methodology derived from this study.

1.4.5 Scale

Lang *et al.* [24] show the challenges when using a distributed file system at a massive scale in a super computer. The file system used in the Intrepid IBM Blue Gene/P system at Argonne Leadership Computing Facility (ALCF) is the Parallel Virtual File System (PVFS). The deployment uses enterprise class storage arrays (DDN 9900) to export block devices as individual logical units (LUNs). Each LUN is a RAID 6, where the parity calculations are handled by the DDN 9900 rather than the storage nodes.

The block devices are subsequently exported by the storage nodes through the PVFS to the stateless clients.

Such an approach is also possible using Ceph, where the OSDs are located on exported LUNs from a SAN or internal RAID controllers. Nelson [25] [26] has tested such an operating mode for Ceph using different storage controllers and exporting modes for performance analysis. Using Ceph with RAID volumes with parity calculations (5 or 6) did not perform well. Using a one-to-one mapping between the OSD and the physical disk displayed a better performance.

Sevilla *et al.* [27] investigated the performance differences between scale-out and scale-up approaches when using the Hadoop workload. They investigated the penalty that arises from adding fault tolerance in terms of checkpoint intervals and sequential and parallel access speedup. The speedup factor is limited by the sequential portion of the data accesses, according to Amdahl's law [28]. The authors conclude that scale-out and scale-up storage solutions are both limited by the hardware and the interconnect speed. Without the appropriate interconnect between the storage nodes and the compute nodes a speed bottleneck will result.

1.4.6 Optimization

Costa and Ripenau [29] propose an automated configuration tuning system for the MosaStore distributed file system. Distributed file systems that come without the option to alter the configuration come with a one-size-fits-all solution that takes away the option to extract more performance or to adopt the system to the environment and workloads. Versatile storage systems on the other hand expose configuration parameters and allow alteration at runtime. Although versatile storage systems allow for better adoption to the workload, a knowledgeable administrator is required to do the right configuration changes and understanding the characteristics and requirements of the workload.

In their work they add a controller node that takes in application and storage monitoring metrics and makes decisions depending on the optimization goal (i.e., throughput, storage footprint). The controller uses a prediction mechanism that estimates the impact of a specific configuration on the target performance metrics. The configuration changes are then sent to the actuator that alters the storage systems configuration at runtime. The configuration changes presented in the work are limited to the deduplication engine that is part of the storage system. Storage performance is presented, with and without, the deduplication, and when using the automated approach, using data with different similarity factors that affect the deduplication performance.

The system is capable of switching the deduplication engine on when the similarity ratio, which is determined on the client, is sufficient to improve performance and reduce storage capacity utilization. The workload used is a checkpointing application, such as the bioinformatics application BLAST that periodically writes checkpoints. The limitation of a single presented configuration change, limits the applicability of the approach for a versatile distributed storage system that offers hundreds of configuration parameters. Therefore, although very valuable work, it can not be applied to a distributed storage system like Ceph where the effects of configuration changes are unknown.

Modern Solid State Drives implement many optimization techniques to compensate for the limited number of write cycles associated with NAND cells [30]. The techniques used are, among others, over provisioning and compression. Over provisioning reserves NAND capacity for write operations since Flash cells can not be overwritten without being erased first. Furthermore, the reserved capacity is used for garbage collection, SSD controller firmware and spare blocks to replace failed flash blocks. Over provisioning is used with percentages as high as 37% [31]. Generally, higher over provisioning ratios result in higher write performance, as shown by Tallis [32] and Smith [31], and longer Flash life.

Compression algorithms implemented directly on the SSD controller aim to reduce the Flash writes and improve performance for compressible data [30] [33]. The performance of devices with compression varies depending on the compression entropy of the data, as shown by Ku [34] [35] and Smith [31]. To avoid wrong results from synthetic benchmarks for performance evaluation on SSDs with embedded compression algorithms, the benchmark has to use incompressible data. With highly compressible data the results do not give an accurate performance representation.

The impact of compression on the performance of SSD controllers is not significant as the synthetic benchmark used for the baseline evaluation uses incompressible data. The over provisioning ratio on the SSDs affects the raw performance of the device and is therefore directly exposed through the testing method.

In the literature, many other storage systems are analysed and improved to achieve a higher performance. In relational database systems the approaches include optimising the internal data structure of the database [36], table sizes, normalization [37] [38] and query optimization [39] to improve database performance. These approaches are not applicable to Ceph, where the data is distributed in a random fashion across the whole data cluster using the CRUSH algorithm and where storage accesses occur at random times with varying frequencies, sizes and access patterns. While it is possible to tune a system to support a specific type of access pattern, workloads generally do not consist of just a single access type and pattern. For storage area networks (SAN) the literature proposes solutions to improve performance by tuning the connection and protocol between the initiator and the target. Oguchi *et al.* [40] have evaluated buffer sizes and behaviour in iSCSI connections to improve SAN performance. Ceph and traditional SAN systems are similar in that they both use network connections to connect clients to their respective storage systems. These network connections are essential components of the environments of these systems. SAN system performance is strongly correlated with the performance of the network communication channel being used and so improving the quality of that channel is important when trying to get the most from the SAN system. The same approach could indeed be adopted for Ceph and corresponding performance improvements could be expected. However, the performance of a Ceph system depends subtly on many components and not the network performance alone.

It can thus be seen that the techniques used for improving performance of traditional storage systems cannot be adopted in their entirety in the Ceph context, where many components, each of which can be configured in a multitude of ways, interact to subtly affect the overall performance. A different approach is thus required, and one such direction, based on mapping workload characteristics to relevant configurations of components in the environment of Ceph, is explored in this dissertation.

The approach taken in this thesis is the development of a structured method of empirical analysis as might be done as a necessary initial step of a more formal approach. The motivation for this is that the impact of a specific configuration change in a particular environment is unknown prior to an empirical test due to the vast number of possible configurations and the unknown relationship between the individual parameters. Thus an approach is adopted where a structured empirical method is used to discover the underlying relationships between configurations, workloads and performance. The obtained performance information can be used subsequently to perform a mapping between configurations and specific workload requirements, and to provide the underlying data for tools such as constraint programming or big data analytics.

Due to the large number of possible configurations and environmental components, such as operating system and hardware configurations, it is very difficult to produce a formal model that is comprehensive and accurate while at the same time compact enough to be tractable for analysis. The insights into the underlying relationships gained from the empirical data gathered from the structured method presented here can reduce the problem state space while accurately reflecting reality. This can provide the starting point for a more scientific approach employing a model that is both accurate and tractable.

Similarly, while a statistical method is possible, it was decided to address the empirical gathering of data in advance of such a method in order to gain a better understanding of configurations, parameters, workloads and performance. The understanding gained from the empirical data could subsequently assist in the choice of statistical model and in building an accurate yet tractable statistical model.

1.5 Dissertation Outline

In this chapter the OpenStack and Ceph systems were introduced and work related to the performance of storage systems was articulated. The proposed methodology to find a configuration that improves workload performance is presented in Chapter 2. It describes in detail the individual steps that have to be taken to get performance baselines of different configurations, how to generate the configuration, how the workload trace is performed, how the mapping between the different storage configurations is performed and subsequently evaluated. Specific cloud workloads are introduced and selected for empirical studies and the three environments of the distributed storage system and their impact on tuning are presented.

In Chapter 3 the testbed used for the empirical experiments is described and the baseline performances of the different configurations are determined. In Chapter 4 the workloads are characterized before a mapping of these characteristics and the baseline performances is performed in Chapter 5, where the proposed mapping is evaluated.

Chapter 2

Methodology

To improve the performance of a Ceph storage cluster for cloud workloads requires a set of methods to be able to quantify such improvements.

2.1 Scientific Framework

The method proposed in this work is an evaluation of the impact that different storage configurations have on performance. The proposed method is empirically evaluated in the context of a highly configurable distributed storage system. Due to the vast number of possible configurations, creating a model that capture all components and interactions is difficult to achieve if not even impossible. Furthermore, the interactions and relations between the parameters is unclear and not well documented. Therefore, an approach has been developed as part of this thesis to evaluate the impact of individual parameters in a smaller scope in isolation rather than testing a large number of parameters as a set, since impacts might otherwise be obscured.

Given the performance fluctuations that are associated with different hardware components and given the fact that there is no structured methodology available for determining in a technology independent manner how the Ceph parameters should be chosen, this thesis attempts to provide a systematic methodology that can be used, regardless of the underlying technology, to identify and to set appropriate Ceph parameters so that a positive impact on workload performance is achieved when the characteristics of the workload are taken into account.

Modern operating systems and distributed storage systems consist of many components that interact with each other on multiple occasions. Each storage access made by a remote client involves multiple components, rather than a single entity. Furthermore, other system services are running on the storage host and might compete for resources or create I/Os of their own, interfering with the storage system. As a consequence,

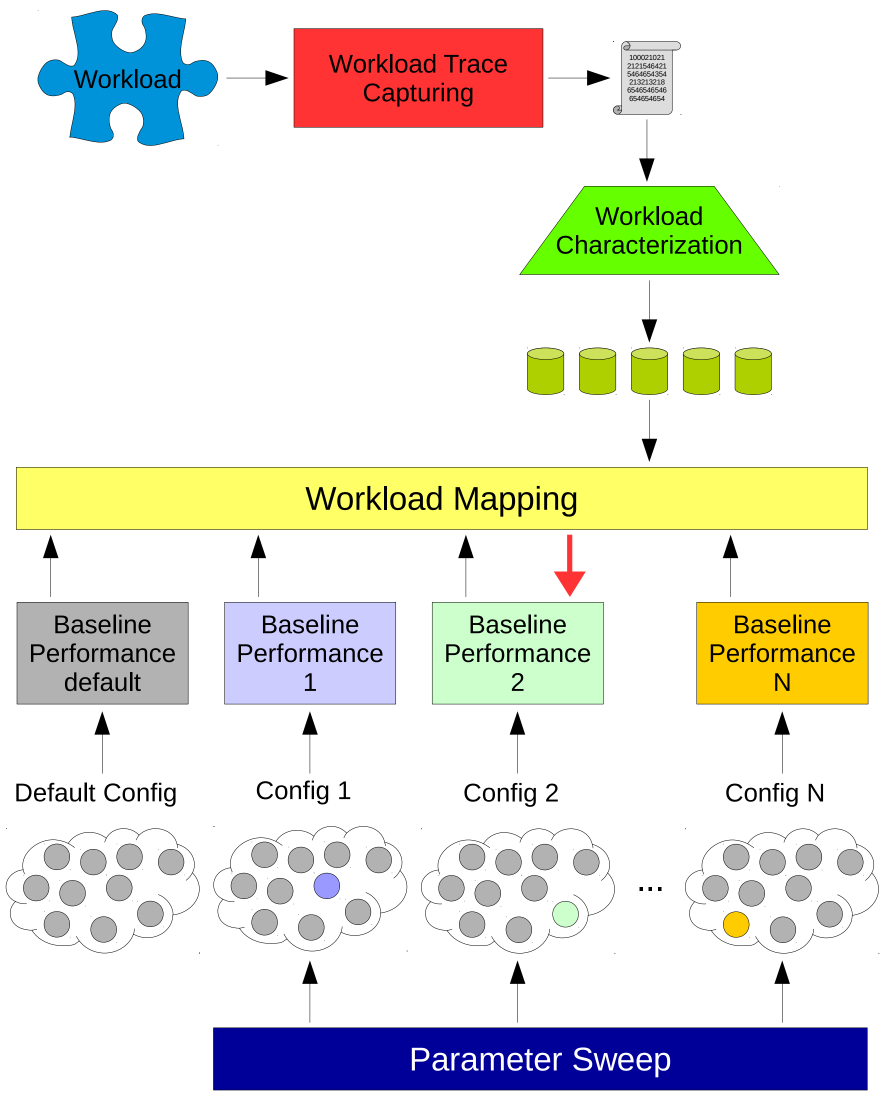


Figure 2.1: Methodology overview.

no two runs can be completely identical to each other. To mitigate these effects the proposed methodology executes each test multiple times to capture an average performance across multiple runs. In contrast to the configurations proposed in the public domain, the proposed method also measures the impact each configuration change has on performance, rather than proposing a single configuration without quantifying and determining the impact of each parameter change as is the prevalent approach taken in the literature.

Since the storage access characteristics and requirements differ between applications, it is not possible to propose a configuration that performs better for all workloads nor in all environments. Furthermore, specific workloads come with non-performance related constraints that will have an effect on performance, such as the replication count. A higher number of replications increases resilience against hardware failures but reduces write performance, due to the increased numbers of copies that are generated before a write request is acknowledged. Consequently, the proposed method gives users a methodological way to assess performance changes and their effects on workloads. Furthermore the proposed methodology is not limited to a specific storage system, but can be used to improve other storage systems that come with tunable parameters, as well.

When the proposed methodology is executed on a larger testbed, the results will differ due to scaling effects within the system. Parameters that have minor impact in a small storage cluster may become a bottleneck and potentially limit the achievable performance. As shown by Oh *et al.* [41], certain parameters may become a bottleneck when changing the environment and the used storage device technology. The testing methodology will stay identical for those cases, since it covers general access patterns with multiple clients, but each testing iteration might experience lower runtimes from the scaling effects of the system.

2.2 Procedure

In this chapter a number of methods for increasing the performance of a Ceph cluster in an OpenStack environment are presented. To measure performance improvements accurately, it is necessary to establish a number of baseline performances. A baseline performance is the performance that results from using a baseline access pattern and baseline access size in the system given a particular configuration of Ceph. This is not a trivial process, however only with this knowledge is it possible to measure the effectiveness of the performance enhancing methods. This process involves the creation of a collection of different configurations that are evaluated and compared for their performance gains and losses. In this study a parameter sweep is used to generate these different configurations. In contrast, other methods to change the Software-defined storage system Ceph exist and are shown in Section 2.4, however they are not used in this study, since the chosen method gives the best opportunity for altering the system characteristics and performance. The baseline performance metrics, by themselves, are not useful in making any tuning suggestions for a specific workload with its own requirements and characteristics. Therefore, a detailed storage trace has to be made and analysed. The characteristics of the workload are identified and are mapped on to the most appropriate configuration to run that workload. To demonstrate the applicability of the foregoing process a broad range of workloads were used to empirically test the system. These workloads are practical in nature and have been chosen in accordance with the results of the OpenStack user surveys. The benchmarks for these chosen categories of workloads are described in Section 2.3.

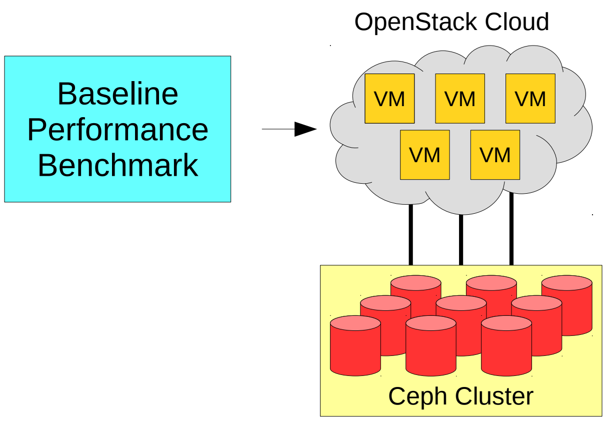


Figure 2.2: Performance baseline generation using an OpenStack cloud deployment, which uses a Ceph cluster as a storage backend, with multiple concurrent virtual machines.

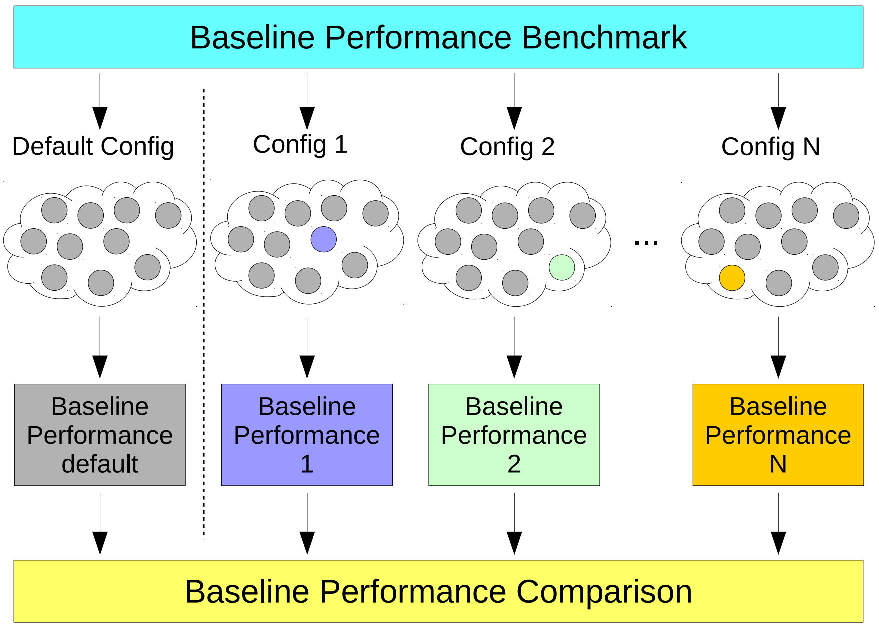


Figure 2.3: Performance baseline generation and comparison.

2.2.1 Baseline

To measure the performance gains or losses of a specific configuration, it is necessary to get a reliable and precise baseline reference. As the aim of this work is to improve the performance in a cloud environment with the Ceph storage being the sole backend for all storage services, the performance is tested with multiple concurrent virtual machines running off Cinder block storage volumes as depicted in Figure 2.2. This approach is useful in testing the system with the interfaces it will use in a production cloud deployment, such as an OpenStack deployment.

To avoid any scheduling or capacity problems on the compute nodes, the virtual machines are configured to use less resources than available on the physical hosts. To get a baseline performance, the throughput of each virtual machine is recorded when executing the Flexible IO Tester (fio) benchmark (described in more detail in Section 2.3.1.1).

The benchmark is executed with 5 different block sizes (4KB, 32KB, 128KB, 1MB, 32MB) to get a detailed measurement of the overall performance and the different access sizes that appear in real systems (presented in more detail in Section 4.2). Testing the system against 5 access sizes is substantially more detailed then previous work, such as the work performed by Intel [7] [42] or Ceph/Inktank [43]. Using even more access sizes would improve the capturing of the respective performance of the storage system under other occurring access sizes, but would increase the runtime for executing the tests for each configuration. Each of these access sizes is then tested for its sequential and random read and write speeds, which results in 20 different benchmark runs (i.e., 5 access sizes $\times$ 4 access patterns). Accounting for eventual differences between the virtual machine clocks and other activities that might have an impact on the measurement, the benchmark is run 9 times and the average performance is calculated. Therefore, a total of 180 benchmark runs were performed for each configuration, resulting in a test duration of about 20 hours.

The resulting different performance baselines can then be used to compare the different configurations directly, as shown in Figure 2.3, or for the mapping process between a workload and different configurations, as shown in Section 2.2.4.

The synthetic baselines are constructed with a clear distinction between sequential and random workloads. This is, in general, not found in real workloads. This point is revisited in Section 2.2.4.

2.2.2 Parameter Sweep

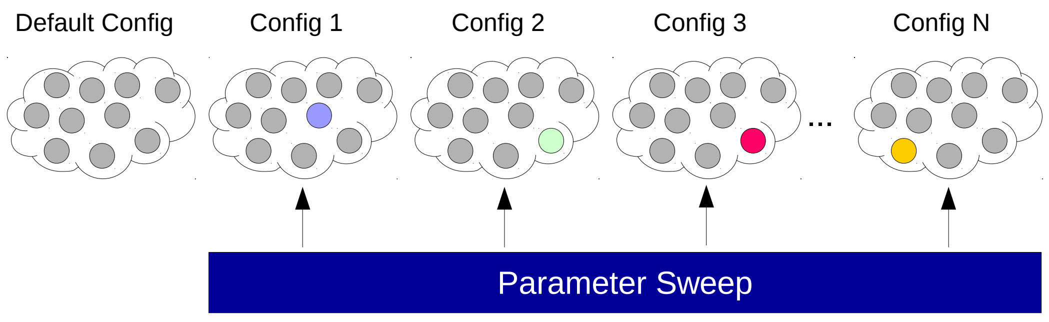


Figure 2.4: Parameter sweep across Ceph parameters resulting in different configurations. Sweeping is also performed on a single parameter (Configurations 2 and 3).

As Ceph is a highly customizable Software-defined storage system, it is difficult to find the correct configuration for a specific workload. Furthermore, many of the parameters lack a description of their impact on performance.

To identify the impact of a single parameter on the overall performance of the storage cluster, a couple of parameters will be chosen and tested individually with values that deviate from the accepted default, as shown in Figure 2.4. The altered configuration will then be subjected to the same testing procedure to establish the baseline performance for that specific configuration. The sweep is not confined to changing the relative values of different parameters, but also involves altering a single parameter by increasing or decreasing its value, as depicted with Configuration 1 and 2 in Figure 2.4. As many of the parameters support signed and unsigned integers, doubles and long values, there are up to 2^{64} possibilities for each type. Exploring them all is therefore an infeasible task.

Some of the parameters have a strong relationship to the hardware used. Mechanical hard drives, due to their design, are unable to handle multiple threads accessing different parts of the disk at the same time due to physical characteristics of the disk head. For solid state drives with no moving parts this relationship may be very different. Increasing a parameter such as the OSD threads, for example, may result in better utilization for one storage device type over another.

The values of some parameters may be tightly coupled to the value of others. Also, some parameters may have an effect if others have been configured in a particular way. All the parameters used in an InfiniBand deployment, for example, will only become active in deployments that use InfiniBand for their interconnect. In other deployments these parameters are dormant and do not influence the performance.

In Section 3.4, the impact of a single parameter on disk throughput is determined in an OpenStack environment through multiple concurrent VMs. In total 24 configurations were tested and analysed for performance using the testing pattern described in Section 2.2.1.

2.2.3 Workloads

To determine configurations of a distributed file system that improve performance for cloud workloads, it is necessary to use representative application types. The workloads that are used in production OpenStack deployments are discussed in Section 2.3. Choosing applications that belong to the correct application types, such as web services, is crucial to get a proper understanding of how the tuning is affects performance in real deployments.

The collection of benchmarks is then analysed in an isolated environment for it's storage accessing characteristics (see Figure 2.5). This requires a detailed storage trace of that application to extract the necessary information to generate a mapping to configurations in the subsequent step. The required information extracted from the trace includes, but is not limited to, the dominant access size, read-write ratio and the randomness of the accesses.

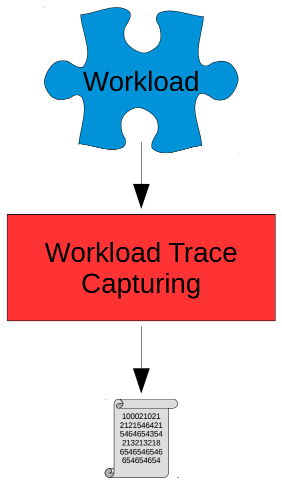


Figure 2.5: Workload trace file generation of all individual storage accesses and their sizes.

Additional information, such as the queue depth during access, is of vital importance if the application is deployed on a physical host, but it loses importance if deployed on a distributed file system in a virtualized environment when potentially thousands of VMs access the data store simultaneously. The burstiness of workloads (*i.e.*, bursts of accesses with short duration interspersed with periods of inactivity) are not looked at in this work, but can be of importance for deployments that have SSD caches available for tuning flushing characteristics.

The chronological chain of events is also of importance. If a workload is using a read-once-write-many approach, such as the workload presented in Section 4.3.4.3, where read accesses all happen at the beginning and only afterwards are write operations performed as the data is cached by the operating system, changes to the caching Tier mode can have a positive effect.

The storage trace of the application does not have to be captured on the physical storage system being tuned. Using a different host to capture the trace might also improve the quality of that trace, since that host can be chosen to avoid shared accesses to the storage system and any consequent influences that that sharing might have on the gathering of the trace.

2.2.4 Mapping

After identifying the performance gains and losses of each configuration and getting the detailed application traces and the extracted characteristics of the storage accesses, a mapping between them has to be constructed. The traces are analysed for their dominant access types and then linked to the tested configurations.

To map a workload trace to the previously tested configurations, the trace access sizes

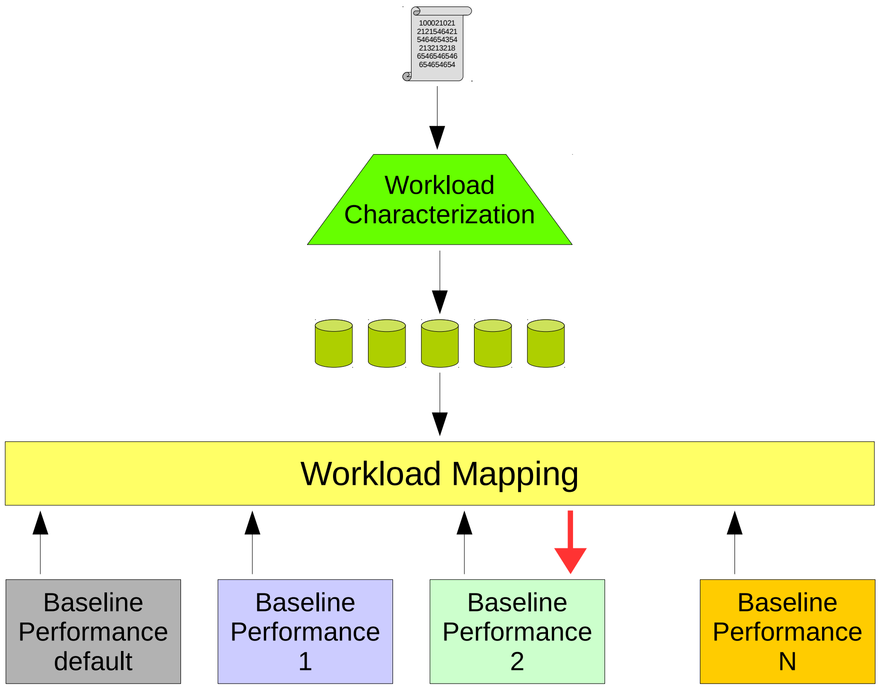


Figure 2.6: The workload trace file is characterized and accesses are mapped to 5 access size bins for reads and writes. This binned workload is mapped to the performance baselines of the different configurations, resulting in a recommendation for a performance enhancing configuration (red arrow).

are combined into bins, that are in turn mapped to an access size of a baseline. These bins are created for both read and the write accesses. As previously mentioned, each baseline is constructed so as to consider 5 different access sizes. However, the tracing tool may report on up to 18 different access sizes within a workload trace. It is therefore necessary to map each of these 18 different access sizes into one of five bins, associated with a particular access size in each baseline. The smallest block size recorded in the trace was 4KB, while the largest was 4MB, these are related to the host configuration and how it exports the disk to the VM. Access sizes in between showed peaks at 32KB, 128KB and in some cases 512KB and above.

Bin sizes were chosen by following the typical access sizes found in the literature (128KB and 1MB). To increase granularity of system evaluation, further access sizes have been added. 4KB and 32KB for better evaluation of smaller accesses and 32MB for very large accesses. While 4KB and 32KB accesses were very frequent in the application traces, 32MB was not. As stated above, the largest access size recorded was 4MB, but 32MB was kept as it is the largest IO size on VMware ESXi server [44]. Therefore 5 bins were created for 4KB, 32KB, 128KB, 1MB and 32MB block sizes, which are identical to the baseline benchmark access sizes. Using 5 access sizes increases the granularity of the storage system baseline performance profile over a two bin approach, as used by Intel [7], which in effect improves mapping accuracy.

Table 2.1: Binning of block access sizes for use in mapping.

Bin	lower bound	upper bound
4KB	0	≤8KB
32KB	>8KB	≤48KB
128KB	>48KB	≤256KB
1MB	>256KB	≤4MB
32MB	>4MB	∞

The mapping of the individual access sizes is performed by mapping accesses that are less or equal to 8KB to the 4KB bin. Accesses greater than 8KB but smaller or equal to 48KB to the 32KB bin. The 128KB, 1MB and 32MB bins are mapped as shown in Table 2.1.

With read and write accesses mapped to their corresponding bins, further analysis has to be done before a mapping between the workload and the baseline performances can be made. Recall that in the baseline analysis, applications were categorized as being either sequential or random. In general, such as clear distinction is not found to be the case in real workloads. These workloads exhibit both random and sequential access patterns and so to distinguish between them the concept of randomness (capturing the mix of random and sequential behaviour) is introduced.

The challenge is to determine a point on the randomness spectrum above which the workload disk access pattern will be defined to be mostly random and below which it will be defined to be mostly sequential. Choosing this point is a non-trivial task and will have an impact on the subsequent analysis presented here. A judicious choice would result from extensive empirical studies. However, inspiration can be taken from [45], where it states that two or more consecutive accesses that exceed a distance of 128 LBNs are considered to be random accesses. All accesses with a distance of less than 128 LBNs are therefore considered sequential. The 128 LBNs value thus partitions the randomness spectrum in two. This partition information is used to characterize the random and sequential nature of various phases of a workload. The relative proportion of the sum of the sequential phases, for both reads and writes, are then mapped to the sequential read or write baseline appropriately in the previously chosen bin and the relative proportion of the random phases, for both reads and writes, are mapped to the random read or write baseline appropriately, again in the previously chosen bin.

If the distance value for separating the sequential from the random accesses would be set to a higher value, more accesses would be considered sequential, resulting in more proposed configurations that perform better for sequential accesses. When choosing a lower value, more accesses would be considered random and configurations that perform better for random accesses would be more likely to be proposed. Therefore, a well chosen distance value can improve the mapping result and associate the accesses with the correct baselines. Testing for the most appropriate value is left to future work.

As all the different configurations are compared against the default baseline, the results are normalized. This allows for better comparison and for direct addition of the individual access types.

Using this relative proportions of the bin sizes, the performance of the workload under different baseline configurations can be calculated using the proposed Formula 2.1.

$$P_{throughput} = \sum_i A_i(p \times rr_i + (1-p) \times sr_i) + B_i(q \times rw_i + (1-q) \times sw_i) \tag{2.1}$$

where i takes in the following values $[4KB, 32KB, 128KB, 1MB, 32MB]$, and where

$p =$ the relative proportion of random reads,

$q =$ the relative proportion of random writes,

$A_i =$ the total amount of reads,

$B_i =$ the total amount of writes,

$sr =$ the performance metrics of the sequential read,

$sw =$ the performance metrics of the sequential writes,

$rr =$ the performance metrics of the random reads,

$rw =$ the performance metrics of the random writes.

$P_{throughput}$ represents the performance of the individual configurations relative to the default configuration. Results above 100 indicate a performance increase over the default configuration for each specific workload considered, while results below 100 indicate a performance decrease relative to the default configuration.

2.2.5 Verification

To verify the results calculated by the mapping algorithm, a approach similar to the baseline performance analysis is used. The workload is tested with 12 virtual machines running the workload simultaneously multiple times. The Ceph configurations that are being tested are the default, the lowest and the highest performing alternative configurations. Note that this does not assume a given relationship between the default and the alternative configurations. The results for this verification step will suggest changes in the performance characteristics that may result from applying these configurations to real workloads. This empirical comparison will be explored in Section 3.

In some cases the resolution of a benchmark result was too low to determine a change in performance. To address this, attempts were made to increase the resolution by using

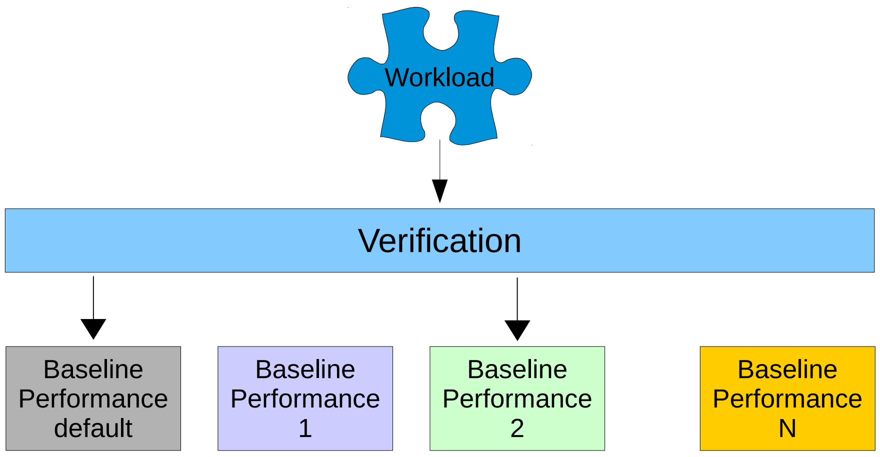


Figure 2.7: Verification of the predicted performance increasing configuration against the workload.

fewer virtual machines and by keeping those virtual machines equally distributed across all compute hosts. For those workloads that were not constrained by the performance of the storage backend, but rather were sensitive to hardware characteristics (CPU performance, memory capacity/speed, etc.), the alternative strategy of increasing the number of concurrent virtual machines while maintaining the homogeneous distribution was adopted.

2.3 Benchmarks

Standard benchmarks that read and write files of a fixed size are helpful in measuring the performance of a file system, but are very synthetic and limit the perspective of the evaluation. The relevance of the results for a specific workload is sometimes questionable, as is the meaning of the results. Tarasov *et al.* [46] discuss the specific levels of the typical file system benchmarks.

Generic file system benchmarks, such as Flexible IO Tester (fio), will be used for testing the performance of different file system configurations. In a later stage, configurations that show performance gains will be validated against real world workloads.

Workloads deployed on production and development OpenStack systems can be extracted from the OpenStack user surveys.

According to the OpenStack User Survey from November 2014 [47] the most common workload deployed on OpenStack production systems are web services with 57%, followed by databases (49%) and unspecified custom user workloads (47%). Other named workload types include quality assurance and test environments (40%), enterprise applications (37%), continuous integration (35%), management and monitoring (31%) and storage/backup (31%). Other workloads did not have a prevalence greater than

30%.

According to the 2015 version of the user survey [1], web infrastructure (35%), application development (34%) and user specific (33%) workloads, show practically an equal distribution. Content sharing workloads were deployed in 17% of the deployments.

Representative workloads for these categories are chosen to evaluate the impact of configuration changes on workloads deployed in production systems.

2.3.1 Synthetic Benchmarks

While synthetic benchmarks can be designed to test access patterns that are rarely appearing in real world workloads, they can provide a good general understanding of how a storage system performs when it is stressed with a specific access size and pattern combination. A single benchmark will typically not be able to reveal all characteristics of the storage system, in that case a combination of benchmarks is necessary to understand how a system performs.

2.3.1.1 Flexible IO Tester (fio)

fio [48] is an open source disk benchmark. It starts a number of threads or processes that perform a particular type of IO operation as specified by the user. Each thread uses globally defined parameters, but distinct parameters for each thread are supported. Supported types of IOs are sequential and random reads and writes. Combinations of these are also supported. Accesses can be defined using a broad selection of block sizes, which can be a expressed as a single size or a range.

fio has support for different IO engines, such as synchronous, asynchronous or cached accesses. Depending on the desired result, specific engines can be used to test the IO path. The IO queue depth can be varied and changing it to higher values can be used to test the performance differences between different IO schedulers. To reflect raw underlying performance (bypassing the cache), fio has support for direct IO and buffered accesses.

To be consistent with the choice made by Intel [7] and with the literature [46], fio was chosen as the benchmark against which all other workloads will be compared.

2.3.2 Web Services and Servers

Web services are software systems that are designed to support a machine-to-machine interaction over a network. It uses an interface that is described in a machine-processable format (specifically WSDL). Other systems interact with Web services using

SOAP-messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards [49].

A distributed system, consisting of a database server, an application server and a web server, is difficult to set up for benchmarking. Doing a performance analysis of a deployed web service can be done using an application such as ApacheBench.

The ApacheBench benchmark (ab) [50] [51] [52] tests the number of requests that an Apache webserver can processes when stressed with 100 concurrent connections (this is the default deployment configuration). It serves a static html page so that the cache is employed as part of the benchmark run. Consequently this will hide the performance of the file system and storage backend. As this benchmark is very CPU demanding, improvements on the storage system will typically not be apparent.

The Postmark benchmark [53] was developed by NetApp to reflect the workload characteristics of email, hotnews and e-commerce systems. Email servers typically create, read, write and delete small files only. Thus, the access pattern across the disk tends to be random. This pattern requires the processing of large amounts of metadata and falls outside the design parameters of most file systems. E-commerce platforms have developed significantly since the introduction of the Postmark benchmark and it is unclear if this benchmark is still relevant for these systems.

2.3.3 Databases

Databases are an essential component for many web services and applications. This relationship does not change when deployed to a cloud environment. To test the performance of the storage system under a database workload there are a couple of benchmarks that can be used.

The Transaction Processing Performance Council (TPC) [54] provides multiple standardised benchmarks to simulate the load of different transaction based systems [21] [55]. The workloads are continuously updated to include new workloads that reflect large transaction based systems, such as warehousing systems [56] [57].

The HammerDB benchmark [58] is an open source database benchmark that supports databases running on any operating system. The front end is available for Windows and Linux. It supports a great variety of databases (Oracle Database, Microsoft SQL Server, IBM DB2, TimesTen, MySQL, MariaDB, PostgreSQL, Postgres Plus Advanced Server, Greenplum, Redis, Amazon Aurora and Redshift and Trafodion SQL on Hadoop) and is widely used by researchers and industry to benchmark databases and hosts. Running HammerDB from the command line is not possible as it is designed to provide a graphical interface for displaying database performance benchmarks. Development of a command line based version was started with Autohammer, but is currently not actively being developed. The testing modes supported by HammerDB are a TPC-C [59] like and a TPC-H [60] like benchmark mode. Both standards are not implemented in full but are capable of predicting official TPC results quite accurately.

The SysBench benchmark [61] is a modular, cross-platform and multi-threaded benchmark tool. It can be used to evaluate system parameters that are important for a host running a database under intensive load [62] [63]. It is designed to determine the system performance without installing a database. The individual tests that SysBench supports are:

- file I/O performance
- scheduler performance
- memory allocation and transfer speed
- POSIX threads implementation performance
- database server performance.

During testing SysBench runs a specified number of threads that all execute requests in parallel. The actual workload depends on the specified test mode and user input. The system supports a time based, a request based or a combined testing limitation.

The database server test is designed to exercise a host like it would be by a production database. For that purpose, SysBench prepares a test database that is then subjected to different accesses. These accesses can be selected from a wide variety, such as simple `SELECT`, range `SUM()`, `UPDATE`, `INSERT` or `DELETE` statements.

The pgbench benchmark [64] is a benchmark tool that executes requests against a PostgreSQL database server [65]. The transaction profile is loosely related to the TPC-B [21] benchmark which is a stress test of the database. It measures how many concurrent transactions the database server can perform. Unlike other TPC profiles, it does not contain any users that might add a "think time" between the requests. If a transaction is finished it will spawn a new one immediately. This makes TPC-B a valid option on a system that might see simultaneous multiplexed transactions and the maximum throughput has to be determined. It can also be used in a scripted fashion, which is the reason it has been picked as the benchmark to simulate a database workload. For more information see Section 4.3.5.

2.3.4 Continuous Integration

Continuous integration is a software engineering process that continuously compiles the source code of an application to see if it compiles without errors. Each new version of the code is tested, sometimes dozens of times per day. For test-driven development unit tests are performed with each run to ensure the code complies to the tests. Furthermore metrics are reported indicating the code coverage of the tests.

Continuous integration platforms are available as hosted solutions, such as Travis CI [66], and as self hosted toolkits, such as Jenkins [67], Hudson [68] or CruiseControl [69].

kcbench [70] is benchmark for a compilation workload, measuring the time it takes to compile the Linux Kernel. Compilation, in general, is CPU bound and is mostly limited by the performance of the CPU, however, memory speed and disk performance may also impact performance. As the Linux Kernel is a very complex project, consisting of many thousands of source and header files, a substantial amount of disk accesses need to be performed. For more information see Section 4.3.4

Compilation times of other applications, such as Apache or ImageMagick are also often used as compilation workloads [46].

2.3.5 File Server

File servers are a commonly deployed application type for cloud services. The type of server might vary, but they are used to store and serve data from and to multiple clients. Implementations of traditional file servers can export a storage device over a network protocol, such as NFS or CIFS, an FTP server, or a file synchronisation system, such as Seafile [71], Nextcloud [72] or OwnCloud [73].

DBENCH [74] is a tool to generate I/O workloads that are typically seen on a file server. It can execute the workload using a local file system, NFS or CIFS shares, or iSCSI targets. It can be configured to simulate a specific number of clients, to determine the throughput the server is able to handle. For more information see Section 4.3.3.

2.3.6 Ceph Internal Tests

Ceph has a way to measure the performance of the cluster using internal tools. These tools can be used directly on the storage node or from a client that has the credentials to access the storage cluster.

Rados bench [75] is a benchmark that reads or writes objects to specific Ceph pools. The object size is variable, as is the number of concurrent connections. The access pattern can only be one of writing, sequential reading or random reading. As a prerequisite for a read benchmark, the cluster has to be filled with files. This has to be specified during the writing benchmark, as that benchmark normally deletes the written objects at the end of the test. The output of the benchmark is presented in Listing E.1.

It is also possible to write directly to an image file/object that is already stored in the Ceph cluster using `rbd bench-write` [76]. This is useful for testing the performance when writing additional data into an existing object. It mirrors the scenario when Ceph is used as the storage backend for a hypervisor for virtual machines, like QEMU/KVM. The limitation of this benchmark lies in its single ability to write to an image file. Performing read accesses is not supported.

The Ceph Benchmarking Tool (CBT) [77] emerged subsequent to the investigation described in this dissertation. This tool can be used to test different Ceph interfaces with different benchmarks. It can be used to execute a Rados bench test on the cluster; it can also be used to run fio (described in Section 2.3.1.1) through different Ceph interfaces. The `librbd` userland implementation is used by QEMU, which allows for an approximation of KVM/QEMU performance without deploying such a system. The `kvmrdbio` implementation tests the performance from a virtual machine using rados block devices and KVM, as used in Openstack deployments using Cinder for virtual machine block devices. This test requires the VM to be deployed before execution. The third implementation can be used to test an RBD volume that has been mapped to a block device using the `KRBD` kernel driver. This implementation is used when an application requires a block device but cannot be run in a virtual machine.

2.4 Tuning for Workloads

Ceph consists of many components and a huge number of parameters (as described in Section 1.2) possibly resulting in hundreds of millions of distinct configurations. From the description of the Ceph system given so far it can be seen that there are many degrees of freedom for choosing an optimal configuration. Here, optimality is considered in the context of tuning the system to best support a given workload.

The Ceph system is composed of very many components and parameters arranged in a hierarchical tree structure, depicted in Figure 2.8. Functional components can be associated with all, or part of each subtree. These functional components essentially form a partition of the Ceph system and embody subcomponents and parameters that can be chosen independently of the rest of the system. However, these partitions can not be considered as being isolated from the other components of the Ceph system. These other components essentially form an environment in which the functional component formed by the partition operates. In the remainder of this dissertation this is referred to as the Ceph environment. Likewise, the Ceph system itself is part of a bigger system and components of that system, such as hardware and operating system configurations or indeed the organizational requirements coming from a particular deployment, such as authentication and security policies, all form the greater Ceph environment. The environments of a functional component may indirectly constrain the performance of that component.

To improve a functional component with respect to a given workload, there are at least two alternative processes that can be considered. The first involves fixing both the Ceph environment and the greater Ceph environment and choosing the parameters of the subcomponents appropriately. A second approach could involve starting with a fixed configuration of the functional component and changing either the Ceph environment and or the greater Ceph environment to improve the performance of that functional component. The functional components capable of being configured by either of these methods include the pool, the monitor and the metadata server. This work concentrates predominantly on the pool in combination with approach two. This approach is thought to be more promising, and is studied here from an empirical perspective, since the myriad of constraints imposed by the environment on the pool, can be explored with view to relaxing as many as possible in the process of tuning it for a given workload.

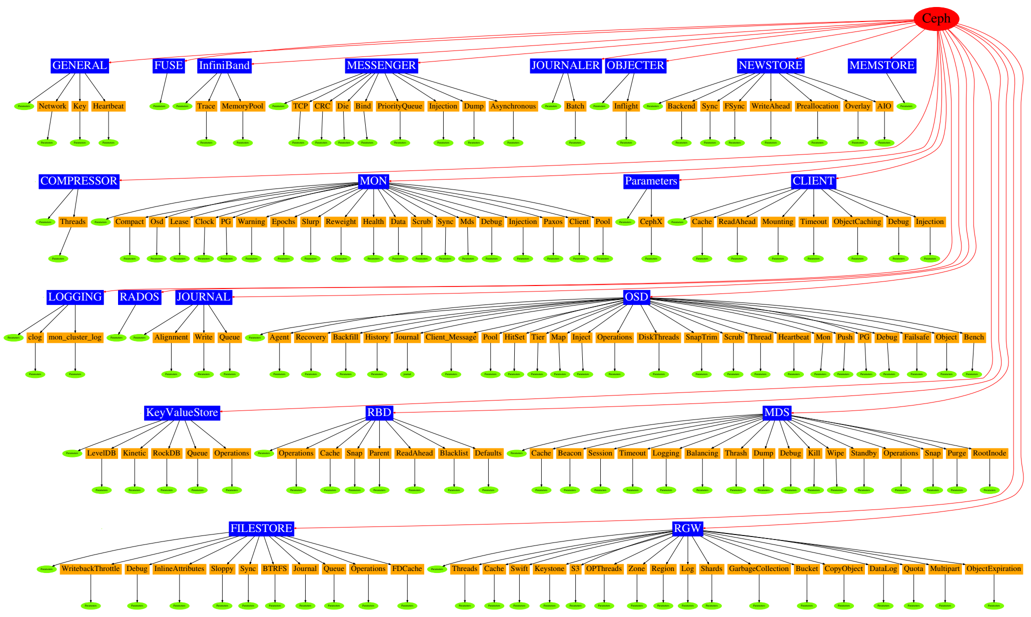


Figure 2.8: Ceph parameter hierarchy with **red** representing the Ceph root, **blue** components, **orange** categories and **green** the parameters. Note that parameters may be associated directly with a component or within a category of a component.

2.4.1 The Ceph Environment

Functional components, as describes above, constitute a partition of the Ceph system. All parameters outside of a particular partition constitute the environment of that partition. And, according to approach 2, these parameters will be changed in an effort to improve the performance of the functional component defined by that partition. The environment of the functional component representing pools is depicted in Figure 2.9.

This environment illustrates the many possibilities to adapt the Ceph environment so that a pool can optimally support a given workload. In a practical Ceph deployment many distinct pools may be created to support the needs of a structured organization. Consequently, all of these pools will share the same environment and changes to that environment will affect all pools simultaneously.

Since the average workload associated with each pool will typically differ from pool to pool, one Ceph environment configuration may not best fit all pools and associated workloads. Thus, the improvement process becomes more challenging. Either 1) priority is given to a particular pool, when the environment is being configured, resulting in a potential degradation of performance of the others, or 2) the environment is configured so as to balance the needs of all pools simultaneously. The latter approach will most likely result in no pool being optimally configured nor see an overall improvement. This dissertation focusses on the former approach and leaves the latter to further investigation.

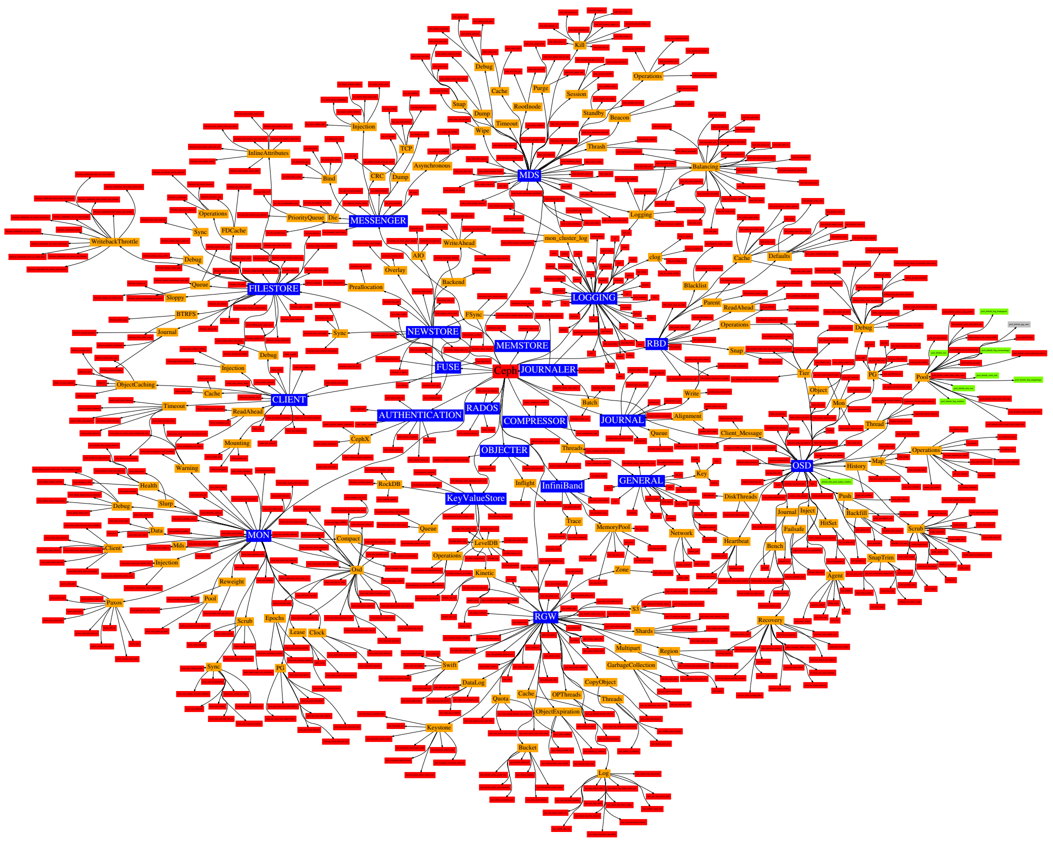


Figure 2.9: Tunable Ceph environment parameters (highlighted in **red**) when optimizing a functional component (partition highlighted in green). In this instance the functional component is a Ceph pool.

The Ceph components and their associated parameters constituting the entire Ceph system are depicted in Figure 2.8. When a system is initialized by a system administrator a number of components determining how and where a Ceph cluster will operate need to be appropriately configured. These include Logging, Authentication (CephX) and the General component, which includes parameters for configuring public and private networks, cluster UUID and the cluster heartbeat. Logging within the system can be set to different levels, which might have to be increased to debug a malfunctioning component when default logging levels are insufficient. Depending on the constraints from the greater Ceph environment, authentication might be tightened or loosened to meet the required levels of security.

Of the 870 tunable parameters of Ceph, 182 parameters affect the behaviour of the OSDs, such as the default number of placement groups used for a pool and the number of threads per OSD, as depicted in Figure 1.4. In addition, 121 parameters influence the behaviour of the MONs, like the update frequency of the cluster map and the ratio for marking OSDs as full. For the MDSs, there are 105 parameters, while 106 parameters affect the RADOS gateway (RGW).

The Ceph Filestore, the component that stores the data on the OSD, is configurable by 92 parameters. The new filestore, that has become Bluestore, has been in previous versions of Ceph, where it was called Newstore, has 29 parameters in Ceph version 0.94. The way in which data is written to the journal is configurable, as are the RADOS block devices (RBD), used e.g. by virtual machines, and the CephFS. Other components, such as the Client, Messenger, Compressor, Objecter and Memstore are also configurable and can have an effect on cluster behaviour and performance.

The effect of changing individual Ceph environment parameters on pool performance is shown in Section 3.4.

2.4.2 Pools

Internal parameters of the functional component of a pool are depicted in Figure 2.10. These parameters can be used in conjunction with approach 1 to initially configure the pool. These parameters are listed in Table 2.2. While some parameters are common for all deployed pools, others pertain only to tiered pools (described in detail in Section 2.4.3). When optimizing the parameters of a functional component, such as a pool, the Ceph environment and greater environment parameters are fixed.

The parameters of the functional component pool can can be divided into different categories. The first category is security related. It includes parameters to ensure safe handling of pools. They include `nodelete`, `nopgchange` and `nosizechange`. These parameters do not affect performance and so can be selected with impunity.

The second category relate to replication count and placement, which directly influences reliability and stability. These parameters include `size`, `min_size`, `pgp_num`,

Table 2.2: Ceph pool options.

size	Sets the number of replicas for objects in the pool. This only works on replicated pools.
min_size	Sets the minimum number of replicas required in the cluster for I/O. This can be used for allowing or denying access to degraded objects.
crash_replay_interval	Amount of time in seconds to allow clients to replay acknowledged, but uncommitted requests.
pgp_num	The effective number of placement groups to use when calculating data placement.
crush_ruleset	The ruleset to use for mapping object placement in the cluster.
hashpool	Set or Unset HASHPSPOOL flag on a given pool. When true HASHPSPOOL is hashing the pg seed and pool together instead of adding to create a more random distribution of data.
nodelete	Set or Unset NODELETE flag on a given pool. This prevents the pool to be deleted by accident or intent. This was added as a safety feature.
nopgchange	Set or Unset NOPGCHANGE flag on a given pool. This prevents the pools placement group count to be changed.
nosizechange	Set/Unset NOSIZECHANGE flag on a given pool. This prevents the pools replication count to be changed.
hit_set_type	Enables hit set tracking for cache pools. This will enable a Bloom filter [78] to reduce the memory footprint for the hashtable.
hit_set_count	The number of hit sets to store for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon. The value has to be 1 as the agent currently does not support values greater 1.
hit_set_period	The duration of a hit set period in seconds for cache pools. The higher the number, the more RAM consumed by the `ceph-osd` daemon.
hit_set_fpp	The false positive probability for the bloom hit set type.
cache_target_dirty_ratio	The percentage of the cache pool containing modified (dirty) objects before the cache tiering agent will flush them to the backing storage pool.
cache_target_full_ratio	The percentage of the cache pool containing unmodified (clean) objects before the cache tiering agent will evict them from the cache pool.
target_max_bytes	Ceph will begin flushing or evicting objects when the max_bytes threshold is triggered.
target_max_objects	Ceph will begin flushing or evicting objects when the max_objects threshold is triggered.
cache_min_flush_age	The time (in seconds) before the cache tiering agent will flush an object from the cache pool to the storage pool.
cache_min_evict_age	The time (in seconds) before the cache tiering agent will evict an object from the cache pool.

`crush_ruleset`, `hashpspool` and `crash_replay_interval`. Changing parameters in this category directly influences performance.

The third category relates to caching in a tiered system. Parameters in this category change the movement of objects between the hot and the cold storage and change the caching algorithm. These parameters include `hit_set_period`, `hit_set_fpp`, `cache_target_dirty_ratio`, `cache_target_full_ratio`, `target_max_bytes`, `target_max_objects`, `cache_min_flush_age` and `cache_min_evict_age` and are only used in conjunction with a tiered pool.

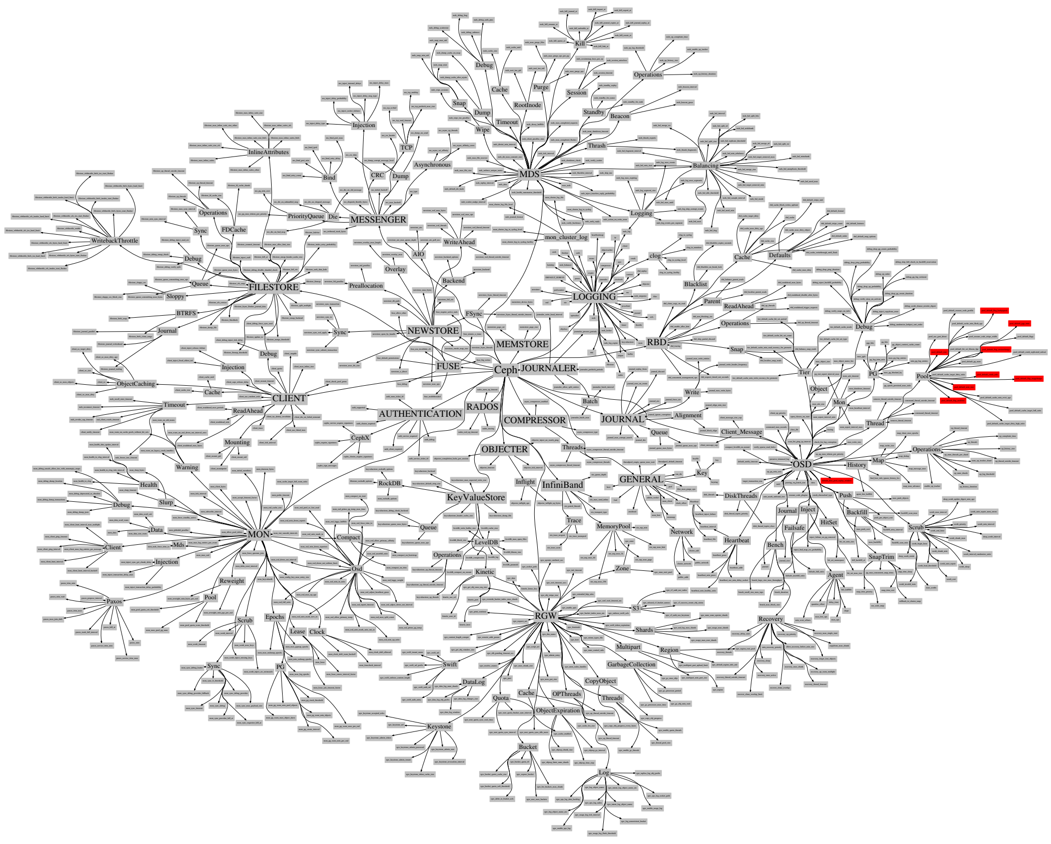


Figure 2.10: Ceph parameters (highlighted in **red**) directly affecting the pools.

While most parameters directly applicable to a pool are performance related, not all of them are active in every system. Furthermore, many of the parameter values arise from the environment and the requirements of the workload that is to be executed. For critical applications a high number of replicas might be desired to ensure accessibility and integrity. While a high number of replicas enhance safety, write performance will be reduced since multiple copies of the data have to be written before the write operation is acknowledged. In systems with less critical workloads, a lower replication count can help to improve write performance. Setting the correct number of placement groups can also assist in improving performance, since the data is spread across a larger number of OSDs, allowing scalability effects to improve performance. Changing these parameters after the pool is created is a time consuming task. Increasing the replication count would require the system to create extra copies for each object in the pool. Changing the data distribution through the placement groups or the `crush_ruleset` would initiate a complete redistribution of the data throughout the storage cluster. Both of these operations can consume a large amount of time (potentially many hours or days) where the system is operating with degraded performance.

2.4.3 Pools with Tiering

Tiering is a mechanism to hierarchically organizing the storage system to maximise performance by minimizing latency and maximising throughput. Typicall storage devices with different characteristics are combined to create the best price/performance ratio. Tiered systems are numbered from 1 to n, where Tier 1 represents the fastest access/best performance Tier. Historically a Tier 1 storage system used fast spinning disks (15k rpm) for low latency and high throughput. Disadvantages of these drives were high power consumption, low capacity and price. Therefore, they were only used as a fast top-level cache for specific workloads, such as databases. Tier 2 was typically populated by disks with a rotational speed of 10k rpm. These offered slightly slower access times, but these were cheaper and had higher capacities. Tier 3 typically consisted of SAS or SATA disks rotating at 7.2k rpm. These drives were available in even greater sizes with lower power consumption and a lower price. They were typically used as cold storage and for sequential accesses associated with applications such as video streaming. In some cases deployments also used a fourth Tier, which employed tape drives as long term archival storage. Tiering represents an active system in which data automatically migrates in both directions through the various levels to keep the most used data (hot data) in the fastest Tiers and redundant copies or infrequently used data on the slower Tiers.

With the advent of solid state disks (SSD), the components in a tiered system have changed. SSDs have replaced the 15k SAS drives in Tier 1, offer a higher throughput and reduced access time. SSDs are based on flash chips and can only sustain a certain number of writes before failing (flash cell deterioration). This write amplification count varies with the SSD NAND [79]) chip type.

- Single Level Chips (SLC) offer the highest write amplification. They store one Bit per cell. The drawback of these chips is the high manufacturing cost and limited capacity per chip.
- Multi Level (MLC) and Triple Level (TLC) Chips are the mostly used in consumer level devices, but can also be found in enterprise class SSDs. They store two and three bits per cell. These chips are cheaper to produce and offer larger capacities with a penalty of a reduced write amplification count. One way to counter this is to add extra chips as spare capacity that is used by the controller to level the wear across the available NAND cells. Where consumer drives have around 8-9 % of the total flash capacity set aside for over-provisioning, enterprise drives can have up to 25%. Using over-provisioning helps to increase the lifetime of the drive without having to use expensive SLC NAND at the cost of adding extra chips.
- A NAND type that tries to combine the best of both is Enterprise MLC (eMLC) that achieves the capacity of MLC with write amplification counts in between SLC

and MLC. Intel is using a similar technology in their datacentre SSDs, where it is known as HET MLC [80] (High Endurance Technology).

A tiered storage system using SSDs and conventional hard drives is often found to use the following schema:

- Tier 1 is usually served by SSDs using SLC, eMLC or HET MLC NAND to cope with the constant writes to the flash cells.
- Tier 2 is served by MLC based SSDs for read intensive applications to reduce the cost of the installation.
- In some cases it might still be useful to use fast spinning disks (10k rpm, 15k rpm) as an extra Tier or go straight to the 7.2k rpm SAS disks in Tier 3.
- Depending on the use case, it might also be useful to add a Tier 4 with extra high capacity disks (6-10 TB) as cold storage, as some of these drives use shingled recording [81], which increases capacity at the cost of latency; instead of updating a block, the whole track has to be re-written. This makes them unsuitable for frequently accessed data due to the heavy penalty associated with altering content.

Some manufacturers combine Tiers 1 and 2 and do not differentiate between read and write intensive disks, when the environment does not exceed the manufacturers drive writes per day (DWPD). A DWPD corresponds to writing the total capacity of the disk once per day. Depending on the drive, this value can exceed 3 without adversely affecting the expected lifetime of the disk.

As SSDs get faster, the storage interface consisting of SAS 6Gbit/s and SATA 3 do not provide the necessary bandwidth and so become the bottleneck to performance improvements. Therefore, the industry has added SAS 12Gbit/s and SATA Express. Furthermore, the manufacturers use PCIe directly. This allows for a bandwidth of up to 1GB/s per lane. In the case of the Intel DC P3700, an x4 interface is used to achieve a sequential transfer speed of up to 2800 MB/s [82]. Using PCIe also comes with the benefit of reduced latency. To reduce the latency even further the NVMe specification [83] (NVM Express or Non-Volatile Memory Host Controller Interface Specification) has been created. It bypasses parts of the storage stack to reduce the latency for data access (see Figure 2.11). When these PCIe SSDs are used in a tiered storage system, they are sometimes called Tier 0.

In Ceph, Tiers have the same functionality, improving performance across Ceph interfaces (RBD, RADOS, CephFS). In Ceph Tiers can be configured to create a cache pool servicing servicing slower pools associated with slower storage media. This type of cache has two different modes of operation:

Write-back Mode In write-back mode, a client will write data directly to the fast

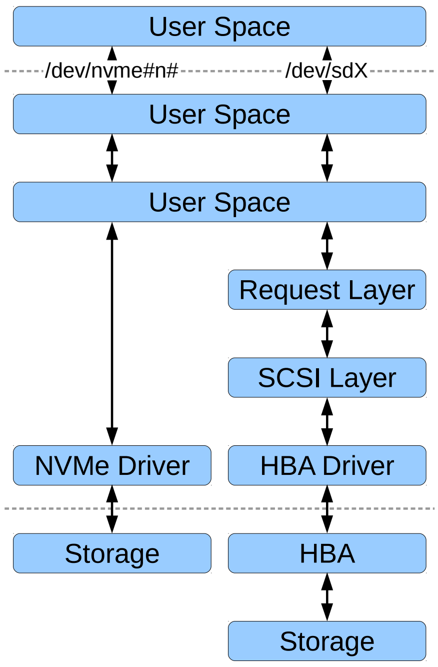


Figure 2.11: Schematic comparison between the NVMe and the SCSI storage stack within an OS.

cache Tier and will receive an acknowledgement when the request has been finished. Over time, the data will be sent to the storage Tier and potentially flushed from the cache Tier. When a client requests data that does not reside within the cache, the data is transferred first to the cache Tier and then served to the client. This mode is best used for data that is changeable, such transactional data.

Read-only Mode In read-only mode, the cache will only be used for read accesses. Write accesses will be sent directly to the storage Tier. This mode of operation is best used for immutable data, such as images and videos for webservices, DNA sequences or radiology images. Since the consistency checking with the Ceph Tiers is weak, read-only mode should not be used for mutable data.

When the pool functional component is extended to include Tiering, the number of parameters associated with that component increase and consequently the number of parameters in the environment of that component decrease (see Figure 2.12). Thus, the degrees of freedom for improving the pool using Approach 2 are reduced.

2.4.4 Heterogeneous Pools/Greater Ceph Environment

In general, all the functional components of Ceph can see all the elements of the greater Ceph environment and have the same view of those elements. However, it is possible to configure a functional component, such as a pool, so that its view of the greater Ceph environment differs from the views of some or all of the other pools. In this

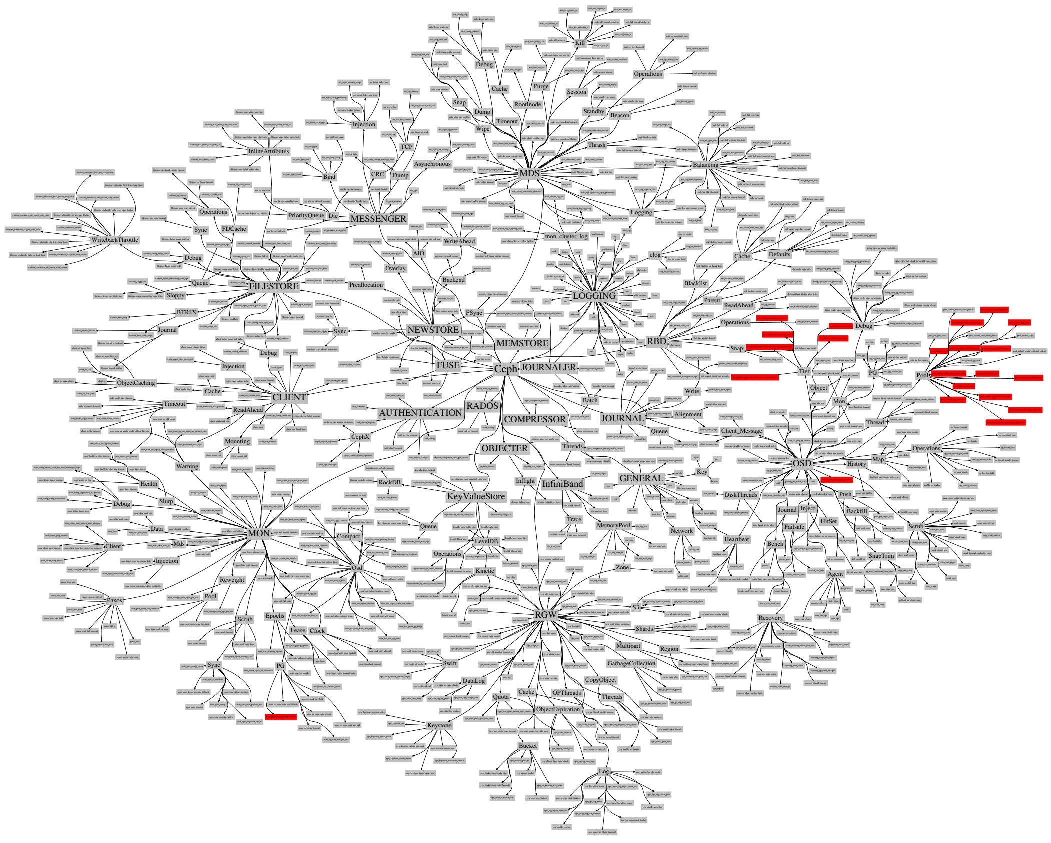


Figure 2.12: Ceph parameters (highlighted in **red**) directly affecting the pools with tiering.

way it is possible to create heterogeneous pools, each of which behaves differently as determined by specific components in the greater Ceph environment. It follows that the heterogeneous pools may be improved independently of each other via the greater Ceph environment, in contrast to improving functional components via the Ceph environment where all pools are simultaneously effected by changes to that environment.

The greater Ceph environment includes components between Ceph and the physical underlying hardware, such as the file system (deployed on the OSD), the IO scheduler (used by the operating system kernel to dispatch I/Os from the application to the physical layer) and the underlying hardware itself. The greater Ceph environment is an elemental component to a Ceph deployment, but its components can be modified to suit the specific deployment and workload.

For Ceph these components are, for the most part, transparent. The disk scheduler of the storage device hosting the OSD is not known within Ceph. The file system deployed on the OSD, on the other hand, is important since the device is mounted with different mounting options and is potentially subjected to different limitations of that file system, such as the limited number of files supported by ext4.

Showing how the behaviour of heterogeneous pools can be changed via the greater Ceph environment is explored in Section 3.5.

2.4.5 Multi Cluster System

If Approach 2, for improving a pool via the Ceph environment, is found to be ineffective, possibly due to a number of pools in the environment requiring diverse tuning, the process of storage cluster partitioning may offer a viable alternative. By separating the storage cluster into a number of distinct Ceph environments, local improvement within each may deliver a better solution.

Such a multi-cluster solution allows for pool improvement using Approach 2 without being constrained by requirements of pools within the same cluster. The drawback of such a multi-cluster solution is a potential reduction in overall performance, due to the reduced number of OSDs within each cluster. Another drawback of using multiple pools is the inability of using Tiering between the clusters, since Tiering is only supported between pools within the same cluster.

A Ceph multi-cluster solution can be achieved in one of two ways. The first is to run multiple OSD daemons on the host, belonging to different clusters. The other is to run the Ceph cluster components in containers (e.g., LXC, Docker) or virtual machines, as shown in Figure 2.13, to achieve the required partitioning.

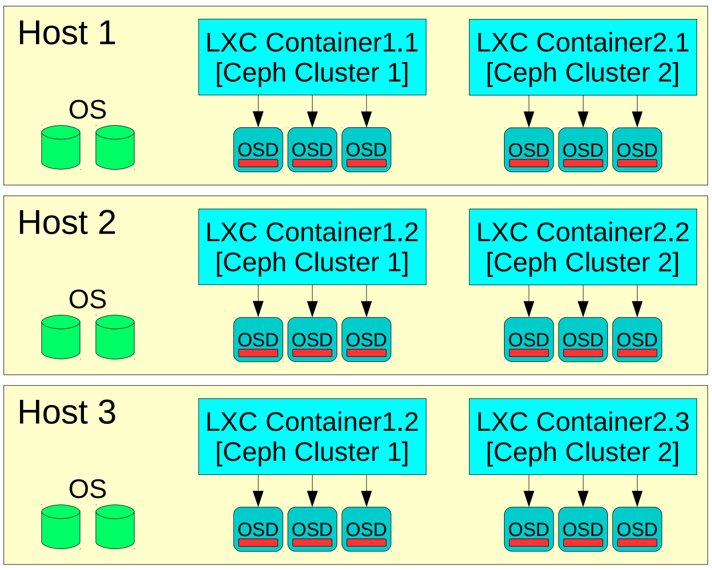

Host 1
OS
LXC Container1.1 [Ceph Cluster 1]
LXC Container2.1 [Ceph Cluster 2]
OSD
Host 2
OS
LXC Container1.2 [Ceph Cluster 1]
LXC Container2.2 [Ceph Cluster 2]
Host 3
OS
LXC Container1.2 [Ceph Cluster 1]
LXC Container2.3 [Ceph Cluster 2]

Figure 2.13: Ceph multi cluster using LXC containers using 3 nodes.

Chapter 3

Empirical Studies

This chapter describes the creation of a testbed hosting an OpenStack and Ceph deployment. This testbed is then used to explore the improvement of Ceph pools using the procedures and methods described in Chapter 3. Similar studies could be applied to improve other functional components following the experimental methodology described here but these are not pursued in this dissertation. In Section 3.1 the elements of the greater Ceph environment are described. In Section 3.2 a benchmark system is set up. It consists of a benchmarking server that sends out benchmark tasks to connected clients, which, in this instance, are deployed on the virtual machines used for the baseline evaluation. The initialization of the virtual machines and the installation of the benchmark client are described in the same section. The parameters of the Ceph environment whose values are impacted by the greater Ceph environment are presented in Section 3.3. Furthermore, a mechanism for sweeping through the parameters of the Ceph environment to identify and to set those parameters that could potentially improve the performance of a pool is described. Subsequently, in Section 3.4 the impact of changing each of those parameters in isolation, on the performance of the pool, is examined.

3.1 Testbed

The testbed used to carry out the empirical investigation is composed of a Ceph deployment and a collection of hardware and system software components constituting the greater Ceph environment. The hardware components include physical servers, physical storage systems and the network infrastructure. The system software components, treated here in Sections 3.1.4 and 3.2, includes the operating system, the OpenStack deployment and the deployment mechanism. An overview of the actual hardware configuration used in the testbed is shown in Figure 3.1.

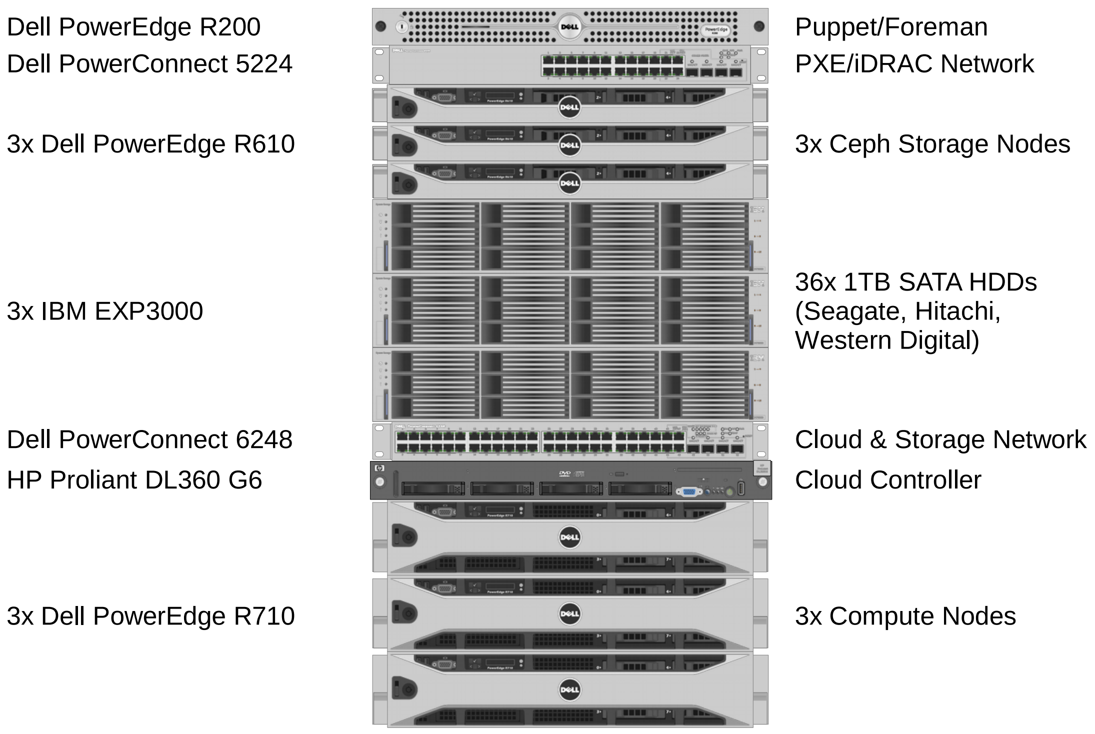


Figure 3.1: Hardware used in the testbed.

3.1.1 Physical Servers

The hardware used in the testbed consist of three different types of physical servers with various specifications (see Table 3.1). From these specifications appropriate hardware is chosen to implement the components of the system effectively. Thus,

- The Dell PowerEdge R200 [84] is used to deploy the operating system and the installed software using Puppet and Foreman (see Section 3.1.4.2). Typically this is done via the internet, thus a separation between the external network and the internal network is required. One is connected to the external network, where the operating system and software packages are stored, the other is connected to the internal network, with the target servers for the software deployment. This separation also allows the Dell PowerEdge R200 to run a DHCP server to generate internal IP addresses.

- The HP ProLiant DL360 G6 [85] server is used as an OpenStack controller node that runs all OpenStack services except Nova compute. This requires separate networks to handle OpenStack internal communication and the external services, such as the OpenStack dashboard Horizon. Therefore, the network service requires an extra network port that can be used as a network bridge to assign an external IP address to virtual machines without connecting the compute nodes to the public network.

- The Dell PowerEdge R610 [86] servers are used for the Ceph storage system. These servers are equipped with an LSI Logic SAS3444E [87] 3GBit/s 4-Port SAS HBA. They are connected via an external mini SAS connector (SFF8088) to the IBM SAN expansion trays, as described in Section 3.1.2. The memory capacity of 16 GB is necessary, since each Ceph OSD requires up to 1 GB of memory when under heavy load, such as a rebuild process. These servers only offer two PCIe expansion slots. One is used by the SAS adapter and the other is used for the Intel ET network card. The system is thus limited to eight 1 GBit/s network ports (4x onboard Broadcom Corporation NetXtreme II BCM5709 [88], 1x Intel Gigabit ET Quad Port Server Adapter [89]).
- The Dell PowerEdge R710 [90] servers are used for the compute service of OpenStack. With 12 physical cores and 32 GB of memory, they are capable of hosting many concurrent virtual machines. The total number of network ports sums up to 12 (4x onboard Broadcom Corporation NetXtreme II BCM5709, 2x Intel Gigabit ET Quad Port Server Adapter). Because all the virtual machines will run directly off the external storage, the internal disks, in this case the 32 GB SD cards in each server, are not a performance bottleneck.

Table 3.1: Physical server specifications and their roles in the testbed.

Model	Dell PowerEdge R200	Dell PowerEdge R610	Dell PowerEdge R710	HP ProLiant DL360 G6
CPU Model	1x Intel E4700	1x Intel Xeon E5603	2x Intel Xeon E5645	2x Intel Xeon E5504
Cores per CPU	dual	quad	hexa	quad
CPU Clock Rate	2.6 GHz	1.6 GHz	2.4 GHz	2.0 GHz
CPU Turbo	NA	NA GHz	2.67 GHz	NA
Memory	4 GB	16 GB	32 GB	16 GB
Storage	2x 2TB RAID-1	32GB SD card	32GB SD card	4x 300GB RAID-5
NIC	2x 1GBit/s	up to 12x 1GBit/s	12x 1GBit/s	8x 1GBit/s
Virtualization	X	Y	Y	Y
Role	Puppet-master, Foreman	Network, Storage	Compute	Controller

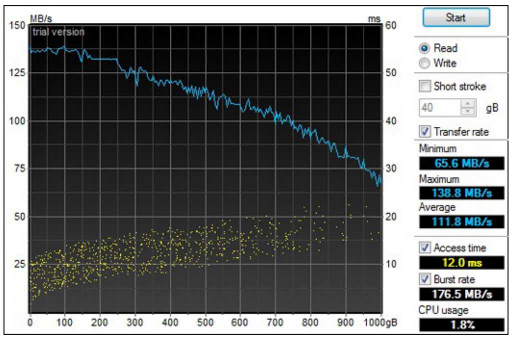
3.1.2 Storage System

Since the Dell R610 servers have no capacity to accommodate hard drives directly in the chassis, they have to be attached externally. Each server is connected via an LSI SAS3444E [87] SAS controller to an IBM EXP3000 expansion tray, populated with

12 hard drives, resulting in a total drive count of 36. Each storage tray is populated with 4 Western Digital RE4 1 TB and a mixture of 8 Hitachi and Seagate 1TB drives. Only the Western Digital RE4 drives were used in the pool improvement experiments. The drive specifications are presented in Table 3.2. Detailed transfer diagrams of the Western Digital RE4 drives are presented in Figure 3.2a and 3.2b.

Table 3.2: Specifications of used harddisks.

Manufacturer	Hitachi	Seagate	Western Digital
Name	Ultrastar A7K1000 [91]	Barracuda ES.2 [92]	RE4 [93]
Model	HUA721010-KLA330	ST31000-340NS	WD1003-FBYX
Capacity	1 TB	1 TB	1 TB
Rotational Speed (rpm)	7200	7200	7200
Cache	32 MB	32 MB	64 MB
Seek time (ms)	8.2 (read, typical)	8.5 (average read), 9.5 (average write)	12.0 (read), 4.5 (write) see Figure 3.2a and 3.2b
Sustained transfer rate (MB/s)	85 - 42	102 (max)	128 (max)
Interface	SATA 3 Gbps	SATA 3 Gbps	SATA 3 Gbps
Installed drives	9	15	12



[✓ Read] Write

☐ Short stroke 40 gB

☑ Transfer rate

Minimum 65.6 MB/s

Maximum 138.8 MB/s

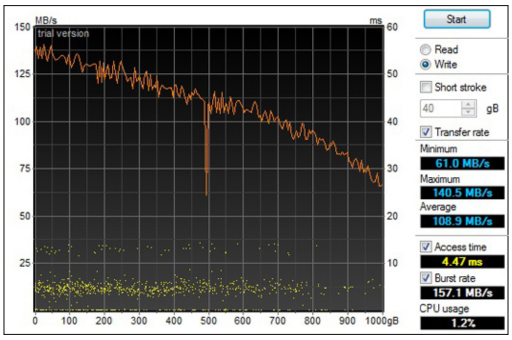
Average 111.8 MB/s

☑ Access time 12.0 ms

☑ Burst rate 176.5 MB/s

CPU usage 1.8%

(a) read



Read [✓ Write]

☐ Short stroke 40 gB

☑ Transfer rate

Minimum 61.0 MB/s

Maximum 140.5 MB/s

Average 108.9 MB/s

☑ Access time 4.47 ms

☑ Burst rate 157.1 MB/s

CPU usage 1.2%

(b) write

Figure 3.2: Transfer diagrams for Western Digital RE4 1TB (WD1003-FBYX) with access time measurements.

3.1.3 Network

The networking architecture for the testbed is quite complex and requires separate networks for different services, as depicted in Figure 3.3. In addition to the external network, the need for five additional separate networks has been identified. These are explained in detail in the following subsections. These are derived from reachability,

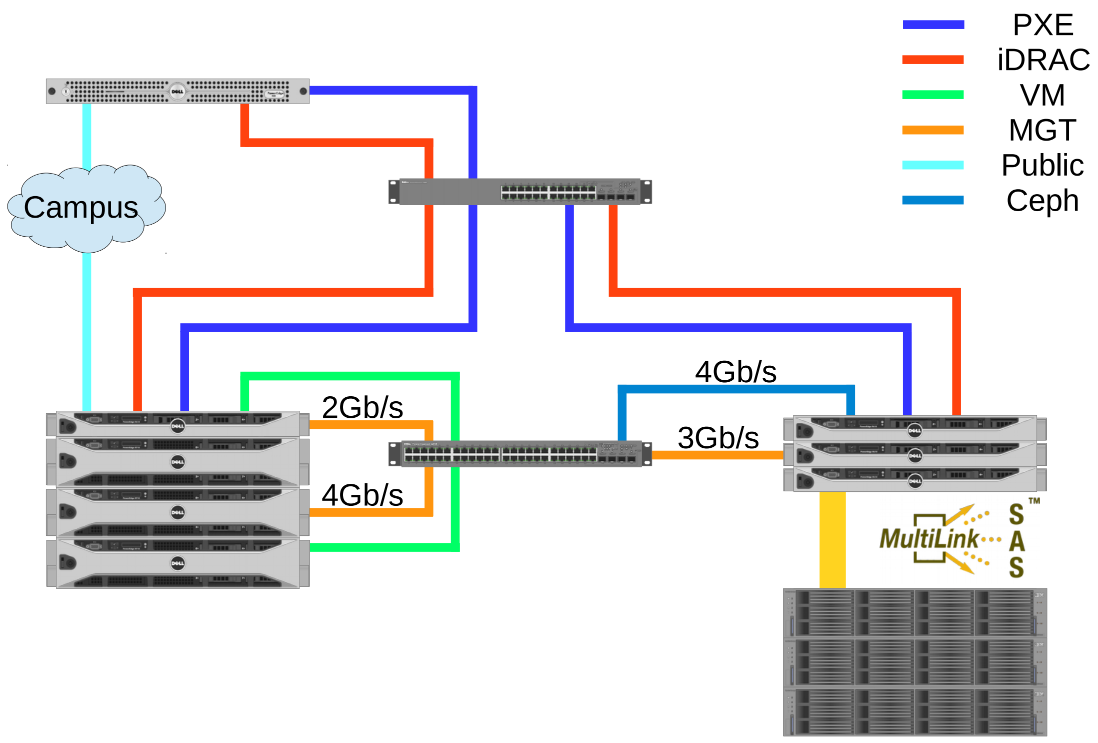


Figure 3.3: Testbed network architecture consisting of 5 separate networks and the direct connection between the storage nodes and storage trays [95].

isolation and bandwidth requirements. Some of these networks use bonded network interfaces to increase the capacity of the network links. The IEEE 802.3ad [94] protocol is used to achieve NIC bonding. The setup of the network bonds on the servers is achieved through a Puppet manifest and `ifenslave`. For increased network performance, the Maximum Transmission Unit (MTU) has been increased from 1492 to 9000 bytes on the servers and to 9216 on the switches.

3.1.3.1 External Network

The external network is the connection to the wider college network. As this is exposed to the campus and is limited to very few ports to the research lab specific VLAN, only a small number of hosts can be attached to it. Therefore, planning is required to ensure that only certain testbed hosts are exposed to the public network, as necessary.

Thus, only two servers in the testbed are attached to the external network:

- The deployment server, Phantomias, that also acts as the a proxy server and gateway to the internet.
- The cloud controller, that needs two ports on the public network for the OpenStack dashboard and network services, such as an unbound network interface for assigning floating IPs on the public network to virtual machines.

A single external network node is a single point of failure, however, it provides required network isolation and security.

3.1.3.2 Deployment Network

All servers used in the testbed are attached to the deployment network. The deployment network is used to manage the machines over the out-of-band management controller iDRAC that is integrated into the Dell PowerEdge servers. This interface allows for monitoring and restarting of the machines remotely. Furthermore, it can pass the video output and keyboard controls to a virtual console to interact with the machine without physical interaction.

This network uses the Preboot Execution Environment (PXE) to manage the operating system installation on individual nodes. The deployment server, Phantomias, runs a DHCP server to assign IP addresses and a Trivial File Transfer Protocol (TFTP) server providing the netboot images for installing the operating system or booting the installed OS.

Furthermore, this network connects to the Internet through the Proxy server running on Phantomias to download packages. The bandwidth requirements on this network are low and 1 Gigabit links are sufficient, since the connection to the external network is the limiting factor.

3.1.3.3 Storage Network

The storage network is an internal network between the Ceph storage nodes. A separate storage network, to separate the internal replication network from the public network is recommended, since replication tasks require high throughput. In this deployment, bonded network interfaces are used to provide enough bandwidth to handle the replication load (four interfaces on each server). This requires switches that are IEEE 802.3ad capable, which allows link aggregation with multiple ports. The bandwidth of the bonded interfaces is shown in Table 3.3.

3.1.3.4 Management Network

The management network is used for all communications between the OpenStack services. All servers, with the exception of Phantomias, are attached. As OpenStack is capable of using a CEPH storage cluster directly for its storage services (Glance, Cinder, Swift), the network also connects to the storage node's public interfaces. The OpenStack storage services only manage the access to the storage cluster but do not serve data to the compute nodes. The compute nodes connect directly to the storage cluster, which leads to large bandwidth requirements on the network, since all storage IO between the storage cluster and the compute nodes pass through this network. The controller only performs intensive network operations when a volume is created from an image. This task requires the controller to download the image, convert it to a raw format and upload it to the volume storage pool.

In this deployment, bonded network interfaces provide sufficient bandwidth to handle the replication load. This requires IEEE 802.3ad capable switches to support link aggregation with multiple ports. The compute nodes and the storage nodes each use three interfaces and the controller two more to provide the required bandwidth. The measured bandwidth on the different servers is shown in Table 3.3.

3.1.3.5 VM Internal Network

The VM internal network enables VM communication between hosts. This allows the separation of the VM communication from the other communications within the cloud system. Furthermore, it is used to create GRE tunnels (Generic Routing Encapsulation) between the VM and the egress point on the controller when the VM is assigned a floating IP. The bandwidth requirements in this testbed are expected to be low, and therefore Gigabit connectivity should be sufficient to allow all targeted workloads. The bandwidth on the links is shown in Table 3.3.

Table 3.3: Measured (`iperf`) network bandwidth of the different networks.

Network	**Storage**	**Management**		**VM Internal**	**Deployment**
Bonded Ports	4	2	3	1	1
Bandwidth	3.08 Gb/s	1.96 Gb/s	2.50 Gb/s	935 Mb/s	935 Mb/s

3.1.3.6 Network Setup Choices

Using four bonded interfaces for the storage network and three bonded interfaces for the management network on the storage nodes allows for higher throughput from/to the clients/compute hosts, since data is read directly from the OSDs. At the same time, this ratio limits the cluster write speed, because of the data replication between nodes. Data accesses are, in general, more read than write intensive, thus supplying enough bandwidth to clients is more important than overprovisioning the replication network.

3.1.3.7 Network Hardware

The network hardware consists of a mixture of Broadcom BCM5709c NetXtreme II Gigabit Ethernet (onboard PCIe x4) [88] and Intel Gigabit ET Quad Port Server Adapter (PCIe v2.0 x4) [89] network cards. They are attached to a Dell PowerConnect 5224 [96] and a Dell PowerConnect 6248 [97] Gigabit Ethernet switch through CAT6 Ethernet cables. The switch performance details are presented in Table 3.4.

Table 3.4: Network switch specifications.

Model	Dell PowerConnect 5224	Dell PowerConnect 6248
Ports	24 10/100/1000BASE-T, 4 SFP combo ports	48 10/100/1000BASE-T, 4 SFP combo ports
Switching Capacity	48.0 Gbps	184 Gbps
Forwarding Rate	35.6 Mpps	131 Mpps

3.1.4 Rollout

Installing the operating systems and the different types of software, configuring the software and systems is a labour intensive task, when replicated over many identical machines. Configuration management tools, such as Puppet and Chef, can make this task much easier. A configuration for these systems is captured in source code, hence the term Configuration as Code (CaC) or Infrastructure as Code (IaC). Configuration management tools use a machine-processable definition file rather than a hardware configuration. There are currently three different approaches for configuration management: declarative (functional), imperative (procedural) and intelligent (environment aware). Each of these approaches handles the configuration in a different way [98]:

- The declarative approach focuses on what the eventual target configuration should be.
- The imperative approach focuses on how the infrastructure is explicitly changed to meet the configuration target.
- The intelligent approach focuses on why the configuration should be a certain way in consideration of all the co-relationships and co-dependencies of multiple applications running on the same infrastructure.

Since configurations are code they can be tested for certain errors using static analysis tools, such as `puppet-lint` or `foodcritic`. Configurations can be applied in a repeatable fashion, which allows the deployment of many machines with the same configuration script. This might not seem significant when looking at a small collection of machines, but when used in an environment where machines are redeployed on a regular basis or to a large number of hosts, it is worth the effort.

In the testbed, the configuration management tool allows for continuous deployment of software to the storage nodes and the cloud system.

3.1.4.1 Operating System

The Ubuntu operating system (version 14.04 LTS) is deployed on all nodes of the testbed. Ubuntu was chosen since it is the reference platform for OpenStack.

To allow Ubuntu users to use newer Kernel versions, Ubuntu has the LTSEnablementStack [99]. This gives access to Kernel versions of the non LTS versions of Ubuntu without upgrading the whole installation to a non LTS version. The command for installing the enablement stack is shown in Listing E.2.

Using the 15.04 (Vivid) enablement stack in this testbed is particularly important, as there have been many changes to the code of the BTRFS and XFS file system that improve stability and reliability [100]. These are core components of the testing, therefore it is crucial to have these improvements installed to prevent erroneous conclusions.

3.1.4.2 Puppet and Foreman

Puppet [101] is a configuration management and service deployment system inspired by CFEngine that has been in development since 2001. Puppet configurations are stored as *manifests* that are centrally managed by a *Puppetmaster* server. Puppet is implemented in Ruby, however, some platforms, such as Android, are not compatible. Puppet is based on a client-server model and is capable of scaling very well. One server managing over 2000 clients is realistic. Puppet can be deployed on, and manage, both virtual and physical machines. In the latter case, it can install the hosts operating system and the necessary packages after automatically connecting to the Puppetmaster. For cloud environments, Puppet has a suite of command-line tools to start virtual instances and install Puppet on them without having to manually log in to each virtual machine. Since 2011, Puppet has been available under the Apache 2.0 license; previously, it was released under the GPL v2.0 license.

Puppet is a popular configuration management tool and is widely used by companies such as Nokia, Dell and the Wikimedia Foundation. The user base seems to be focused mainly on Linux, especially Ubuntu and RHEL, but Puppet supports Windows and Unix as well. Many free manifests are available that address a wide variety of service deployment and administration tasks. The manifests themselves are written in Ruby or in a Puppet description language (a simplified version of Ruby). Puppet has two different web interfaces. One, the Puppet Dashboard [102], is developed by Puppet Labs, which is available in both the commercial and, with a reduced set of functionalities, in the community version. The second interface is Foreman [103], which has more functionality and integrated support for cloud systems. It requires, in addition to the standard database used by Foreman, PuppetDB (v2.2) to use `storeconfigs`, used to export configuration details from hosts and in return used by other hosts for their configuration, such as a database server address. Both of the interfaces are capable of displaying the status of nodes and of assigning manifests and roles to them, but the Puppet dashboard incapable of provisioning virtual resources directly.

Extensive documentation is available for Puppet, which introduce the topic, presuming a working knowledge of the basic concepts and focus on best practise approaches and very advanced setups [104] [105].

3.1.4.3 Puppet Manifests

The deployment makes use of a great variety of Puppet manifests that are used to deploy OpenStack and essential configurations to individual nodes.

The manifest used to set up the network configuration on the nodes is the `example42/network` manifest. It sets up the network interfaces and supports bonded network interfaces.

In the testbed, StackForge/OpenStack manifests are used to deploy OpenStack. These are continuously being developed and upgraded. They are referenced in the official OpenStack documentation for deploying OpenStack with Puppet, are very complex, but offer total control on all individual setup parameters of OpenStack.

The Enovance Ceph manifests were used to deploy Ceph via Puppet on the testbed.

The manifests assigned to the individual hosts differ depending on their role within the overall testbed (see Figure 3.4). The controller node has the most manifests assigned, as it hosts most of the components of OpenStack. The Compute hosts are only use a small number of manifests, while the storage nodes use none. Further information on the individual manifests is presented in Appendix C.

3.2 Initialization

To do the tests as described in Sections 2.2.1 and 2.2.3 multiple servers have to be set up, and virtual machines have to be configured identically to avoid differences between runs.

3.2.1 Testing System

The testing system consists of the Ceph storage cluster with three storage nodes and the OpenStack services described in Section 3.1. In addition, a virtual machine on

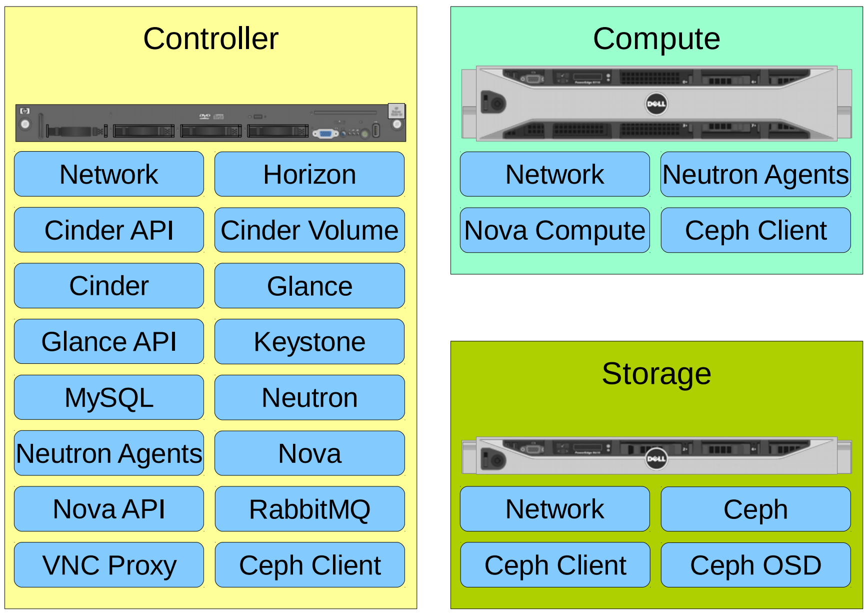


Figure 3.4: Roles for the individual node types.

a different host was used as the benchmarking server. The benchmarking system is the Phoronix Test Suite [106]. It comes with the options to upload results to the OpenBenchmarking.org [107] platform or to a private server using Phoromatic [108]. In the testbed, the latter approach is used, so that the results can be easily extracted from the system for post processing. The online platform is limited to simple plots which are not fit for purpose.

OpenStack supports the configuration of instances at boot time by passing data via the metadata service to the virtual machine. The virtual machine image requires the `cloud-init` [109] package to be part of the Glance image. Linux distributions, such as Ubuntu or Fedora, offer cloud images with pre-installed `cloud-init`, that is compatible with many cloud systems, such as OpenStack, Amazon AWS and Microsoft Azure. It offers many ways of modifying the VM when booted, such as changing the hostname, user account management, injecting SSH keys and running user defined scripts.

For the testing, a bash script was created to install the Phoronix Test Suite client and the benchmark profiles (see Listing E.3). As these profiles normally only execute three times, the repetition count and testing duration was extended to conform to the testing procedure described in Section 2.2. At the end of the script, it starts the client, connects to the Phoromatic server and finishes. The client, after being registered with the server, is ready to execute the benchmarks selected by the server.

```
[global]
osd_pool_default_pgp_num = 1024
osd_pool_default_pg_num = 1024
osd_pool_default_size = 2
osd_pool_default_min_size = 2

[client]
rbd_cache = false

[osd]
debug ms = 0
debug osd = 0
debug filestore = 0
debug journal = 0
debug monc = 0

[mon]
debug ms = 0
debug mon = 0
debug paxos = 0
debug auth = 0
```

Listing 3.1: Basic Ceph cluster configuration with debug and reporting disabled.

3.2.2 Test harness

Each virtual machine is set up to run the tests presented in Listing E.4. These constitute the baseline performance tests. The test cases, representing the workload tests, are presented in Listing E.5. Of these, a specific workload is loaded and executed as appropriate.

3.3 Cluster configuration

The Ceph cluster is configured to host multiple pools, pinned to different drives. The pool used for the benchmarks is isolated on the 12 Western Digital RE4 drives. The cluster replication count has been set to two to limit the impact of the cluster network bandwidth limitation (see Table 3.3). This ensures that each block is transferred only once per replication, rather than twice. With a replication count of three, the file would be written to two other hosts which would double the network traffic and therefore reduce the write performance. In a bigger cluster, the replication load is spread and therefore the network bandwidth dependency will be less crucial, but in a small cluster it is a limiting factor.

The Ceph configuration for these experiments uses the default settings, and the parameters can be seen in the following configuration snippet. The debugging and reporting function on the OSD and Monitors are disabled and use CephX for authentication, as shown in Listing 3.1.

Table 3.5: Tested parameter values and their default configuration. For example, Configuration B reduces `osd_op_threads` by 50%, while Configuration C increases it by 100% and Configuration D by 400%.

Parameter	**Default**	**Tested**
osd_op_threads	2	1 (B), 4 (C), 8 (D)
osd_disk_threads	1	2 (E), 4 (F), 8 (G)
filestore_op_threads	2	1 (H), 4 (I), 8 (J)
filestore_wbthrottle_xfs_bytes_start_flusher	41943040	4194304 (K), 419430400 (L)
filestore_wbthrottle_xfs_ios_start_flusher	500	5000 (M), 50 (N)
filestore_wbthrottle_xfs_inodes_start_flusher	500	5000 (O), 50 (P)
filestore_queue_max_bytes	104857600	1048576000 (Q), 10485760 (R)
filestore_queue_committing_max_bytes	104857600	1048576000 (S), 10485760 (T)
objecter_inflight_op_bytes	104857600	1048576000 (U), 10485760 (V)
objecter_inflight_ops	1024	8192 (W), 128 (X)

The parameters under test (see Table 3.5) are part of the Ceph environment and affect all pools equally. As described in Sections 2.2.1 and 2.2.2, by design, different configurations differ in exactly one parameter. Thus, the affect of each parameter can be seen in isolation.

The parameters to tune are chosen in a three stage procedure. In the first step parameters that are dictated by the environment and greater Ceph environment are picked and set, such as the cluster ID or the data distribution. In the second step parameters are filtered for their relation to performance. Parameters that enable or disable counters or logging would be set to the desired setting and others left to their default configuration. The remaining set of parameters are the ones to be tested for their impact and relation to performance. Since there is no documentation available that guides users in setting them, they have to be picked randomly in the third step and tested for their impact.

The selected parameters relate to the OSDs and the filestore. Using Ceph in combination with Cinder and Glance does not require using components such as the RADOS Gateway, which would be required when using OpenStack Swift, or the metadata server (MDS).

- The `osd_op_threads` parameter specifies the number of threads to handle Ceph OSD Daemon operations. Setting it to zero disables multi-threading, while increasing it may increase the request processing rate. Depending on the hardware being used, the result can be positive or negative. If a device is too busy to process a request, it will timeout after a number of seconds (30 seconds by default).

- `osd_disk_threads` specifies the number of disk threads, used to perform background disk intensive OSD operations, such as scrubbing and snapshot handling. This parameter can affect the pool performance if the scrubbing process coincides with a data access. This parameter defaults to 1, indicating that no more than one operation is processed concurrently.
- `filestore_op_threads` specifies the number of file system operation threads that may execute in parallel.
- `filestore_wbthrottle_xfs_bytes_start_flusher`, `filestore_wbthrottle_xfs_ios_start_flusher` and `filestore_wbthrottle_xfs_inodes_start_flusher` configure the filestore flusher, preventing large amounts of uncommitted data building up before each filestore sync. Conversely frequently synchronising large numbers of small files can adversely affect performance. Therefore, Ceph manages the commitment process by choosing the most appropriate commitment rate using these parameters.
- `filestore_queue_max_bytes` and `filestore_queue_committing_max_bytes` specify the size of the filestore queue and the amount of data that can be committed in one operation.
- `objecter_inflight_op_bytes` and `objecter_inflight_ops` modify the Ceph objecter, which handles the placement the objects within the cluster.

3.4 Evaluation

The following work has been partially published in Scalable Computing: Practice and Experience journal [110].

In the foregoing sections a testbed was created containing a number of storage pools. Different configurations of parameters in the environment of those pools were then created and the affects of these different configurations, while running the `fio` [48] tool as a benchmark, on pool performance were recorded. In the presentation of the results over the forthcoming sections these configurations are labelled B-X; configuration A represents the default Ceph configuration. The benchmark was set to run for 300 seconds and a test data size of 10GB. The IO engine was set to sync, which uses `fseek` to position the I/O location and avoids caching. In this way a worst case scenario test could be performed. Access was set to direct and buffering disabled. For each run, there was a start and a ramp delay of 15 seconds. Random and sequential access patterns were tested for both reads and writes, each with block sizes of 4KB, 32KB, 128KB, 1MB and 32MB. A total of 9 runs for each benchmark configuration was executed to achieve a representative average over multiple runs.

A total of 12 virtual machines, equally distributed across the three compute hosts, was used to stress the system. Each VM was set to use 4 cores and 4GB of memory. The virtual disk was set to use a 100GB Cinder volume. RADOS Block Device (RBD) caching was disabled on the Ceph storage nodes and on the compute hosts in the QEMU/KVM hypervisor settings. The diagrams in the following sections show the mean value across all 12 VMs.

3.4.1 4KB

The series of experiments on configurations A-X, while running the benchmark using a read and write block size of 4KB, were compared in terms of **I**nput/**O**utput Operations **P**er **S**econd (IOPS).

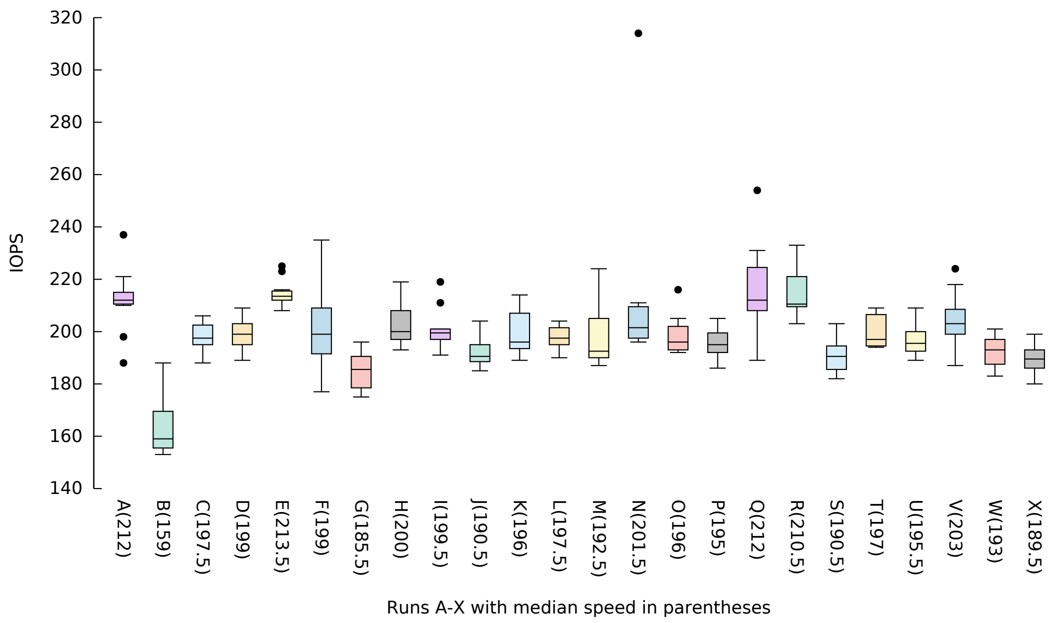


Figure 3.5: FIO random read 4KB.

Figures 3.5 and 3.6 show the performance for random read and write access workloads, respectively. In Figure 3.5, configuration B (`osd_op_threads` decreased) with 159 IOPS deviates most from the default configuration A. Since the `osd_op_threads` are set to 2 at default, it reduces the concurrency of the read operations. In fact, the performance of the default configuration can at best be matched but not exceeded. In Figure 3.6 the number of IOPS is so low that no real conclusion can be drawn.

When the storage system is tested against 4KB sequential read accesses (see Figure 3.7), the difference between the lowest performing Configuration F (`osd_disk_thread` increased x2) and the highest performing Configuration A (default) is over 105% or 378 IOPS. Configurations Q (`filestore_queue_max_bytes` increased), E (`osd_disk_threads` increased x2) and N (`filestore_wbthrottle_xfs_ios_start_`

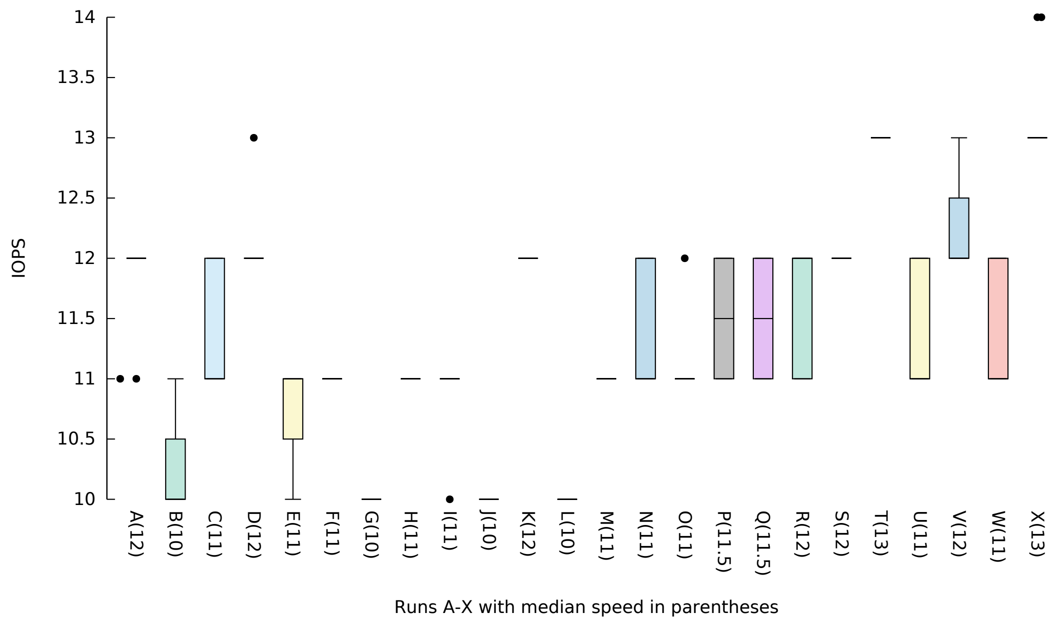


Figure 3.6: FIO random write 4KB.

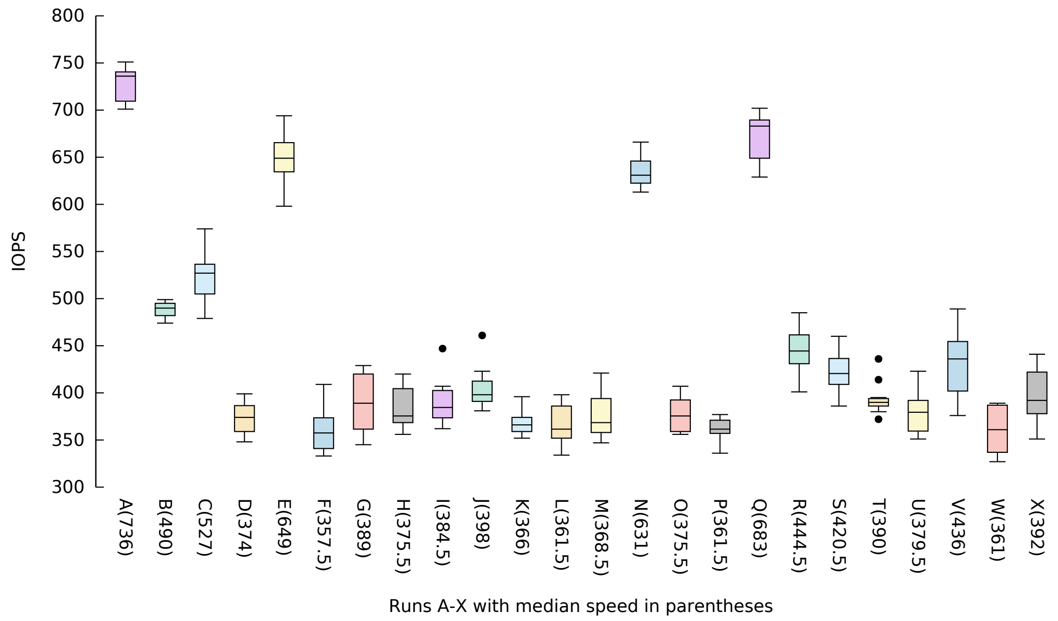


Figure 3.7: FIO sequential read 4KB.

`flusher` decreased) perform much better than other configurations, but no configuration can match the Default A. For 4KB sequential writes (see Figure 3.8), the results are very even, except for Configuration N. In contrast to the 4KB read accesses, where Configuration N performed well, performance here is reduced by 25% compared to the mean of the other configurations. This suggests that, when writing small blocks, the small flusher threshold is contraindicated, whereas it does not negatively impact read performance.

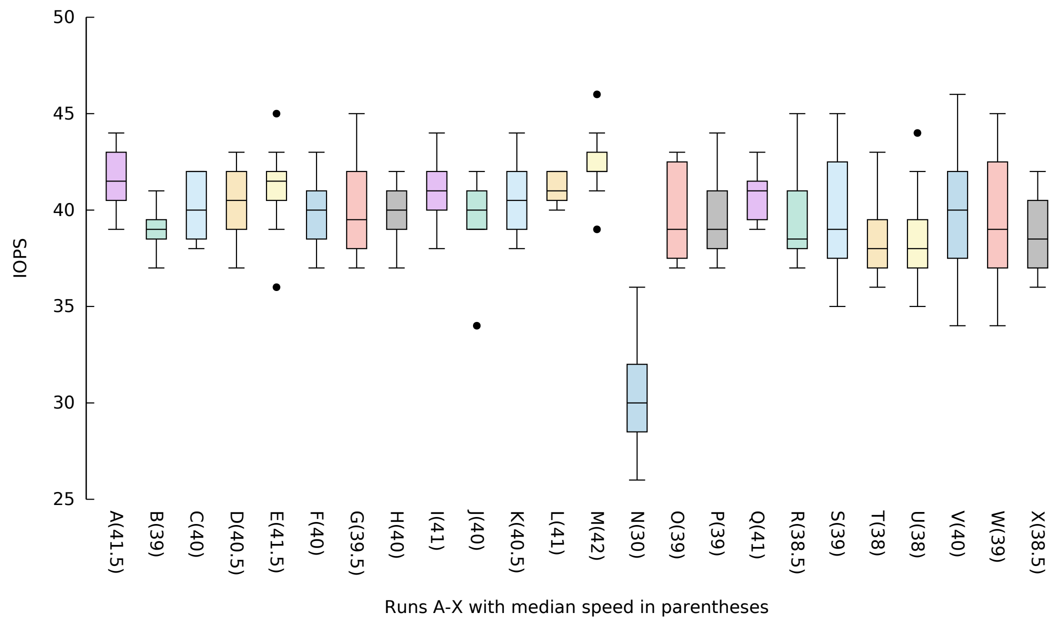


Figure 3.8: FIO sequential write 4KB.

3.4.2 32KB

For 32KB random read accesses (see Figure 3.9), the performance of the different configurations was very similar to the default Configuration A. The configuration with the greatest increase over the default (of 2.6%) is Configuration S (`filestore_queue_committing_max_bytes` increased). Configurations C, H, and R show an increase between 1.3% and 2%. The configuration with the biggest performance drop is B (`op_threads` decreased). In this case, the performance drops by 59.6%. As with the 4KB random reads, the low concurrency on the OSD harms performance when using small random I/O.

During 32KB writes, the limitations of the underlying hardware are clearly visible (see Figure 3.10). Nevertheless, Configuration P (`filestore_wbthrottle_xfs_inodes_ start_flusher` decreased) increases performance by 2 IOPS without any variation between the hosts, which is a 16.7% performance increase, while Configuration B (`op_threads` decreased) reduced throughput by 2 IOPS. Overall these results are not indicative of a real performance increase, due to the actual size of the differences.

The sequential 32KB read performance of the cluster is positively influenced by most of the configurations (see Figure 3.11). Only Configurations E (`osd_disk_threads` increased), K (`filestore_wbthrottle_xfs_bytes_ start_flusher` decreased) and V (`objecter_inflight_op_bytes` decreased) reduce throughput by up to 3.4%. The highest gains of about 17% are achieved by Configurations D (`osd_op_threads=8`) and R (`filestore_queue_max_bytes` decreased). Many other configurations increase performance by about 10%. In general, the results show a high amount of jitter, with

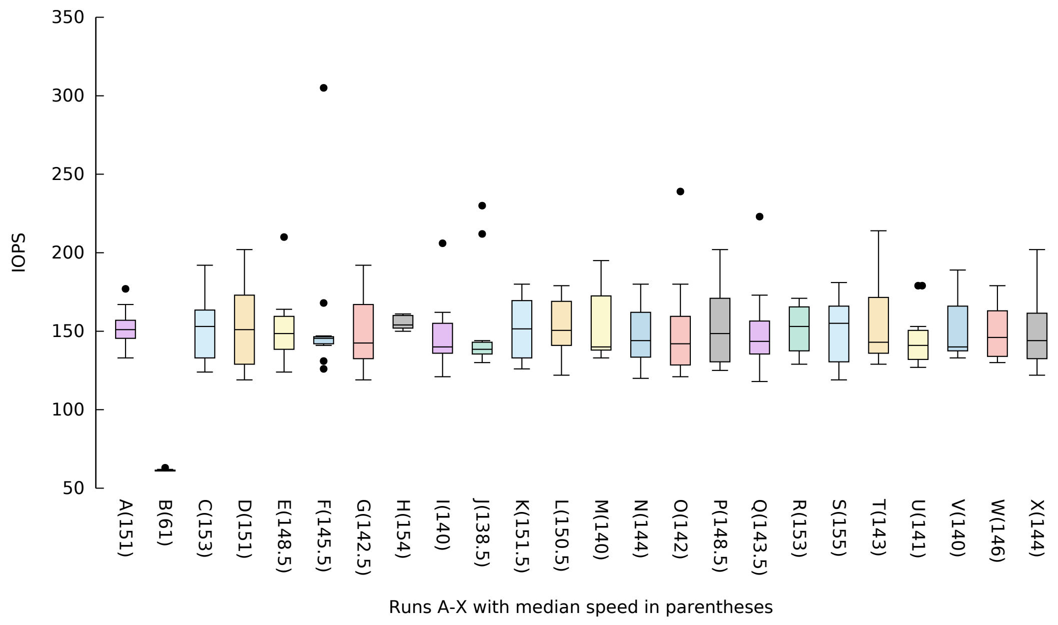


Figure 3.9: FIO random read 32KB.

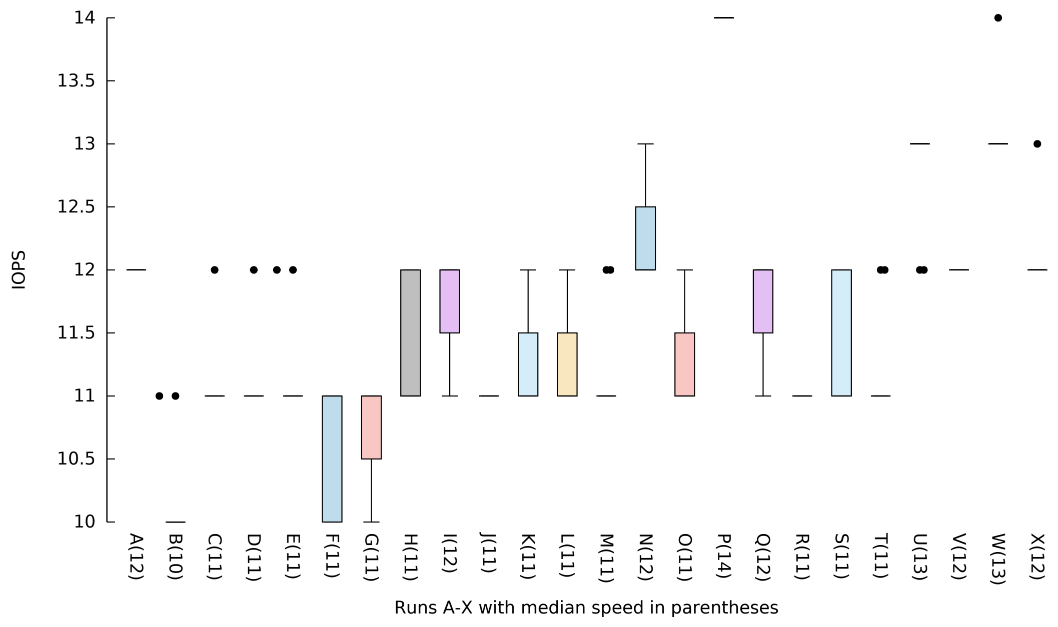


Figure 3.10: FIO random write 32KB.

results spreading up to 90 IOPS (Configuration W).

For the sequential 32KB writes, there is no configuration that clearly outperforms the default Configuration A (see Figure 3.12). In contrast, Configurations K (`filestore_wbthrottle_xfs_bytes_start_flusher` decreased) and N (`filestore_wbthrottle_xfs_ios_start_flusher` decreased) have a highly negative impact on throughput. While the former reduces it by 10.5%, the latter reduces it by 22.4%. Changing the write back flusher to flush data earlier has a direct impact on small sequential write accesses. While such a behaviour was also visible for Configuration N during 4KB sequential writes, it was not observed with configuration K, since the access block size was too small to breach the write back buffer threshold. In testbed with faster hardware, the impact of both would be more visible, since transfers would be interrupted more frequently by the flusher.

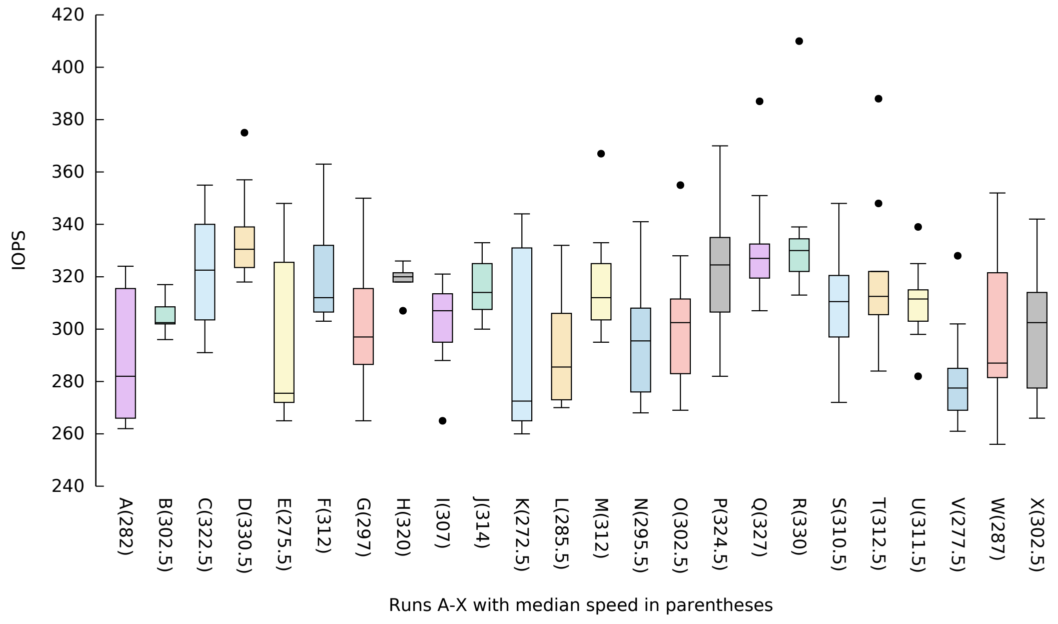


Figure 3.11: FIO sequential read 32KB.

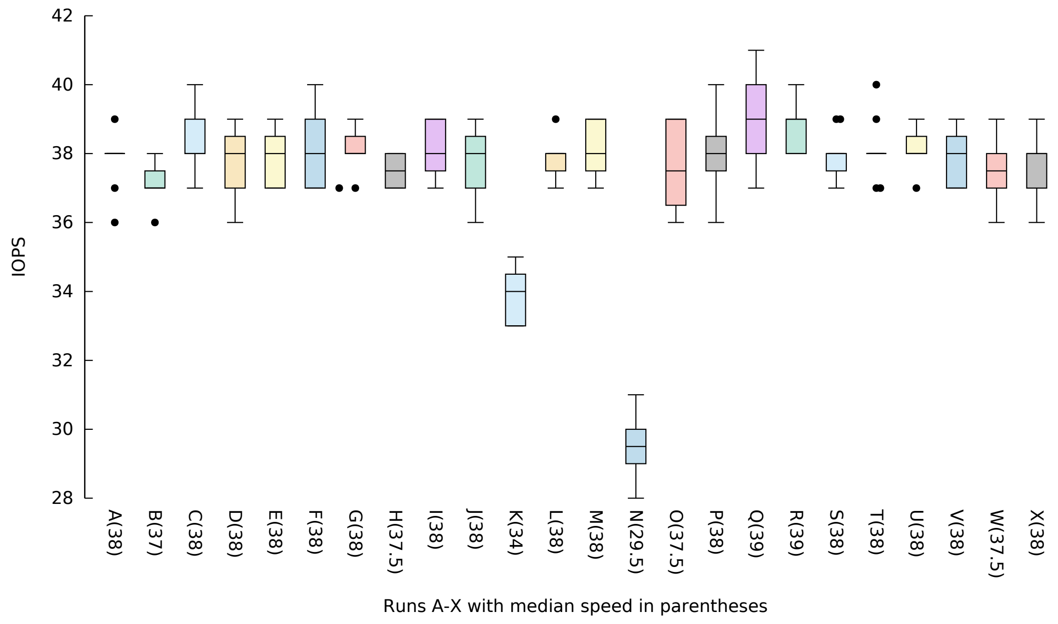


Figure 3.12: FIO sequential write 32KB.

3.4.3 128KB

When using 128KB block sizes for random read accesses (see Figures 3.13), all configurations show improvement over the default configuration, except for Configurations B (`osd_op_threads` decreased) and N (`filestore_wbthrottle_xfs_ios_start_flusher` decreased). A maximum gain of 8% was observed for Configuration K (`filestore_wbthrottle_xfs_bytes_start_flusher` decreased). When writing random blocks with the same block size (see Figure 3.14), the difference was more pronounced, with K being 14% faster than the default. The performance difference between the best configuration, K (`filestore_wbthrottle_xfs_bytes_start_flusher` decreased), and the worst configuration, L (`filestore_wbthrottle_xfs_bytes_start_flusher` increased), was almost 30%. In this case the same parameter with different values changes the performance to a great extent. The same pattern can be observed, with smaller differences, for each of the pairs from M,N to W,X. The performance of the default configuration lies between the worst and the best configurations. This is a remarkably fortuitous choice for the default configuration, since,from a study of the history of Ceph [111], it seems to have been chosen arbitrarily and has never been updated since the system was conceived and implemented.

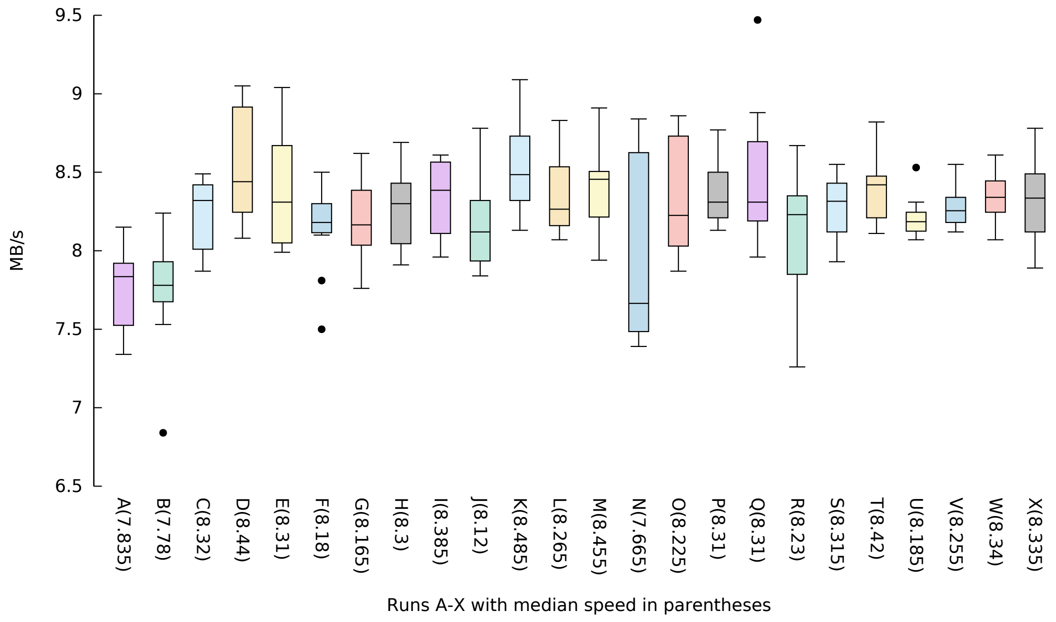


Figure 3.13: FIO random read 128KB.

For sequential 128KB read accesses (see Figure 3.15), the performance is comparable between all configuration with a difference of just 9% between the lowest (Configuration O) and the highest (Configuration Q). The default configuration is only surpassed by Configurations N (`filestore_wbthrottle_xfs_ios_start_flusher` decreased), Q (`filestore_queue_max_bytes` increased), R (`filestore_queue_max_bytes` decreased) and X (`objecter_inflight_ops` decreased). The beneficial effect of Configu-

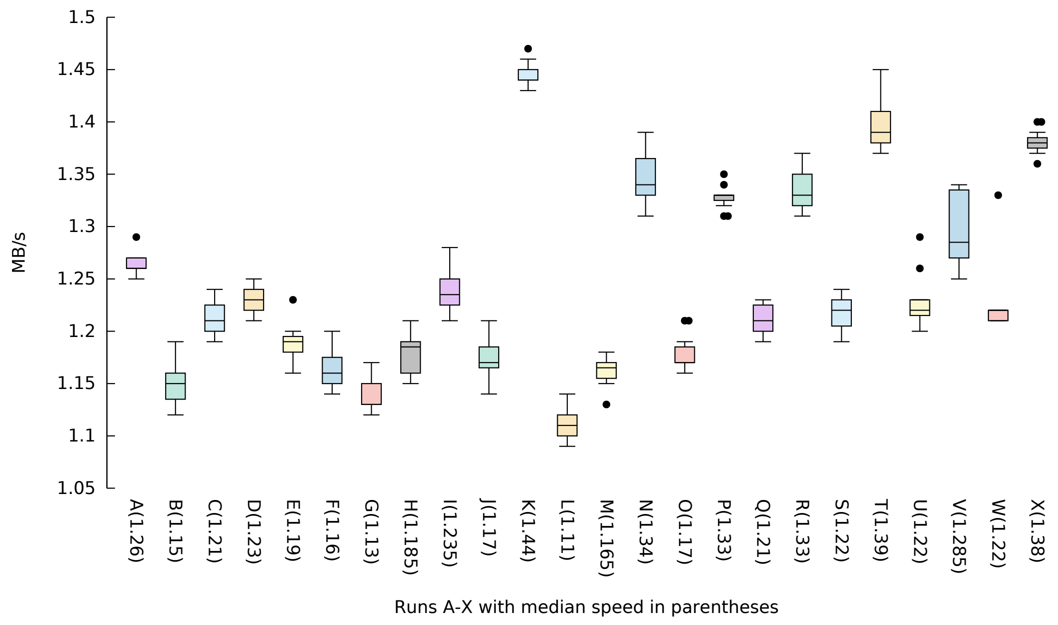


Figure 3.14: FIO random write 128KB.

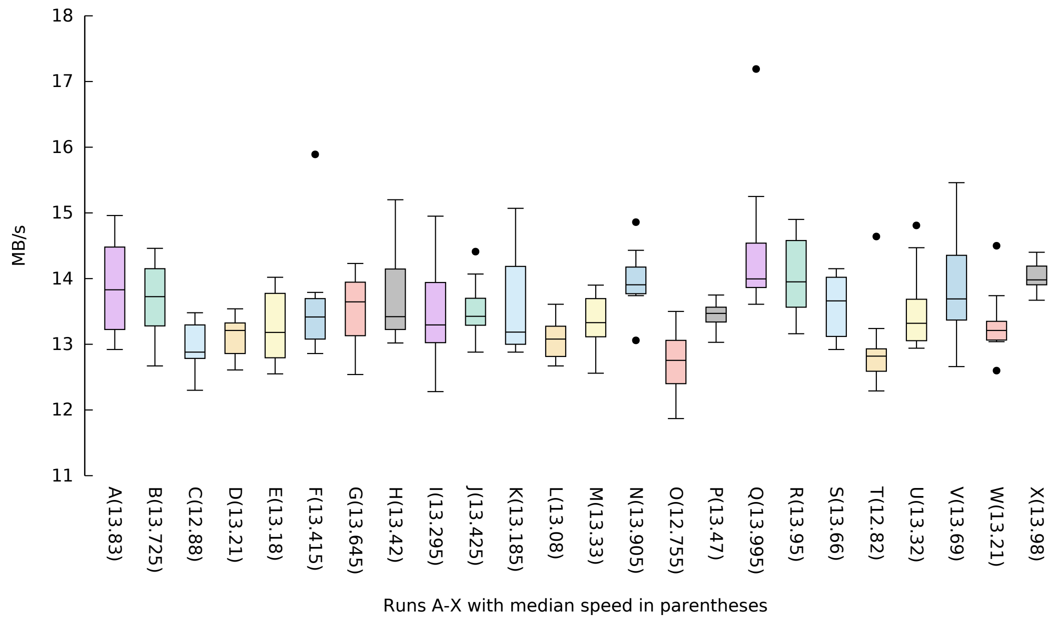


Figure 3.15: FIO sequential read 128KB.

rations Q and R is unexpected, since they both modify the same parameter differently and yet both result in increased performance.

For sequential 128KB write accesses (see Figure 3.16), Configurations N (`filestore_wbthrottle_xfs_ios_start_flusher` decreased) and K (`filestore_ wbthrottle_xfs_bytes_ start_flusher` decreased) had the most negative impact on performance. Configurations K and N performed 16.5% and 10.5% lower, respectively, than the default configuration. In both of these configurations the write back flusher

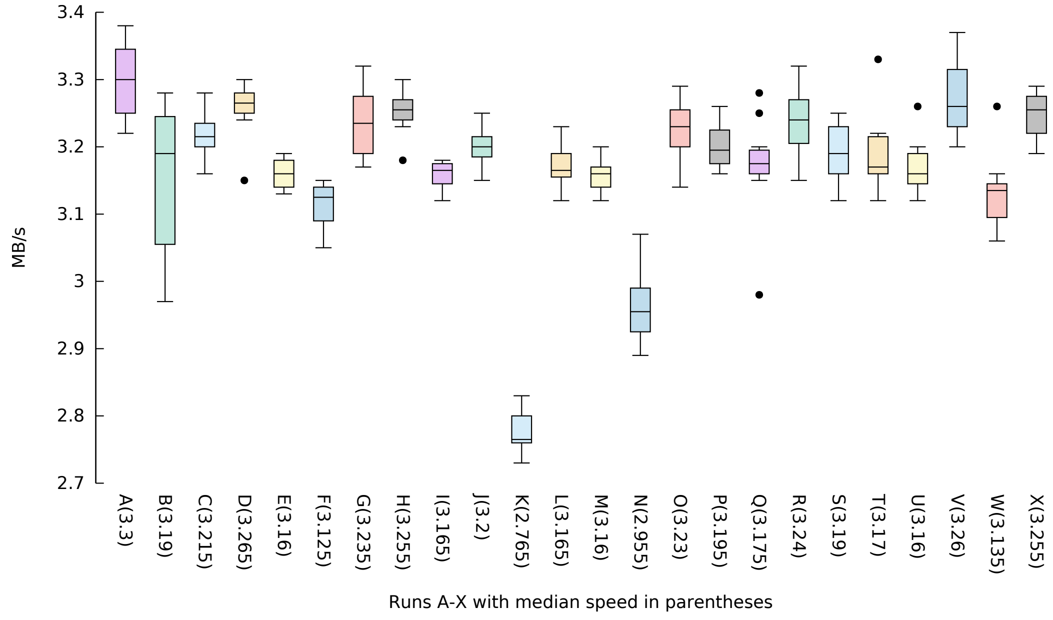


Figure 3.16: FIO sequential write 128KB.

process is executed too frequently, reducing throughput. Gains were not observed under this access pattern.

3.4.4 1MB

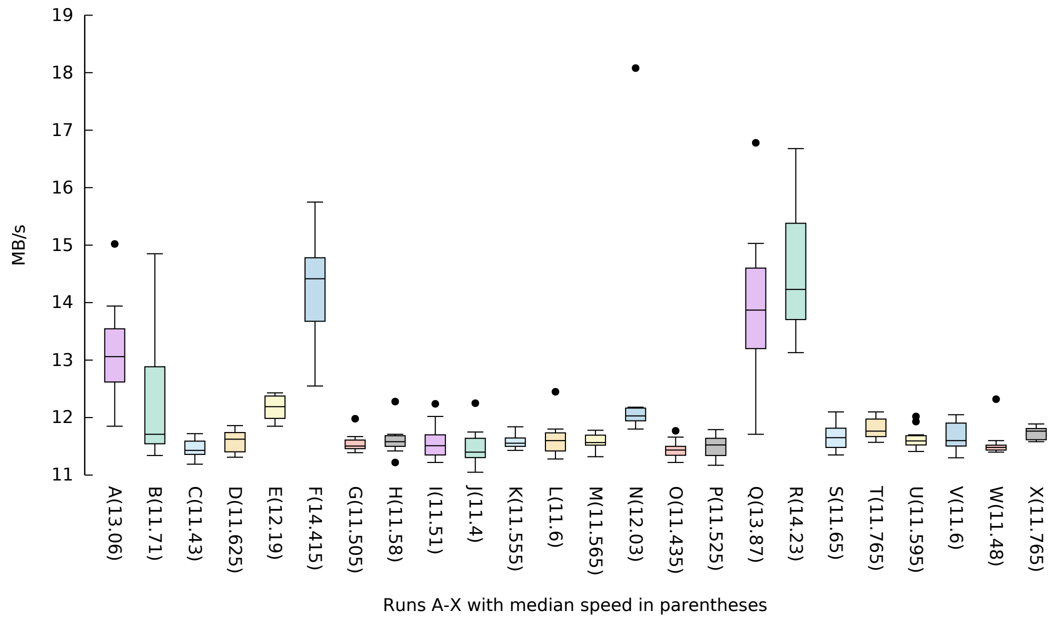


Figure 3.17: FIO random read 1MB.

For random 1MB read accesses (see Figure 3.17), Configurations F (`disk_threads` increasd x4), R (`filestore_queue_max_bytes` decreased) and Q (`filestore_queue_` `max_bytes` increased) performed better than the rest, with Configuration F improving performance by 10% over the default. These configurations, in addition to the default and Configuration B, showed large variations between the different VMs, while the results of the other configurations were more uniform. The most disruptive configuration was J (`filestore_op_threads` increased x4), reducing performance by 13%.

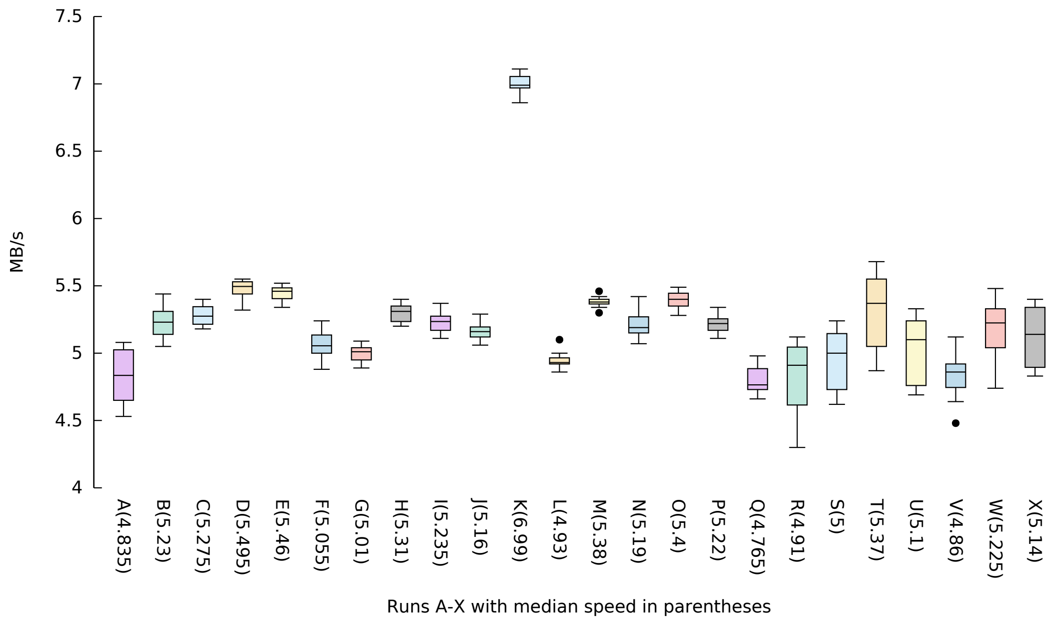


Figure 3.18: FIO random write 1MB.

For random 1MB write accesses (see Figure 3.18), Configuration K (`filestore_wbthrottle_xfs_bytes_start_flusher` decreased) improved performance by 44.5% over the default configuration. Configuration Q (`filestore_queue_max_bytes` increased) was the only configuration that showed reduced performance by 1.5%. Remarkably, all other configurations improved performance over the default.

For sequential 1MB read accesses (see Figure 3.19), no configuration improved performance. The highest regression observed was 8% (Configuration W). Configuration B (`osd_op_threads` decreased), while not increasing performance on average, showed large variation between the different concurrent VMs.

For sequential 1MB writes accesses (see Figure 3.20), Configuration K (`filestore_wbthrottle_xfs_bytes_start_flusher` decreased) showed the highest gains of 28%. Configuration U (`objecter_inflight_op_bytes` increased) was the only configuration to reduce performance, producing a drop of 1.0%.

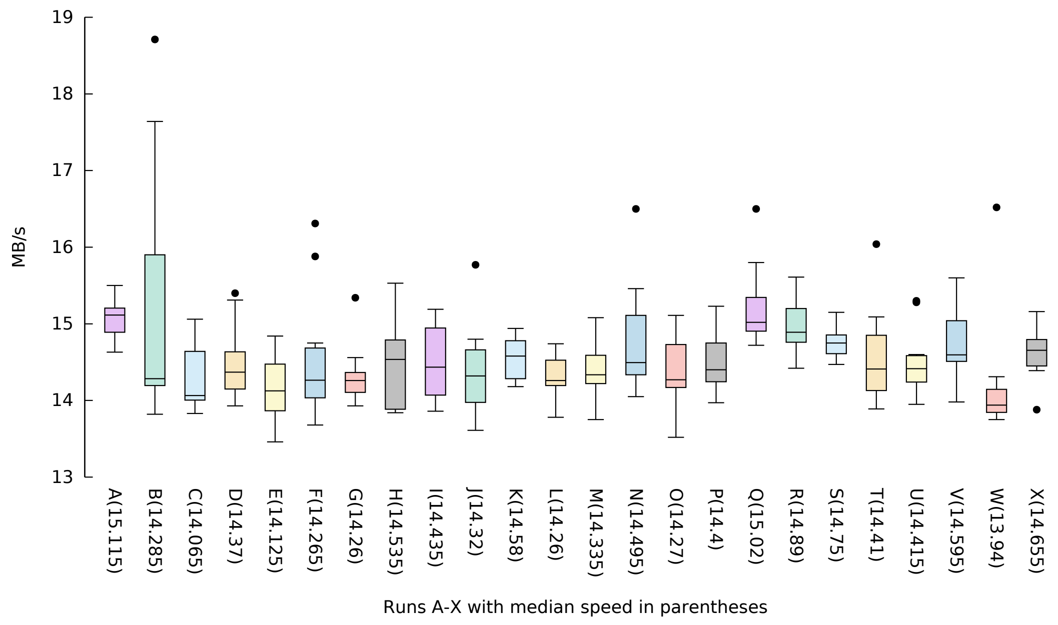


Figure 3.19: FIO sequential read 1MB.

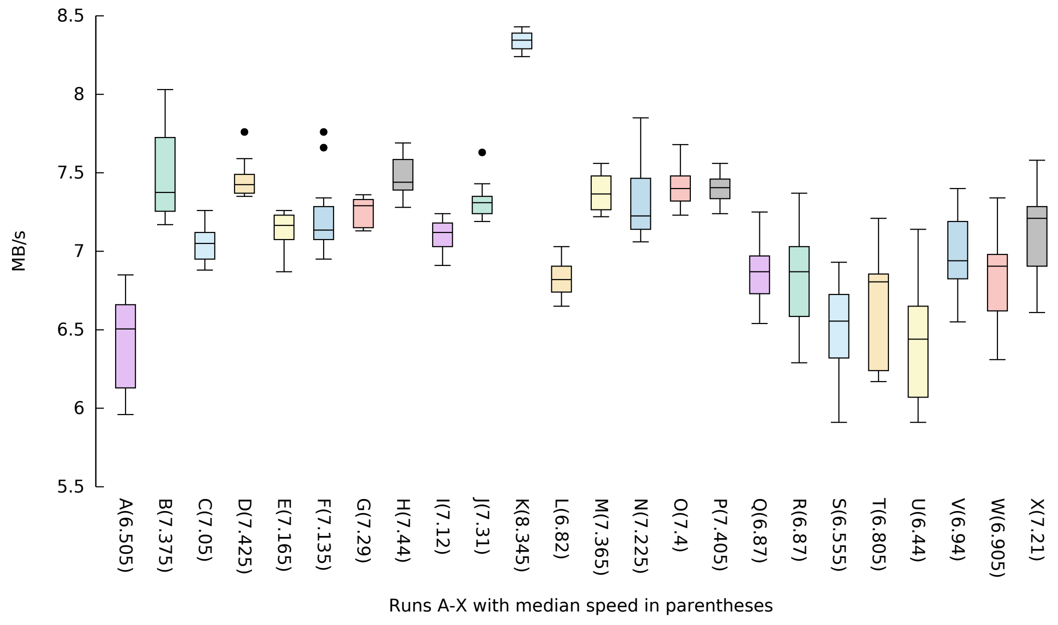


Figure 3.20: FIO sequential write 1MB.

3.4.5 32MB

For random 32MB read accesses (see Figure 3.21), only Configuration Q (`filestore_queue_max_bytes` increased) improved performance over the default configuration. For random 32MB write accesses (see Figure 3.22), Configuration R (`filestore_queue_max_bytes` decreased) performs best. Configurations Q and R modify the same parameter differently, resulting in performance increases and decreases between the random 32MB reads and writes respectively. Thus, the

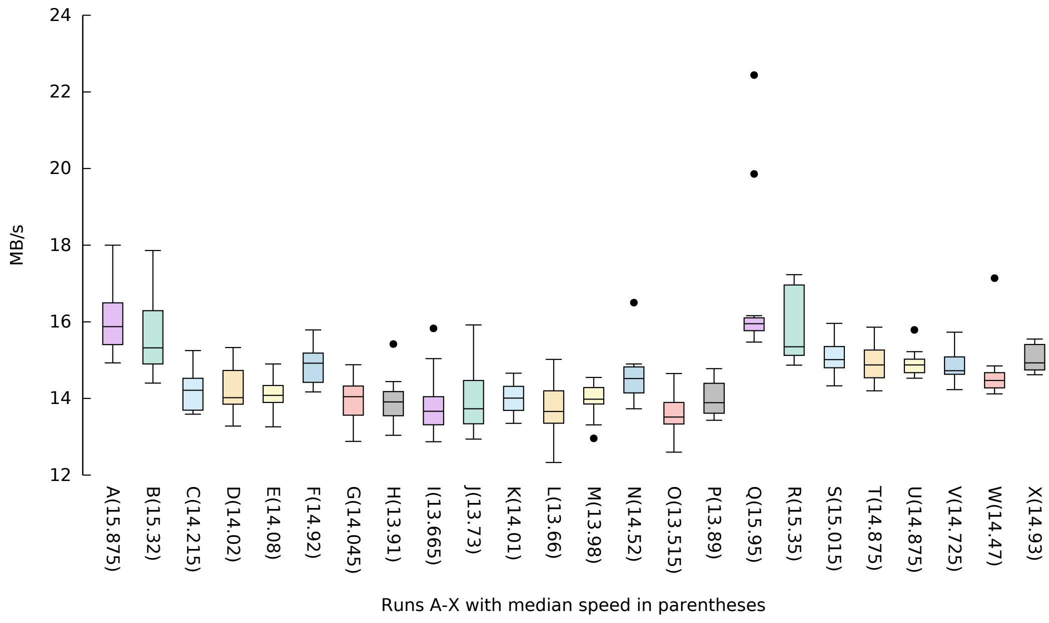


Figure 3.21: FIO random read 32MB.

parameter when altered in a particular way has a positive effect when reading and a negative effect when writing and when altered in the opposite way has the respective opposite effect. As before, the performance of the default configuration lies between the worst and the best configurations. Configuration O (`filestore_wbthrottle_xfs_inodes_start_flusher` increased) is the most disruptive for random reads, reducing performance by more than 2 MB/s in comparison to the default configuration. For random writes, multiple configurations (E, H, O) have a strong negative impact.

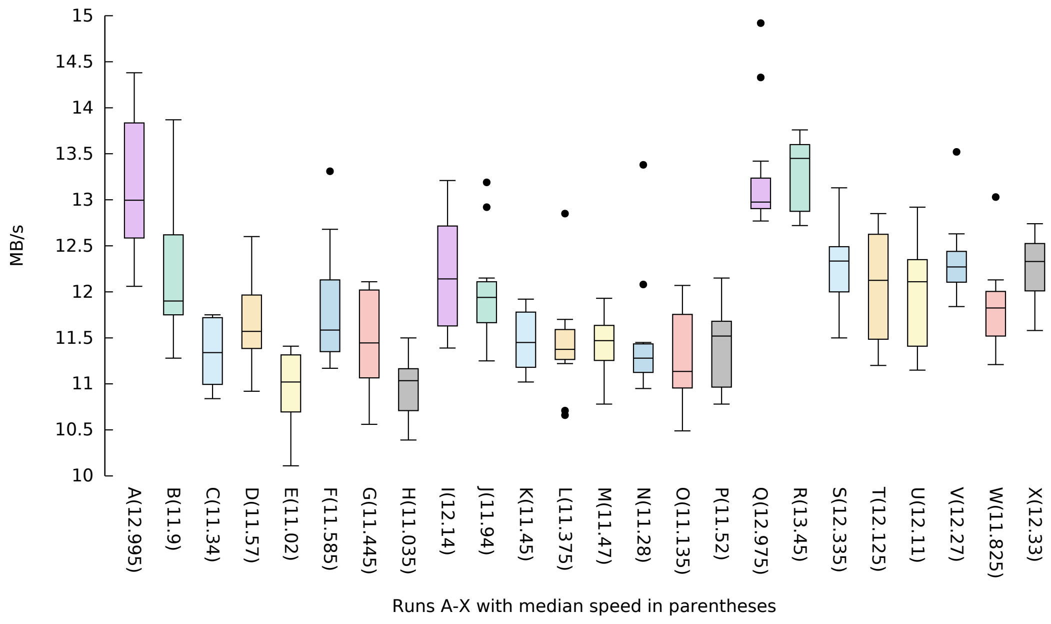


Figure 3.22: FIO random write 32MB.

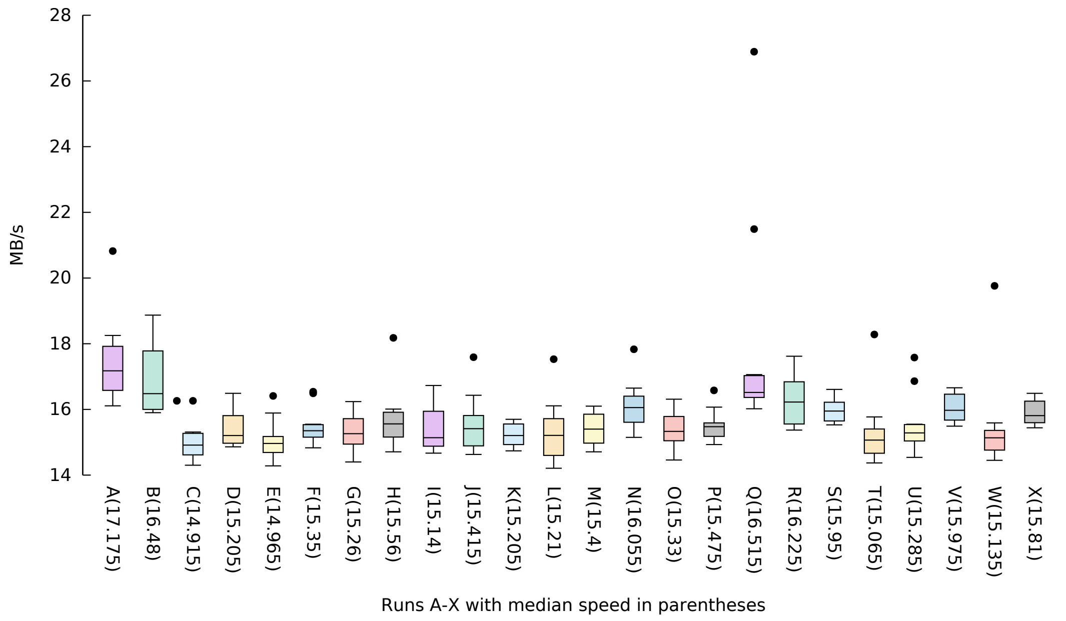


Figure 3.23: FIO sequential read 32MB.

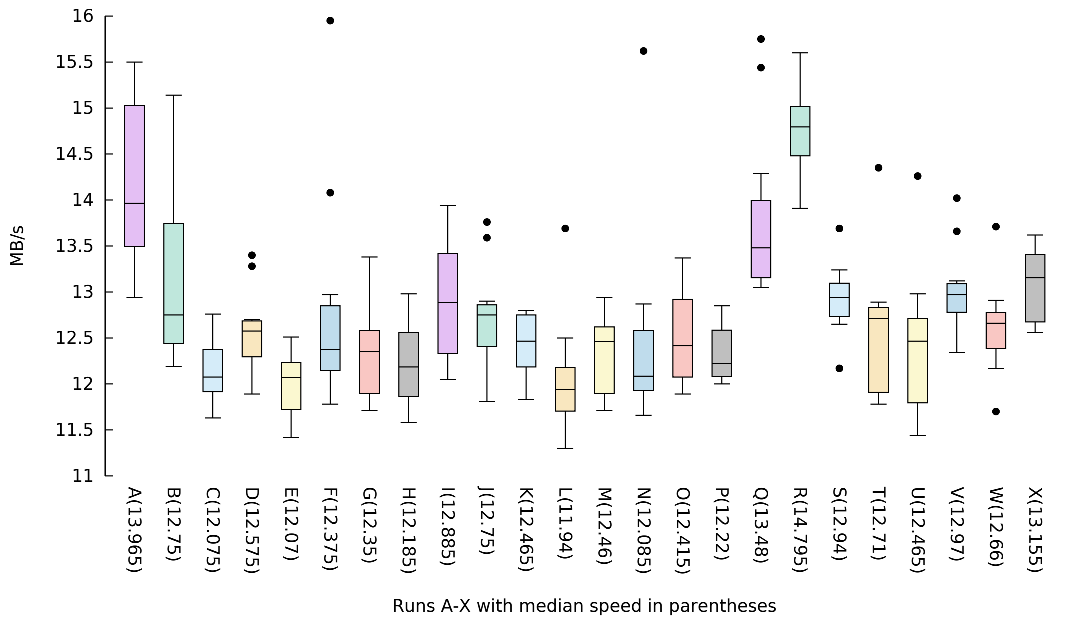


Figure 3.24: FIO sequential write 32MB.

For sequential 32MB read accesses (see Figure 3.23), all configurations that deviate from the default reduce the performance by up to 14% (Configuration C) or 2.2 MB/s. Configuration Q (`filestore_queue_max_bytes` increased) showed small jitter, but multiple outliers that deviated but 10MB/s. For sequential 32MB writes (see Figure 3.24), Configuration R improved performance by 6%, whereas the other configurations reduced performance by up to 14.5% (Configuration L).

3.4.6 Summery

As the size of the access pattern increases from 4KB to 32MB it can be seem that certain parameters become more dominant in influencing performance. Configuration R, for example, performed well for writes larger than 128KB and read accesses with 32KB and 128KB block size. Other access sizes and patterns saw a performance decrease by up to 39.4% (4KB sequential read).

The lowest performing configurations for combined 4KB accesses are Configurations B and G. These two configurations performed similar for writes, but during sequential reads Configuration B (`osd_op_threads=1`) outperformed Configuration G `osd_disk_threads=8`, while for random reads the opposite was observed. A configuration that outperforms the default for 4KB accesses was not tested.

The lowest performing configuration for combined 32KB accesses were Configuration B (`osd_op_threads=1`). This low performance originated from the low performance during random read operations, where performance was about 60% lower than the other configurations. The best performing configuration for 32KB accesses was Configuration P `filestore_wbthrottle_xfs_inodes_ start_flusher=50`, which outperformed the default configuration in random writes and sequential reads by 15%, while no significant differences were recorded for sequential writes and random reads.

For 128KB accesses the lowest performing configurations were Configuration B (`osd_op_threads=1`), L (`filestore_wbthrottle_xfs_bytes_start_flusher= 419430400`) and O (`filestore_wbthrottle_xfs_inodes_start_flusher=5000`). The highest performing configuration was Configuration X (`objecter_inflight_ops=128`). Overall, most configurations increased performance during 128KB random reads, while performance during random writes and sequential reads and writes was mostly reduced.

For 1MB accesses the lowest performing configuration was Configuration U (`objecter_inflight_op_bytes=1048576000`). Performance for read accesses was reduced for most configurations, while performance during writes had mostly increased performance. The best performing configuration for 1MB accesses was Configuration K (`filestore_wbthrottle_xfs_bytes_ start_flusher=4194304`). For this configuration read performance was reduced, but write performance was greatly enhanced.

For 32MB accesses only Configuration R (`filestore_queue_max_bytes=10485760`) matched the performance of the default configuration. It lost performance during read accesses, but gained performance during writes. The lowest performance was recorded for Configurations E (`osd_disk_threads=2`) and L (`filestore_wbthrottle_xfs_bytes_ start_flusher=419430400`). Both of these configurations lost over 12% in each of the 32MB accesses. Other configurations experienced similar performance decreases for reads, while performing slightly better for write accesses.

3.5 Case Studies

In the work presented above the potential impact on pool performance, resulting from changes to the Ceph environment, is examined. The lessons learned are applied in Chapter 5 to determine the Ceph configuration corresponding to the largest performance improvements for particular workload characteristics. In advance of this work, this section presents case studies showing the impact that changes made to the greater Ceph environment have on pool performance. The first study attempts to improve pool performance by changing the file system deployed on the OSD and the second attempts to improve pool performance by altering the I/O scheduler and the associated queue depth. The relationships between these parameters and pool performance are described in Section 2.4.4.

The results obtained in these case studies do not take account of changes in the Ceph environment nor do they relate to changes to parameters associated with the functional component of a pool. As such, they are not considered during the improvement process derived from mapping workload characteristics to parameters of the Ceph environment. Nevertheless, these studies show the affects of environmental changes on pool performance and hence underline the empirical utility of the concept.

3.5.1 Engineering Support for Heterogeneous Pools within Ceph

This following work has been published in the 2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) [112].

Prior to the work done here, it was not clear that Ceph could run different file systems on each OSD within the same Ceph cluster, since this feature is not explicitly mentioned anywhere in the documentation. Currently three different file systems are officially supported (XFS, BTRFS, ext4), with XFS being recommended for production use. BTRFS was supposed to become the future default production file system, but subsequently was dropped in mid 2016 in favour of a new file store. ext4 is supported but not recommended for large clusters, since its limitations constrain the maximum Ceph cluster size.

To provide for multi file system support, the approach adopted here is to physically partition the Ceph cluster. A small test cluster of one host with 10 1TB hard drives (Hitachi Ultrastar A7K1000, see Table 3.2) was constructed to illustrate the utility of the approach. XFS and BTRFS were each deployed on half of the disks using `ceph-deploy` together with standard formatting and mounting settings.

When OSDs were added to the cluster, the system treated each in the same way resulting in a homogeneous view of the file systems resident in the underlying disks. If normal convention is followed, the creation of a pool will result in it using all of the available OSDs, and hence the pool would embody different underlying file systems. To avoid this situation and hence ensure that a pool is only associated with a single file system, a default pool creation is modified via its CRUSH map to recognise only those OSDs associated with a particular file system. This process enables the creation of heterogeneous pools, in the same Ceph cluster, each embodying a different file system. The process of accessing and editing the CRUSH map is described in Section 1.2.1.

The original Crush map of the cluster with one host and 10 OSDs is shown in Listing E.6.

To edit the CRUSH map to create the heterogeneous pools two alterations are necessary:

1. The physical collection of disks with a specific file system as a root, is added.
2. A rule is inserted to specify the use of certain collections only.

Only when these are added can both collections and pools be subsequently used. Listing E.7 shows the modified CRUSH map.

When the new compiled CRUSH map is uploaded, the cluster will change its data distribution accordingly. The two newly created pools can then make use of the new ruleset (see Listing E.8).

To show the difference between the two different pools, some preliminary benchmarks were run using the rados bench tool. These benchmarks were executed with 4KB and 4MB access sizes with 16 and 64 concurrent connections. The access modes used were sequential reads and writes and random reads. The benchmarks were executed three times on a clean cluster with a runtime of 300 seconds each. Before every run the cache was emptied to avoid caching effects distorting the results.

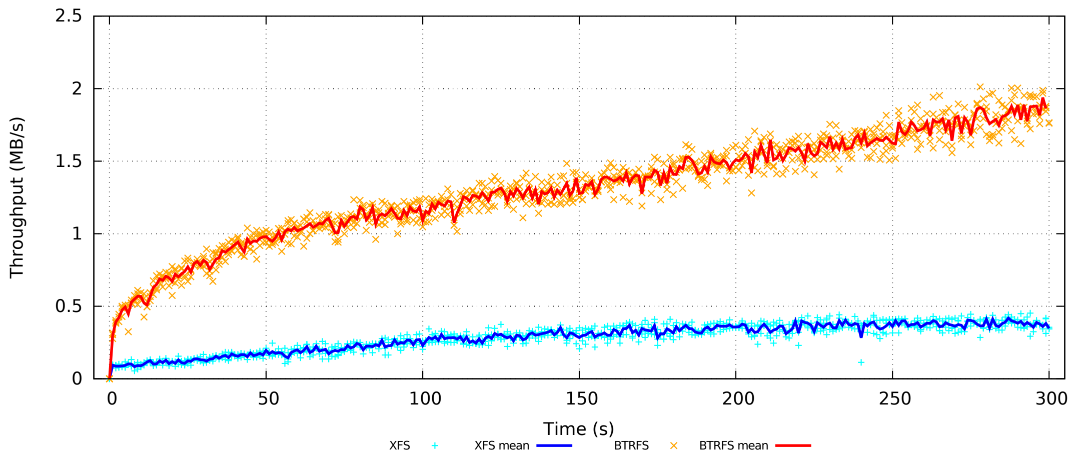


Figure 3.25: Rados bench random 4KB read with 16 threads.

The throughput for random 4KB reads with 16 threads (see Figure 3.25) showed an increase of over 400%. While the throughput curve for the XFS pool reached a limit around 0.4 MB/s, the BTRFS pool showed increasing throughput throughout the whole benchmark run. For both pools, the performance remained consistent over the three.

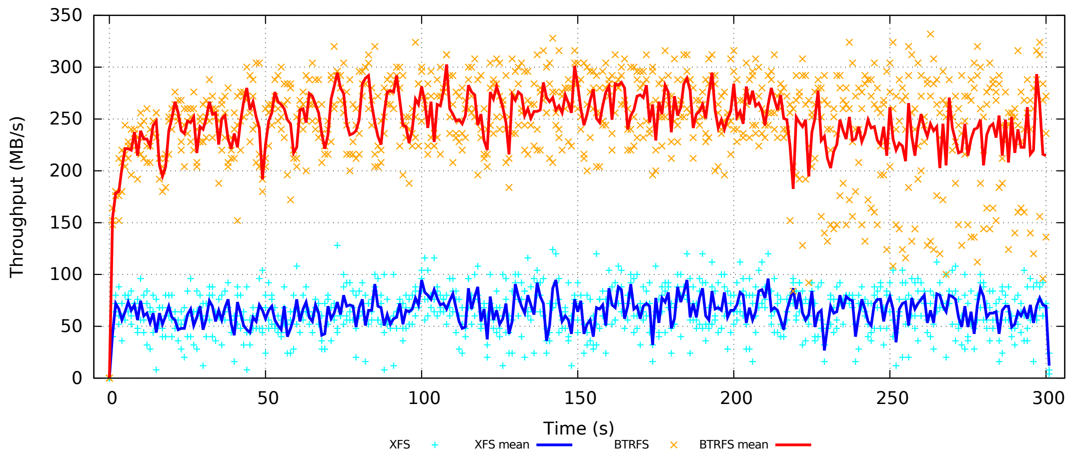


Figure 3.26: Rados bench random 4MB read with 16 threads.

For random 4MB reads with 16 concurrent threads (see Figure 3.26), BTRFS performed better than XFS. While the XFS pool managed an average throughput of 75 MB/s, the BTRFS pool managed around 250 MB/s. It is notable that the throughput varied considerably after 220 seconds runtime. In comparison to the random 4KB reads, the throughput showed more variance and jitter with rates varying between 150 and 330 MB/s for the BTRFS pool, while the variance in the XFS pool was between and 10 and 130 MB/s.

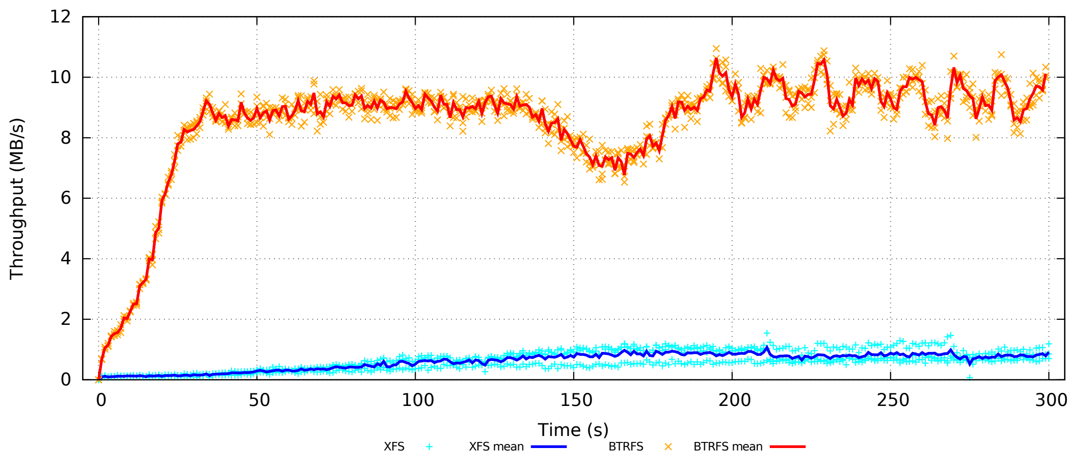


Figure 3.27: Rados bench sequential 4KB read with 16 threads.

The sequential 4KB reads with 16 threads (see Figure 3.27) showed a large throughput difference between the BTRFS and the XFS pool. While the XFS pool achieved, after a warm up phase, an average of around 1 MB/s, the BTRFS pool varied between 8 and 10.5 MB/s and averaged 9 MB/s. Both pools showed no significant differences between runs. Surprisingly, the throughput patterns are identical for all three runs on the BTRFS pool.

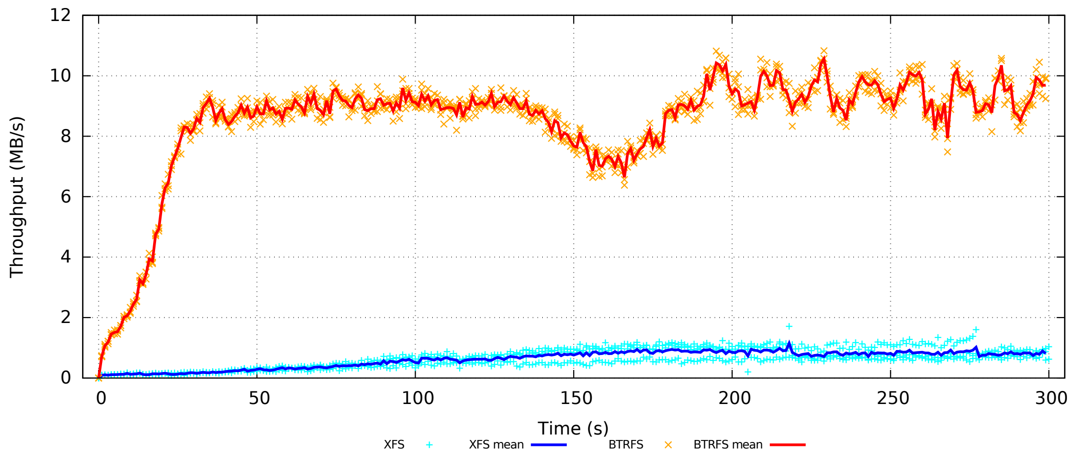


Figure 3.28: Rados bench sequential 4KB read with 64 threads.

The throughput of sequential 4KB reads with 64 threads (see Figure 3.28) is identical to Figure 3.27. Using more threads with this hardware configuration did not increase throughput on either file systems.

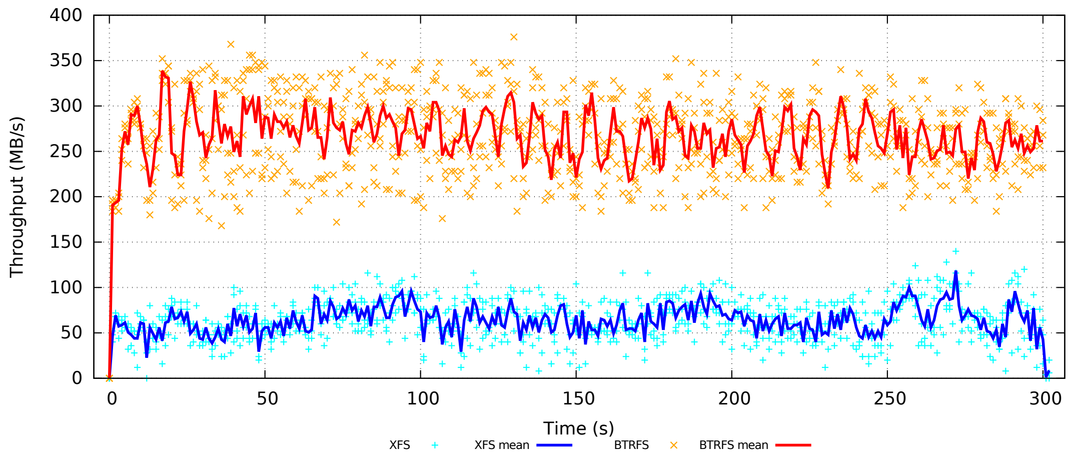


Figure 3.29: Rados bench sequential 4MB read with 16 threads.

For sequential 4MB reads with 16 threads (see Figure 3.29) the throughput graph is similar to the random 4MB reads with 16 threads (see Figure 3.26). The throughput of the BTRFS pool averaged about 275 MB/s, while the XFS pool transferred about 75 MB/s. The jitter for both pools was quite considerate and varied between 150 and 380 MB/s for the BTRFS pool, and 0 and 145 MB/s for the XFS pool.

The drop at the end of the XFS plot is attributed to the way rados bench works. When the benchmark is set to run for a specific duration, it initiates the accesses with the set thread count. If the benchmark hits the set runtime, it stops creating new access threads. The benchmark will only finish when all threads have finished. If only one access is outstanding, it is reported as a single operation in the next reporting interval.

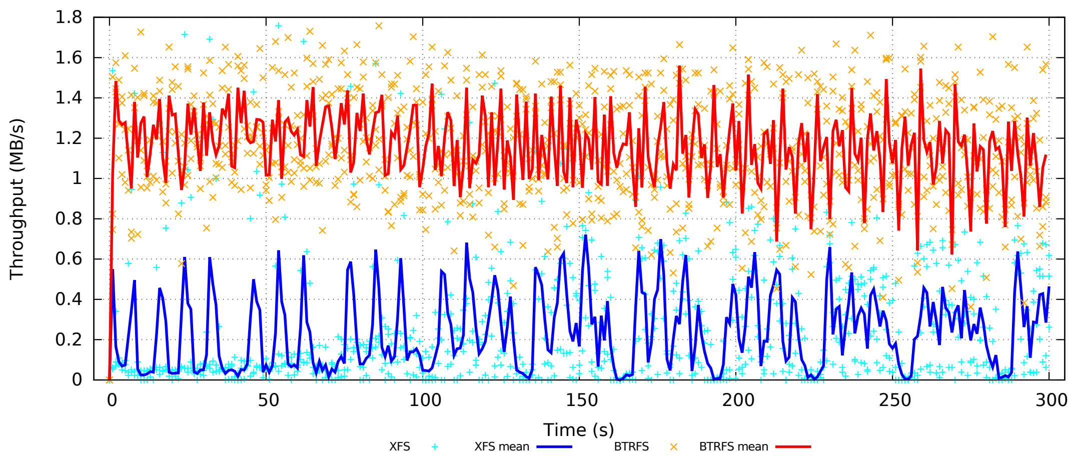


Figure 3.30: Rados bench 4KB write with 16 threads.

For 4KB writes with 16 threads (see Figure 3.30) the XFS pool achieved only 0.25 MB/s or 63 IOPS, and indeed at times zero IOPS were reported. In a production system, this would be a serious problem as it would delay all kinds of small accesses. This access pattern is typical in software compilation (an example is presented in Section 4.3.4). The BTRFS pool performed more consistently and achieved a higher throughput. Both pools showed a high variance of throughput, from run to run, but with a consistent average.

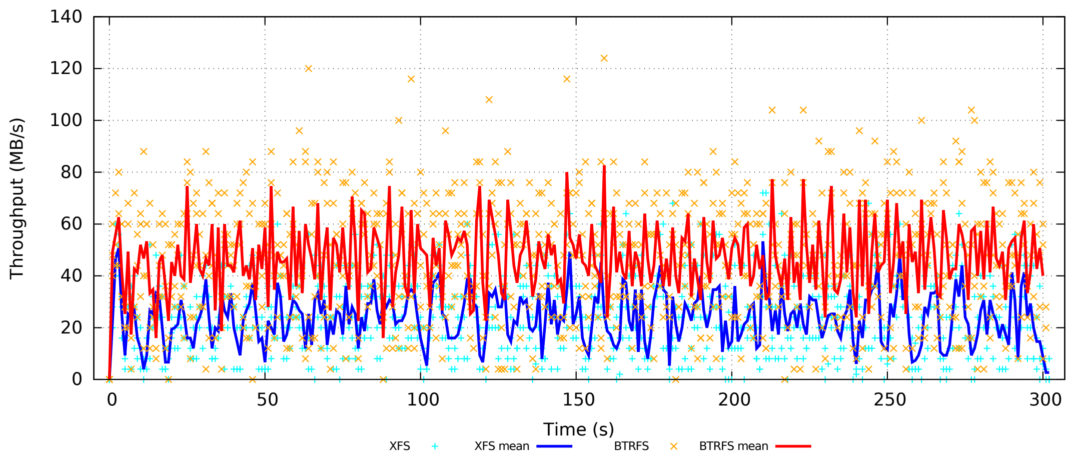


Figure 3.31: Rados bench 4MB write with 16 threads.

For the write benchmark of 4MB accesses with 16 threads (see Figure 3.31), the XFS pool achieved an average throughput of 24 MB/s and the BTRFS pool achieved an average of 46 MB/s. Again, both pools showed a high variance of throughput from run to run, with the XFS pool reporting between zero and 75 MB/s and the XFS pool reporting between zero and 125 MB/s. This behaviour resulting in a widely varying transfer speed.

The results presented here show the impact on pool performance that arise from a change of the file system used on the OSDs. With the hardware used for these tests, large differences between the different pools were observed. The BTRFS pool performed in all tested access patterns better than the XFS pool. Therefore, it is not surprising that BTRFS was selected as the future file system for OSDs when reaching maturity.

3.5.2 I/O Scheduler

The I/O scheduler is a very important component in the I/O path. It takes all requests, potentially reorders them, and passes them on to the storage device. It contains specific policies on how to reorder and dispatch requests to aid in achieving a balance between throughput, latency and priority. The service time of an individual random access may be around 10 milliseconds. In that time a modern single CPU core running at 3.0 GHz is capable of executing 30 million clock cycles. Rather than immediately performing a context switch, in which those cycles are given over to another process, it may be worth considering dedicating some of those cycles to optimise the storage access queue and to conform to the I/O strategy, before the context switch is performed. Another task of the scheduler is to manage access to a shared disk device between multiple process [113] [114].

The I/O schedulers that are shipped with a current Ubuntu Linux kernel are NOOP, CFQ and Deadline. CFQ is the default.

- The NOOP scheduler operates with a first-in-first-out (FIFO) policy. The requests are merged for a larger request dispatch but not reordered.
- The deadline I/O scheduler is a C-SCAN based I/O scheduler with the addition of soft deadlines to prevent starvation and to avoid excessive delays. Each arriving request is put in an elevator and a deadline queue tagged with an expiration time. While the deadline list is used to prevent starvation, the elevator queue aims to reorder requests for better service time. The deadline times for reads and writes are weighed differently. Read and write requests have a deadline of 0.5 and 5 seconds, respectively. Read requests are typically synchronous and therefore blocking, while write requests tend to be asynchronous and non-blocking.
- The CFQ I/O scheduler is the default Linux scheduler. It attempts to:
  - apply fairness among I/O requests by assigning time slices to each process. Fairness is measured in terms of time, rather than on throughput.
  - provide some level of Quality of Service (QoS) by dividing processes into a number of I/O classes: Real-Time (RT), Best Effort (BE) and Idle.
  - deliver high throughput by assuming that contiguous requests from an individual process tend to be close together. The scheduler attempts to reduce

seek times by grouping requests from the same process together before initiating a dispatch.

– to keep the latency proportional to the system load by scheduling each process periodically.

Changing the I/O scheduler of the host and within the virtual machine can have a significant difference in performance as shown by Boutcher *et al.* [115], while Pratt *et al.* [116] have shown the performance improvements achieved by using a different scheduler for specific workloads.

In this experiment, a combination of 24 1 TB Hitachi and Seagate hard drives were used (see Table 3.2). The drives were assigned to two separate pools with 4 drives dedicated to each pool on each host. The file system used in this experiment is BTRFS, with a replication count set to 2.

The greater Ceph environment components changed during these tests were the I/O schedulers and the queue size on each individual hard drive. The deadline and CFQ schedulers were used and each was tested with a queue size of 128 and 512 (resulting in four distinct test combinations). A longer queue size allows the disk scheduler to reorganize accesses to reduce disk head movement, at the expense of increased latency of individual requests. Depending on the workload and the number of concurrent connections, performance can be substantially improved, as shown by Zhang *et al.* [117].

The tests were performed using the rados benchmark tool with 4KB and 4MB access sizes. The runtime was set to 1200 seconds. One of the storage nodes was used to host the storage benchmark during execution. Before each run, the all operating system caches were flushed. The results show the average of the three runs. Before the read benchmark was performed, the cluster had to be populated with data to be read by the benchmark.

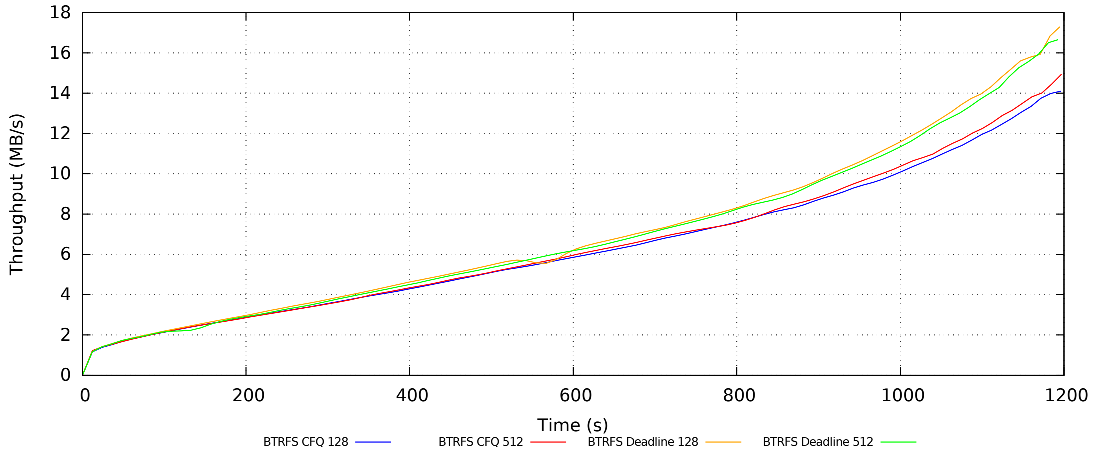


Figure 3.32: Rados bench random 4KB read with 16 threads.

For the random 4KB read benchmark (see Figure 3.32), there were slight performance differences between the two disk schedulers. The deadline scheduler performed slightly better with a throughput advantage of about 2 MB/s after 1200 seconds, which is the equivalent of 500 IOPS. Changes made to the queue size had no effect for each of the schedulers.

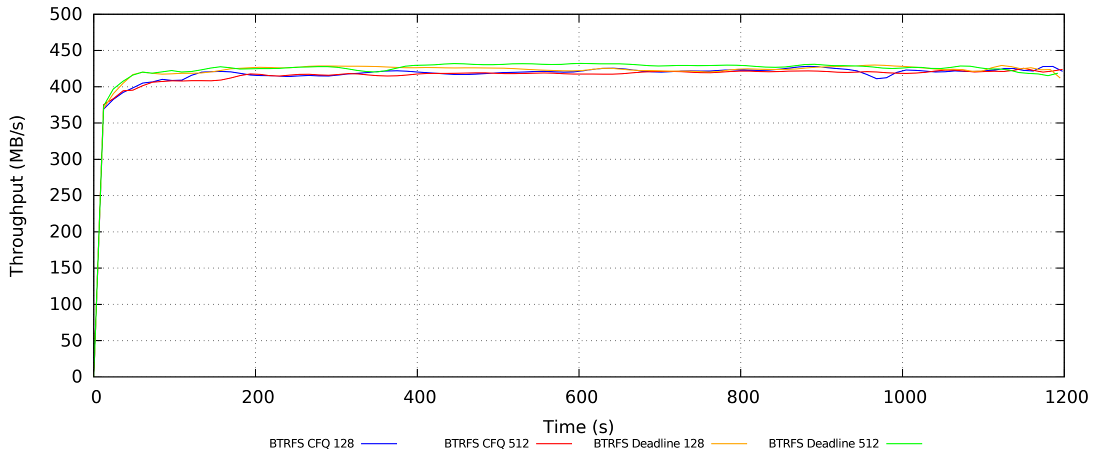


Figure 3.33: Rados bench random 4MB read with 16 threads.

For the random 4MB read benchmark (see Figure 3.33), there were no significant differences between the schedulers and their queue sizes. All combinations achieve between 124000 and 126000 transactions in 1200 seconds, which resulted in a difference of up to 2 IOPS. Such a small difference could be attributed to the resolution of the measurement process and hence is insignificant.

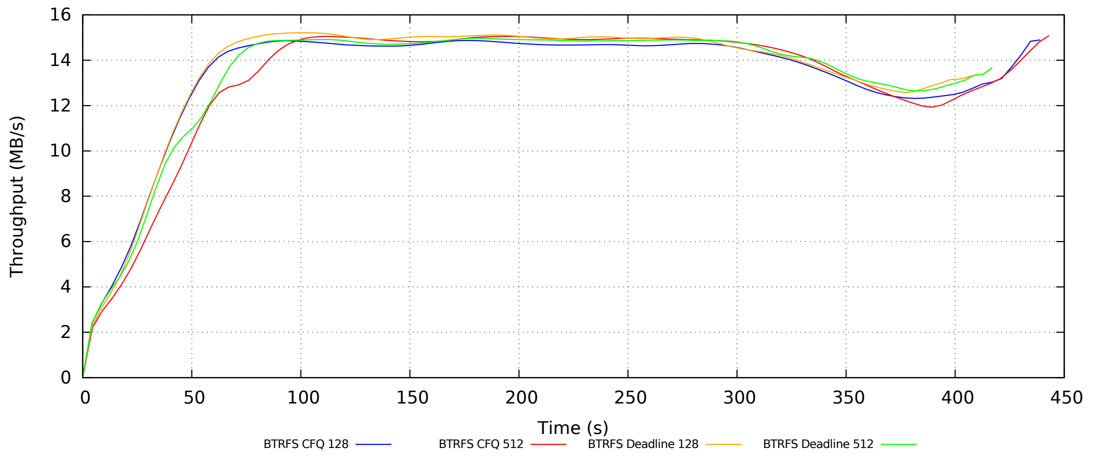


Figure 3.34: Rados bench sequential 4KB read with 16 threads.

The sequential 4KB read benchmark (see Figure 3.34) showed the same pattern for all scheduler and queue size combinations. The deadline scheduler performed around 35 IOPS better than the CFQ scheduler. The deadline benchmarks had about 1.4 million objects on the disks, whereas the CFQ benchmarks were able to access 1.5

million objects. The difference in the benchmark durations results from the fact that all data was read only once and that for the deadline scheduler benchmarks there was insufficient data on the disk to be read before the benchmark duration expired.

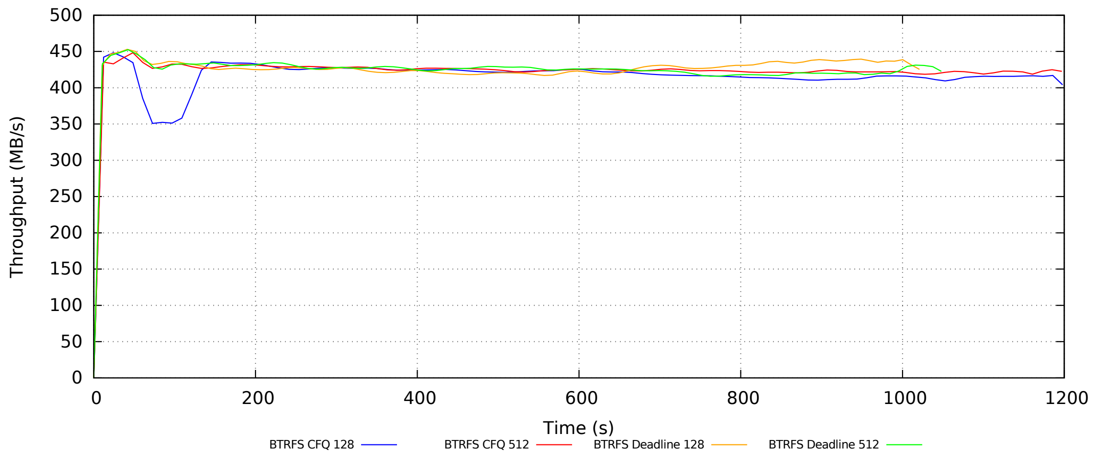


Figure 3.35: Rados bench sequential 4MB read with 16 threads.

During the sequential 4MB reads with 16 threads (see Figure 3.35), there were no significant differences between the different scheduler combinations. All four combinations achieved a throughput of around 420 MB/s. The difference in runtime between the configurations is attributed to the difference between the reading and the writing throughput of the cluster, and the dearth of data to be read. The scheduler combination of CFQ and 128 queue size showed an anomaly for around 60 seconds. During that time the throughput dropped as low as 136 MB/s. The cause of this temporary drop in performance is unclear, but its occurrence in a single run indicates an external effect, such as network traffic or increased CPU load, unrelated to the disk scheduler.

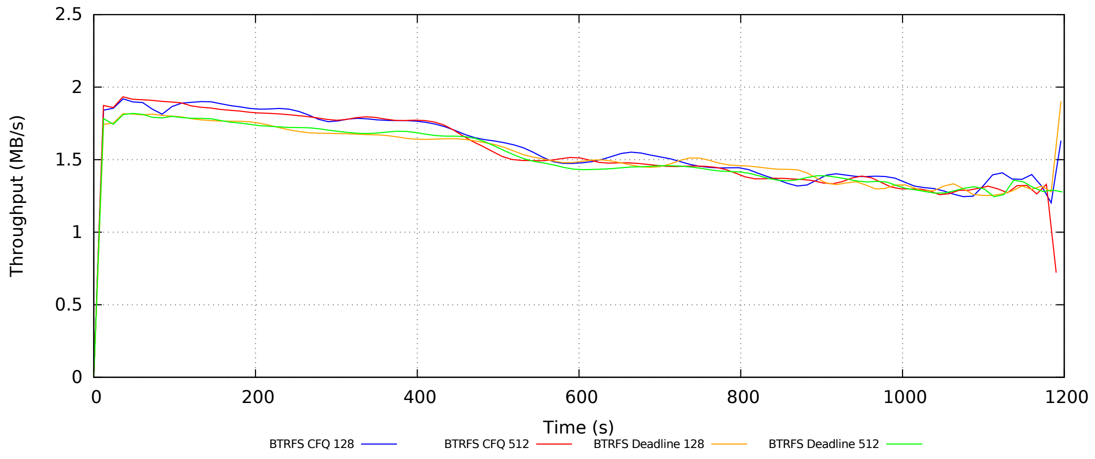


Figure 3.36: Rados bench 4KB write with 16 threads.

For the 4KB writes with 16 threads (see Figure 3.36), all four combinations achieved similar throughputs, ranging from 382 (Deadline 512) to 403 IOPS (CFQ 128). The difference between the slowest and the fastest configuration was around 5.5%.

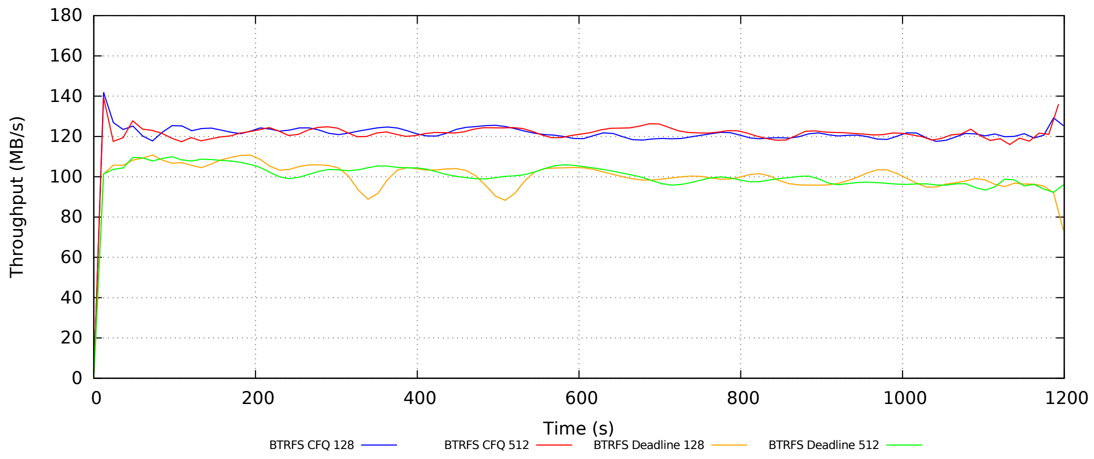


Figure 3.37: Rados bench 4MB write with 16 threads.

The throughput for the rados bench 4MB write benchmark (see Figure 3.37) showed a substantial throughput increase when using the CFQ scheduler. Using the CFQ scheduler resulted in an average throughput of 120 MB/s, while the deadline scheduler achieved 100 MB/s. The size of the scheduler queue did not improve or change the throughput.

The results of the benchmarks using different disk I/O schedulers and queue sizes shows that choosing a specific scheduler can have an impact on performance of a Ceph cluster. The effect can vary depending on the hardware being used, as storage controllers and storage devices differ. The CFQ scheduler is designed to perform best with mechanical hard drives, which is confirmed in these tests. When using SSDs the Deadline or NOOP scheduler is recommended, as it performs better with flash drives [118].

Chapter 4

Workload Characterization

Workload characterization is an important part of performance evaluation. Performance evaluation is a basic tool of experimental computer science for comparing different designs, different hardware architectures and/or systems, and measuring the effect of tuning system or component configurations. It also provides a means to properly assess hardware requirements of production system to meet expected performance goals and targets.

The main factors that influence the performance of a system are the design, the implementation and the workload. The design and the implementation of software can be relatively understandable, as is software architecture, computing architecture and computer hardware, but understanding and modelling workloads is more difficult.

Unfortunately, performance evaluation is often done in a GIGO (garbage-in-garbage-out) fashion [119]. In such evaluations, systems are evaluated with workloads that do not reflect the typical system workload. For sorting algorithms performance is measured in runtime and reported as $O(n \log n)$. For pre-sorted datasets, the runtime can be much shorter, whereas reversely ordered datasets can increase duration to $O(n^2)$. It is therefore important to evaluate a system with representative workloads.

Using the correct workload is also crucial for evaluating complex systems, such as hardware-software combinations. Workloads can be characterized based on their impact on CPU, memory, network and/or disk I/O.

4.1 Storage Workloads

One of the most important components for storage workloads is the file system. It is responsible for safely storing files on the physical disk, ordering and tracking the used physical blocks, the file size and other metadata, such as file owner, creation- and modification time. The way that the file system handles this informations and the structure that it uses to store and retrieve information is vital and can have a significant impact on performance [120]. One file system may handle small sized sequential consecutive accesses well, for example, while another may not.

Files stored on a server or desktop typically vary in number and size. For example, a Linux based operating system may contain files with a file size of zero, representing symbolic links to other files or files in the virtual file system, such as `/proc` and `/sysfs`. Files with a file size of a couple of bytes are often used by the operating system to write the ID of a process at runtime, as used in `/run`.

Figures 4.1 and 4.2 show the file size distributions from a Linux workstation and a Linux server. The Linux workstation contains multiple virtual machine images, CD images, pictures and text files. The Linux server hosts a number of software servers including a MySQL server, a Puppet server and a file server.

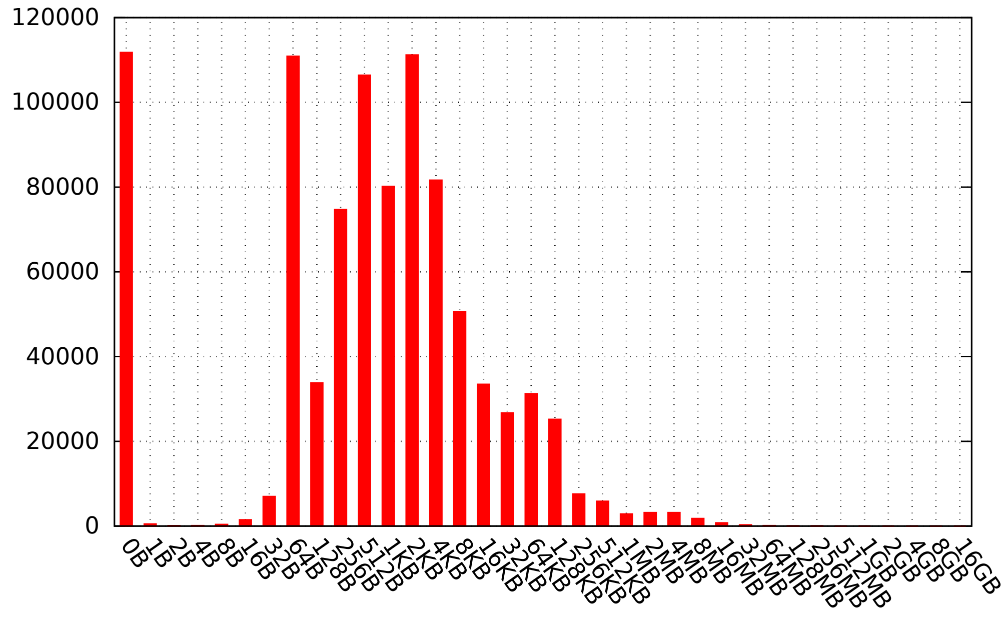


Figure 4.1: File size distribution on a Linux workstation.

The distribution of file sizes from small to large will have an important impact on the file system. Small files can lead to file system fragmentation, which impacts performance, this is being exacerbated by the increase in disk block sizes over time to complement the growing disk sizes. Current mechanical hard drives use 4096 Byte (4K) blocks instead of the traditional 512 Bytes to improve sector format efficiency (88.7% to 97.3%) and error correction coding (ECC).

The total space taken up by small files can be significant. It was always assumed that there will be a shift to larger sized files, due to the increased consumption of media files (audio, video, pictures), but file sizes have only slightly increased over the years [121] [122]. Figure 4.3 shows the cumulative file size allocation on the two example hosts. Both observed machines have a similar file size distribution to the ones mentioned

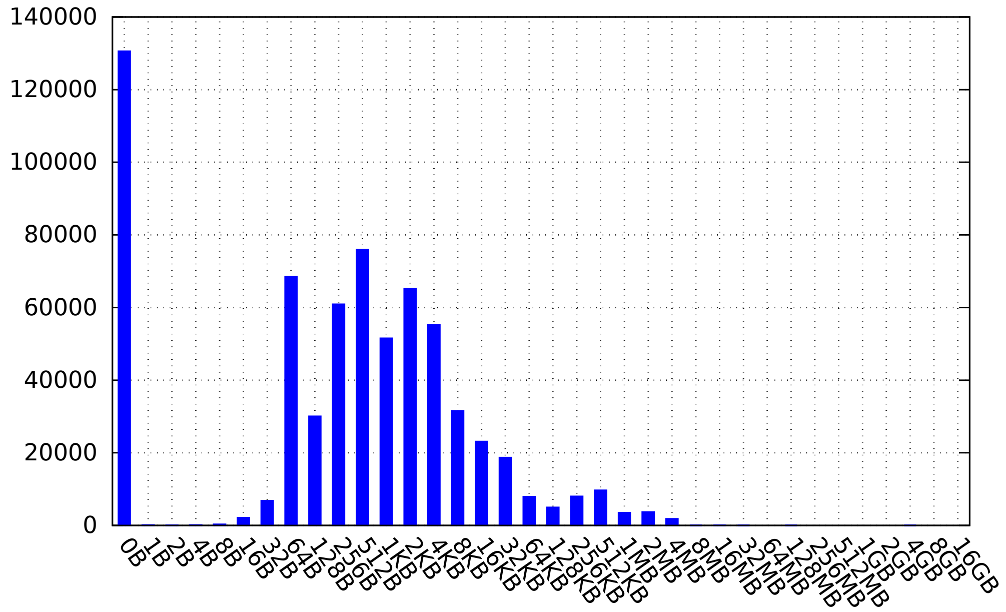


Figure 4.2: File size distribution on a Linux server used for various services.

by Agrawal *et al.* [121] and Tanenbaum *et al.* [122]. The Linux server contains more zero sized files, while the Linux workstation contains more large files. These large files are fewer in number but account for most of the space used (see Figure 4.4). The total disk space used on the Linux server was 34.70 GB (660712 files), while the space taken up on the Linux workstation was 184.84 GB (912660 files). The largest file on the Linux server was a 4 GB virtual machine image, while the largest file on the Linux workstation was a 20.5 GB virtual machine image. Furthermore, the workstation contained media files (*e.g.*, audio, video, pictures), multiple virtual machine images (>4GB) and multiple Linux ISO files.

4.2 Traces

Storage workload characterization for applications is a process that is very common in the enterprise sector. It allows the identification of the distinct access patterns of an application and enables the administrator to optimise the storage system to support the application. Knowing the applications access pattern can help in identifying bottlenecks in the storage subsystem and can improve performance by tuning the system for the specific application characteristics. Setting the correct stripe size in a RAID set, for example, can improve performance for specific applications [123]. Some application vendors might give recommendations for storage system configurations and/or application access patterns. In case such information is not provided, the system administrator has to profile the application to acquire the necessary information.

The following subsections explore a number of ways for extracting trace informations

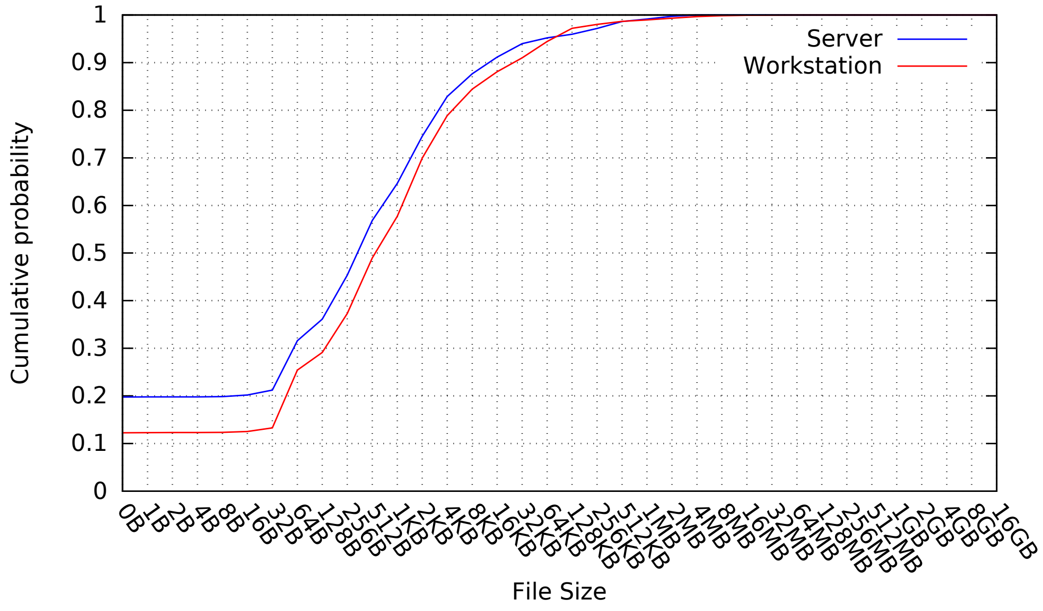


Figure 4.3: Cumulative file size distribution on server and workstation.

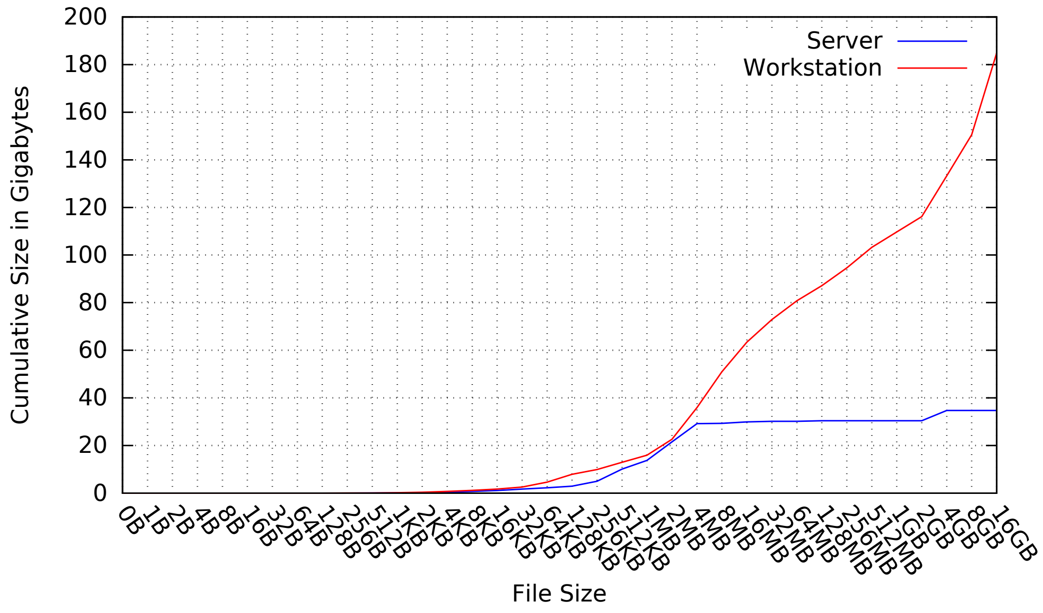


Figure 4.4: Cumulative file size distribution on server and workstation.

from applications. These traces are taken from different layers of the respective systems and platforms.

4.2.1 VMware ESX Server - vscsiStats

VMware ESX Server is a modern hypervisor for managing virtual machines. Each virtual machine is securely isolated and acts as though it were running directly on dedicated hardware. The devices presented to the virtual machine, such as network interfaces or storage devices, are virtual devices. These virtual devices are hardware-agnostic, which make it possible to migrate VMs to a different host with a different hardware configuration. This would not work if a physical device had been directly passed through to the VM.

VMware has implemented a streamlined path for the hypervisor to support high-speed I/O for the performance critical devices network and storage. As shown in Figure 4.5, the hypervisor presents the virtual machine with an emulated network and SCSI storage device (depicted in gray). The calls from these devices are then sent to the NIC and storage driver of the ESX server and subsequently to the physical device. The storage driver emulation presents either an LSI Logic (parallel SCSI or SAS), Bus Logic, IDE, SATA or VMWare Paravirtual SCSI (PVSCSI) device [124] to the VM. The NIC is presented as either a VMware VMXNET3 device, a paravirtualized NIC designed for performance, an Intel e1000 or an AMD 79C970 PCnet32 Lance device [125]. The default template settings in a Linux VM deployed on a VMware ESXi 6.0 host are shown in Listing E.9.

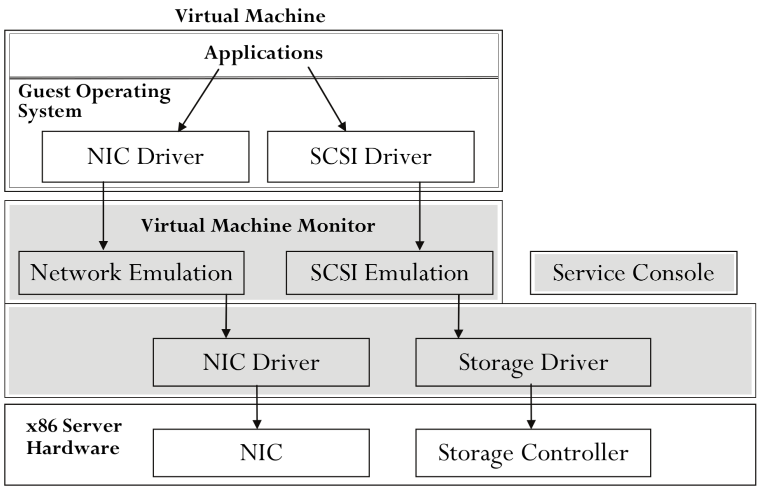


Figure 4.5: VMware ESX Server architecture. ESX Server runs on x86-64 hardware and includes a virtual machine monitor that virtualizes the CPU [126].

As the devices are emulated, it is possible for VMware to extract information on the I/O calls on a host, device and VM basis using vscsiStats. When a VM is configured with multiple disks, these disks can be monitored individually or as a group. This is done with a minimal penalty on performance.

To use vscsiStats, it is necessary to get ESX shell access on the ESX host. For security reasons, this feature is disabled by default, and has to be activated if needed. In the case where the host is accessed remotely, the SSH server will also be required; this is also disabled by default for the same reason.

Starting an I/O trace requires a worldGroupID and a handleID. The worldGroupID represents the virtual machine and the handleID represents the virtual disk. These get these IDs, the command `vscsiStats -l` is used. A sample output from a server running multiple VMs is shown in Listing E.10.

In this listing two virtual machines (Ceph_profiling, vmware-io-analyzer-1.6.2) with 8 disks in total are identified. The worldGroupID is static, whereas the handleID is incremented each time the VM reboots. Care therefore needs to be taken to ensure proper attribution of traces after a VM reboot.

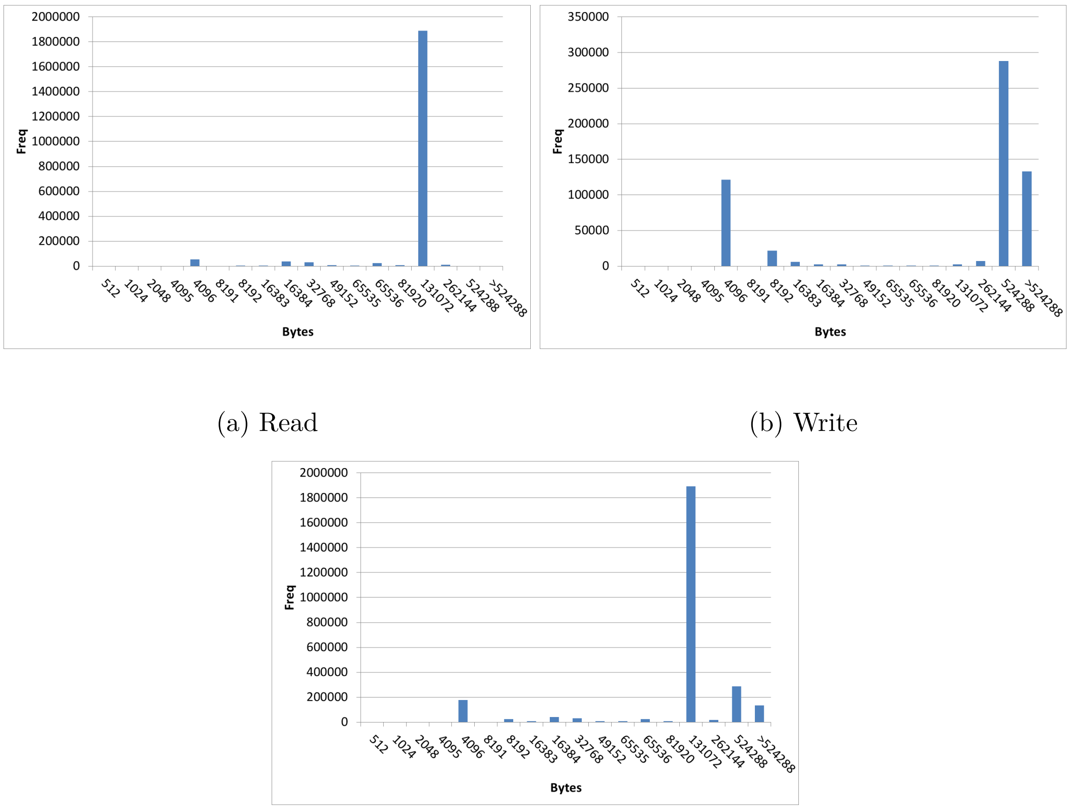
vscsiStats can be used in two different ways: online and offline. These will be considered in more detail in the following subsections.

4.2.1.1 Online Histogram

vscsiStats can be used in an online mode, which takes the trace information and creates histograms of a number of metrics, such as spatial locality, I/O length, interarrival, outstanding I/O and latency distribution. This mode will not record individual I/Os and their positions. The histograms created may be sufficient for use case analysis, however, the absence of time serious information may be an impediment to identifying comprehensive performance enhancements. Some of these histograms, such as latency, represent the performance of the VM on the current host under a specific load. A faster storage backend will result in lower latencies for the same workload. Therefore, some results should be contextualized to the underlying hardware and can not be compared across different hardware infrastructures. In contrast, results that are not tightly coupled to the underlying hardware, such as access sizes, can legitimately be compared.

To get trace information, the tracing tool has to be started for one VM and one (or multiple) virtual disk(s). The histogram can be printed to the console or saved into a comma separated file (see Listing E.11). The histogram counters are continuously increased until the trace is stopped and reset.

The trace results are presented separated for reads (see Figure 4.6a) and writes (see Figure 4.6b) and as a combination of both (see Figure 4.6c). The trace file can be opened with a text editor or in Microsoft Excel, where it can be processed by a macro (by Paul Dunn [127]), to process the data to create individual plots.



(a) Read (b) Write

(c) Total

Figure 4.6: I/O length distribution for a Postmark workload, separated into reads (a), writes (b) and a combined of both (c).

4.2.1.2 Offline Trace

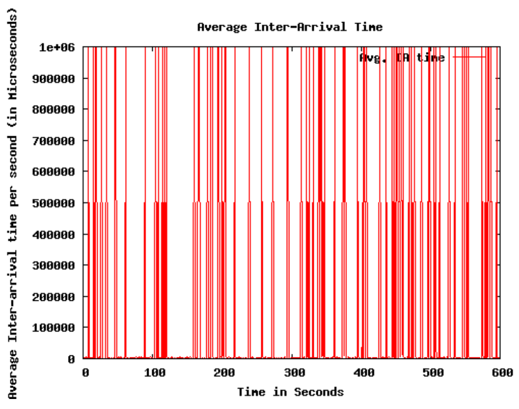
The second operation mode of vscsiStats is the offline mode. This mode allows for a more detailed analysis, but is limited in duration, if stored in the file system root, due to space limitations on the VMware ESXi host. In the root directory the maximum number of traced I/Os appears to be around 830000 or 33MB. Depending on the application and storage system, this number of I/Os may be achieved before a comprehensive trace of the application can be captured. To mitigate this limitation, an alternative location on the datastore may be used to store the trace, thus ensuring that it is not prematurely terminated.

It is possible to run the trace in combination with gzip to reduce the trace file size. This approach works well, but the file has to be decompressed before it can be decompiled. Using the decompilation process directly on the archive will result in corruption.

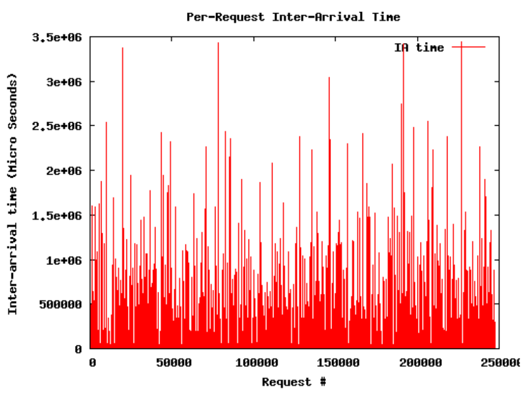
The command sequence to start a full trace is similar to that of starting the histogram trace. The starting command requires an extra trace option. This will create a trace channel that can be recorded by the `logchannellogger`. Traces are recorded in a binary format and can be converted so that they are human-readable. To convert the binary file into a comma separated file, a vscsiStats command is used. The output can be send either to the `stdout` or directly piped into a file. The full process is shown in Listing E.12.

4.2.1.3 VMware I/O Analyzer

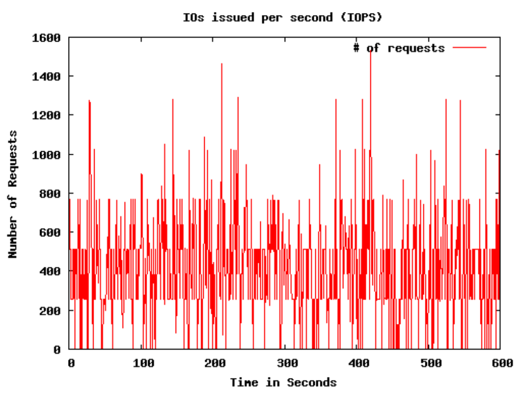
VMware also provides the VMware I/O Analyzer appliance. It provides a web interface to upload the offline trace file and to create a set of plots. The plots generated present the average inter-arrival time (Figure 4.7a), per-request arrival time (Figure 4.7b), IOs[1] issued per second (Figure 4.7c) and logical block number (LBN) distribution (Figure 4.7d) for the location on the disk.



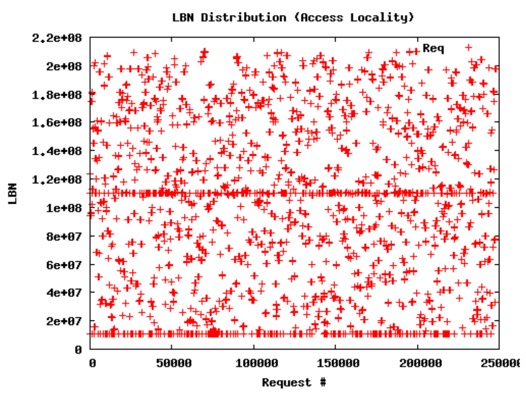
(a) Average Inter-Arrival Time



(b) Per-Request Inter-Arrival Time



(c) IOs issued per second (IOPS)



(d) LBN Distribution (Access Locality)

Figure 4.7: Offline trace plots created by the VMware I/O Analyzer. The I/O workload is a 600 seconds random 32MB read `rados bench` run. The trace is showing the load on a single disk (`/dev/sdb`) of a five disk Ceph cluster.

[1]Note that in the literature IOs and I/Os tend to be used interchangeably.

4.2.2 Other Tracing Tools

The VMware tracing tools in this study, however a myriad of other tools exists for different hardware types and operating systems. These include the Windows Performance Analyser and Recorder, the IBM System Z and low level operating system tools, such as `ioprof` and `strace`. Details of these tools and examples of their used is detailed in Appendix D.

4.3 Application Traces

As noted above, the VMware vscsiStats tool is used in this dissertation to generate application traces to aid in the workload characterization process. This tool was chosen because it offers both a high-level view and insights into individual storage accesses. The host does not require a client component or agent to be installed on the virtual machine, making it agnostic to the operating system running on that virtual machine. Five applications (Blogbench, Postmark, dbench, Kernel compilation and pgbench) were chosen as representative cloud workloads and these were traced and analysed in an attempt to determine their storage access characteristics. These characteristics will subsequently be mapped to appropriate Ceph configurations, as described in Section 2.2.4, in an attempt to improve their performance over execution on the default Ceph configuration (*i.e.*, the benchmark). A description of the workloads and their associated characteristics is given in the following sections.

Of these only the read and write I/O length and seek distance are subsequently used in the mapping process. Interarrival latency gives insight into the sequencing of read and write accesses and as such exposes important access patterns that can be leveraged for improvement. Exploiting outstanding I/Os in the improvement process introduces complexity beyond the scope of this dissertation. Nevertheless, an exploration of outstanding I/Os for the Blogbench workload is given to illustrate this workload characteristic. Outstanding I/O characteristics for the remaining workloads are not presented since the characteristic is not used in the mapping process. Even though interarrival latency is also omitted from the mapping process, an analysis of this characteristic for each workload is presented for the insight it gives into the workload. A proper investigation of outstanding I/Os and interarrival latency is deferred to future work.

4.3.1 Blogbench read/write

Blogbench is itself a portable file system benchmark that tries to reproduce the load of a busy file server [128] [129] [130]. The tool creates multiple threads to perform reads, writes and rewrites to stress the storage backend. It aims to measure the scalability and concurrency of the system.

It was initially designed to mimic the workload behaviour of the French social networking site Skyblog.com, which is known today as Skyrock.com. This site allows users to create blogs, add profiles and exchange messages with each other.

The benchmark starts four different thread types concurrently:

Writers create new blogs/directories, filled with a random amount of fake articles and pictures.

Re-writers add or modify articles and pictures of existing blog entries.

Commenters add fake comments to existing blogs in a random order.

Readers "read" blog entries and the associated pictures and comments. Occasionally, they try to access nonexisting files.

According to the documentation, blog entries are written atomically. The content is pushed first with 8KB chunks to a temporary file that is renamed when the process finishes. 8KB is the default write buffer size for PHP. Reads are performed using a 64KB buffer size.

Concurrent writers and rewriters can quickly result in disk fragmentation. Every blog is a new directory under the same parent. This can cause problems if the file system (like UFS [131] [132]) is not capable of handling a large number of links to the same directory. Therefore, the benchmark should not be used for long durations on systems with file systems having this limitation.

During the trace, the system showed a high CPU utilization of around 100%. This means the workload is CPU bound, masking any storage system limitations.

4.3.1.1 I/O length

When run, blogbench created a total of 122338 I/O accesses to the storage system. Of these 23436 (19.2%) were read accesses and the other 98902 (80.8%) were write accesses. The tool ran for about 345 seconds.

The total I/O length distribution (see Figure 4.8) shows that about 38.2% of the accesses were 4KB in length. 22.9% of the accesses were 8KB, while 16383 Bytes, 16KB and 32KB were 8.3%, 5.8% and 10.4%, respectively. The average I/O length was 48715 Bytes.

When looking at the distribution of read accesses (see Figure 4.9), 4KB and 8KB I/O patterns made up 55% of the total. Of the remainder, two groups of 16KB accesses summed to 19.2% and 14.8% of the accesses were 32KB in size. Block accesses with more than 32KB were much less common and the corresponding rate was less than 10.5%. The average I/O length was 16046 Bytes.

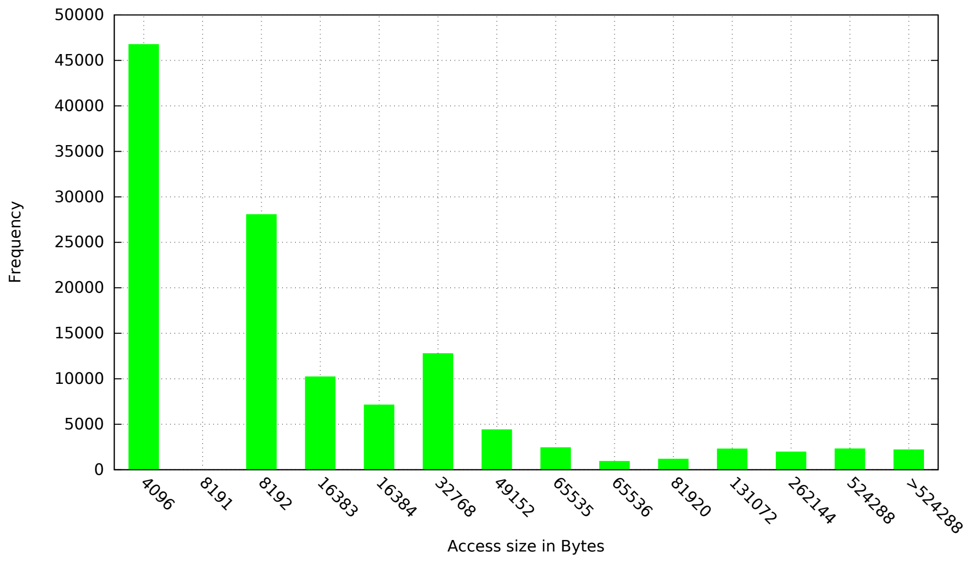


Figure 4.8: Blogbench total I/O length.

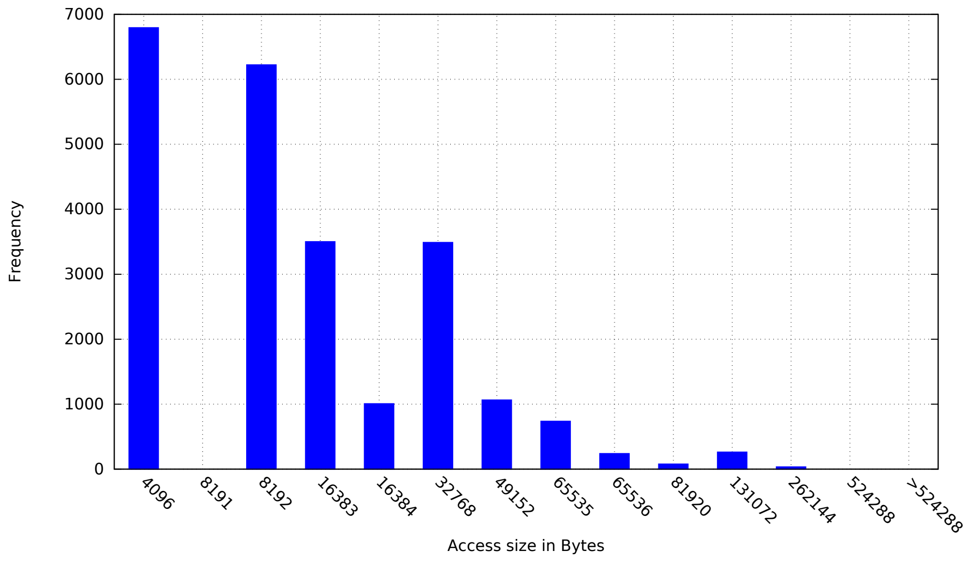


Figure 4.9: Blogbench read I/O length.

The write accesses I/O length distribution (see Figure 4.10) mostly consist of 4KB (40.3%) and 8KB (22%) accesses. Accesses with bigger block sizes were also present in the trace. The reason for this is the pictures that are added to the blogs which have a variable file size. The average I/O length was 56456 Bytes.

Overall, Blogbench uses mainly small block accesses with 4KB and 8KB. I/O lengths of 64KB, as mentioned in the test description, are not very common. As a blog is mainly for serving information rather than storing it, the reading component is more important. Due to the writing part being small I/O access length intensive, a storage configuration that can deal with small accesses would benefit the application most.

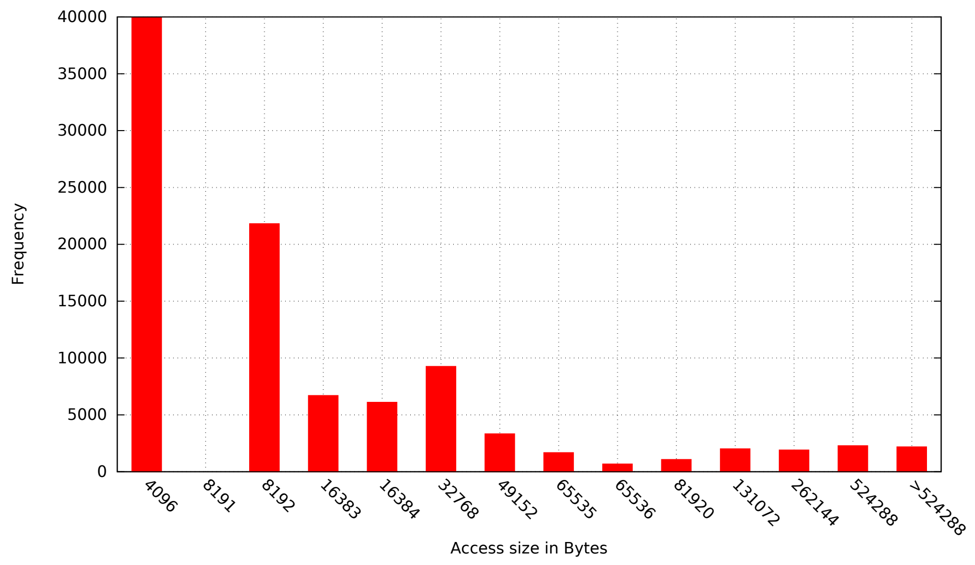


Figure 4.10: Blogbench write I/O length.

4.3.1.2 Seek Distance

The seek distance between the accesses shows whether the application is accessing blocks in a sequential or in a random fashion. The more the seek distances are focused in the centre of the graph, the more likely the accesses are sequential. Accesses further away from the centre indicate random access that require the disk head to move between accesses.

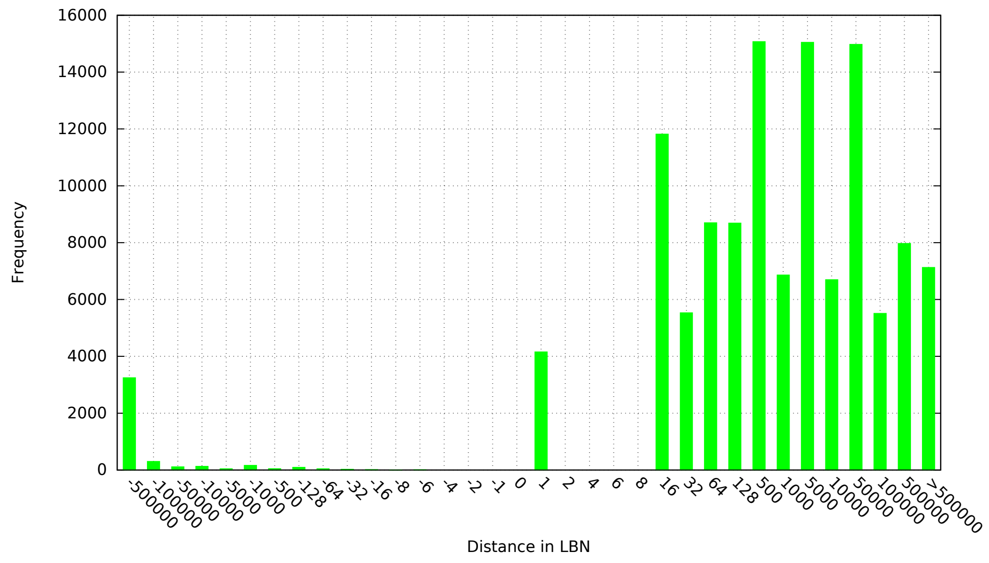


Figure 4.11: Blogbench overall distance.

The analysis of the seek distance of the Blogbench tool (see Figure 4.11) shows an access pattern that contains more random accesses than sequential. Accesses heavily lean in one direction only. Backward seeks represent only 3.5% of all seeks.

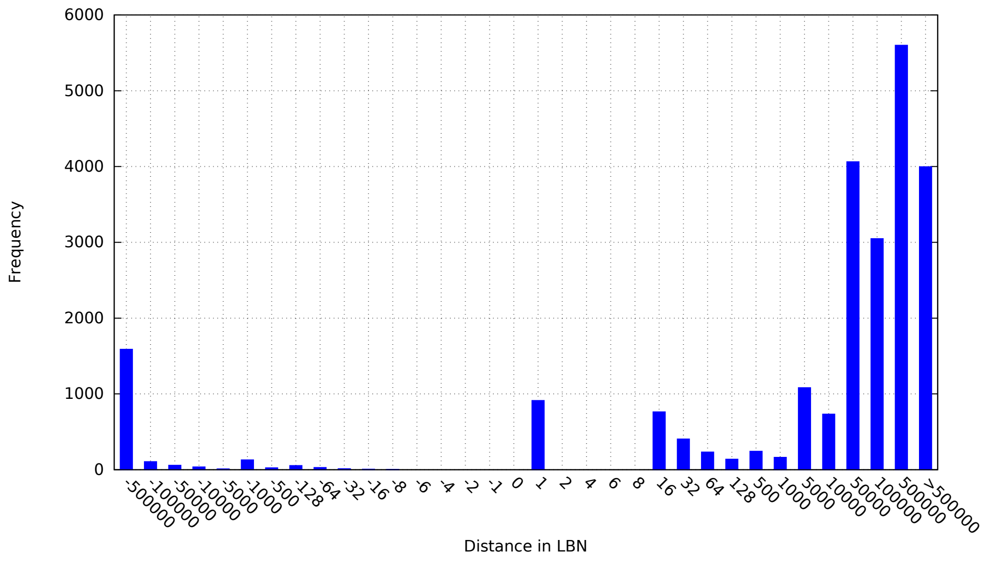


Figure 4.12: Blogbench read distance.

Seeks during reads (see Figure 4.12) are located on the sides of the graph, which indicates a random access behaviour. Of the 23436 read accesses, only 913 are made to successive blocks, representing less than 5% of the total.

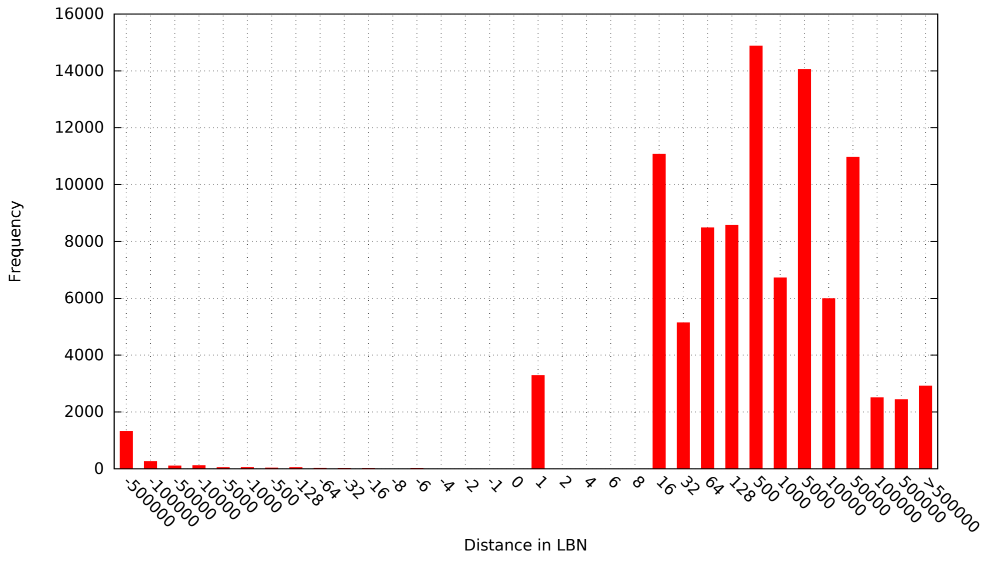


Figure 4.13: Blogbench write distance.

Write accesses (see Figure 4.13) also display random behaviour. Sequential accesses are 3.3% of the total.

Overall, the Blogbench workload show random read and random write access patterns. A configuration improving performance for random reads and writes would be beneficial to support such a workload.

4.3.1.3 Outstanding I/Os

There are multiple queues in the storage I/O path (see Figure 4.14). Each aims to increase storage performance. The operating system contains the I/O scheduler with a specific queue size. The I/O scheduler uses this queue to reorganize and optimize disk accesses to increase performance. The ordered I/Os are subsequently sent to the storage controller. Depending on the controller model, it may contain a storage queue. The storage controller queue size varies between different models. The hard drive also contains a queue. This maximum disk queue size is referred to as the queue depth and this depends on the interface used (SATA: 32; SAS: 254).

The hard drive reorganizes incoming I/Os in an attempt to increase performance by reducing disk head movements. For SATA devices this feature is called Native Command Queueing [133].

The reported outstanding I/Os by vscsiStats indicate the number of I/Os in the storage device queue at the time a new operation is passed on to the storage device. The efficiency of NCQ depends on the number of items in the device queue at a particular that can be reordered efficiently.

Hardware vendors typically specify the random read and write IOPS for storage devices when tested with 32 operations in the disk queue. The performance for fewer operations in the queue are rarely mentioned, but important to understand the storage devices characteristics.

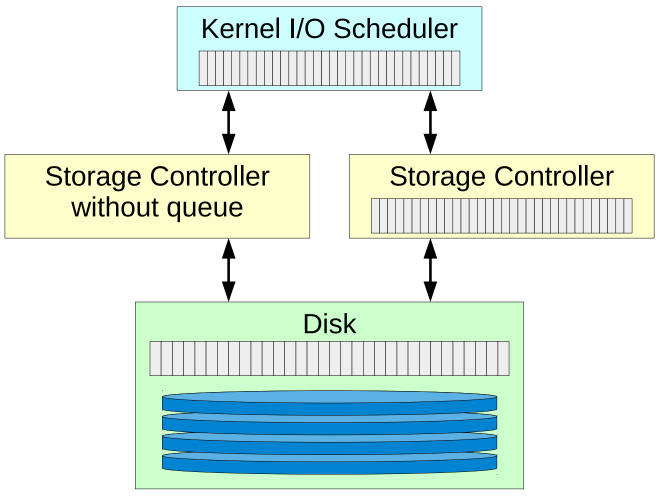


Figure 4.14: Different queues in the I/O path, including the operating system I/O scheduler queue, potential storage controller queue and disk queue.

The queue size on a Linux device can be interrogated with the command shown in Listing E.13.

The output on the tested host was 32, which means the host will never dispatch more than 32 I/Os to the storage device queue.

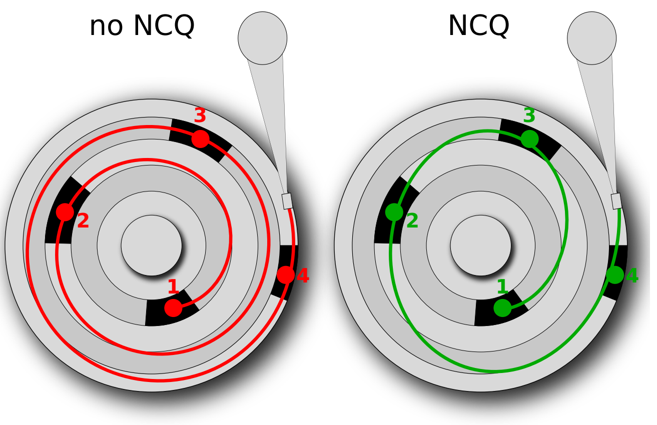


Figure 4.15: Native Command Queueing to improve disk performance by reordering I/Os [134].

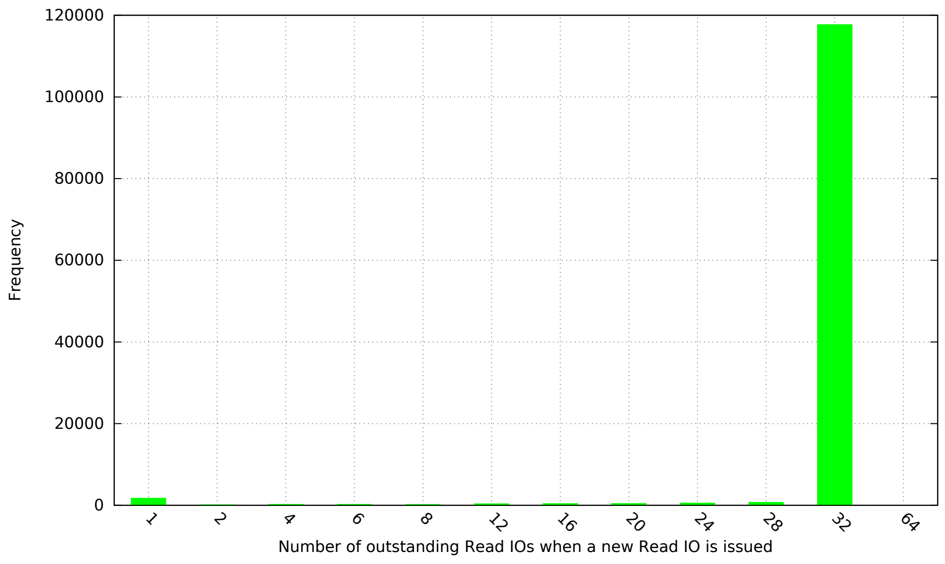


Figure 4.16: Blogbench total outstanding IOs.

The overall outstanding I/O chart (see Figure 4.16) shows a dominant peak at 32. The average number of outstanding I/Os, 30, is close to the maximum queue depth.

For read accesses (see Figure 4.17), the average number of items in the queue was 15. Of the 23436 read I/Os, 17300 (73.8%) were executed when the queue contained between 12 and 24 I/Os.

For write accesses (see Figure 4.18), the number of operations in the queue was 32 for 85% of the I/Os, resulting in an average queue occupation of 29. An occupation of less than 8 was observed for less than 1% of the write accesses.

The Blogbench workload was able to keep the device queue mostly filled. Consequently, the potential for disk head optimization could be maximized. This is important, since, as shown in Section 4.3.1.2, the workload exhibits a predominantly random access

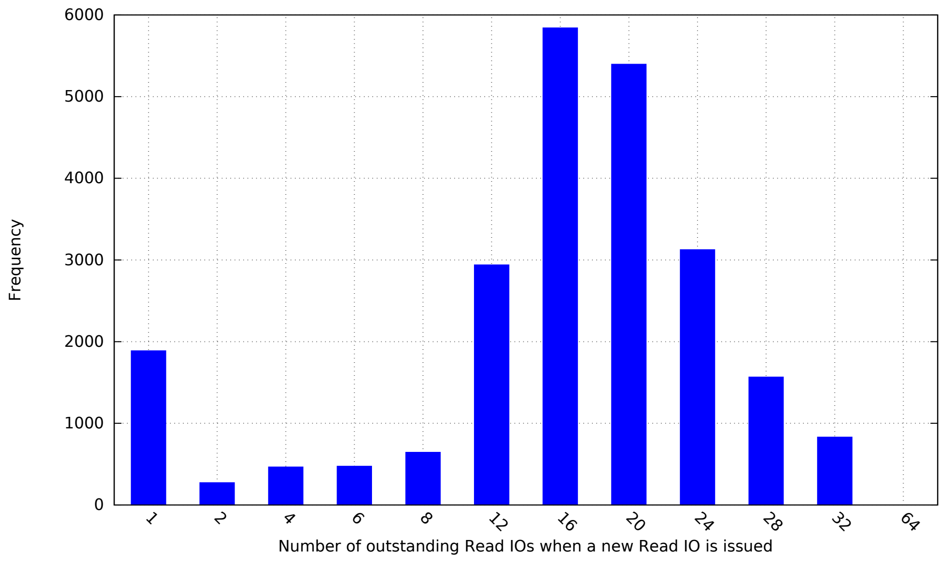


Figure 4.17: Blogbench read outstanding IOs.

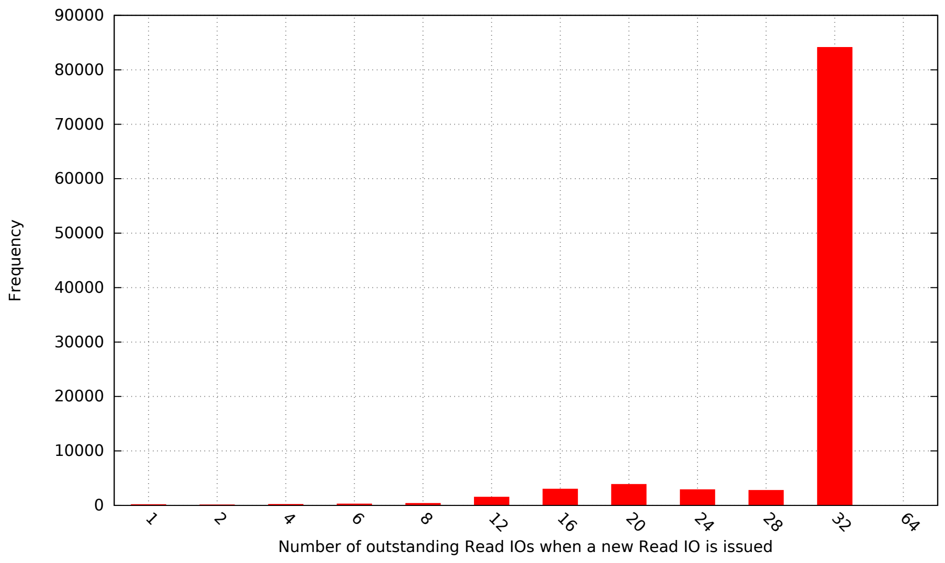


Figure 4.18: Blogbench write outstanding IOs.

pattern.

4.3.1.4 Interarrival Latency

The interarrival latency shows whether an application has a steady or a bursty access pattern. A histogram is not sufficient for this analysis, since it does not reveal the behaviour over time. The offline trace can be used to obtain detailed time serious data which can be used to augment the histogram information.

The overall interarrival latency histogram for the Blogbench workload (see Figure 4.19) shows over 48% of the I/Os have an interarrival time of up to 1 ms, 83% are under 5

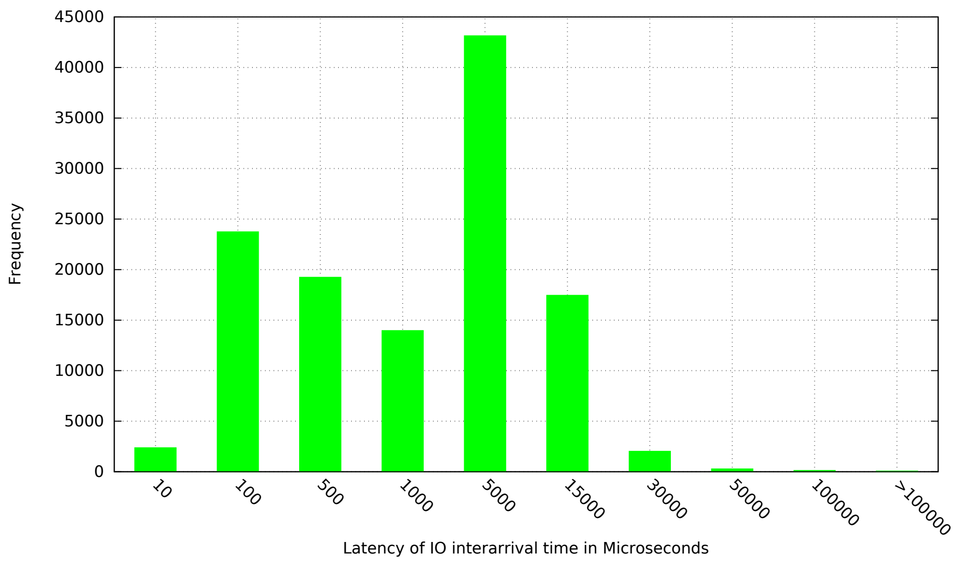
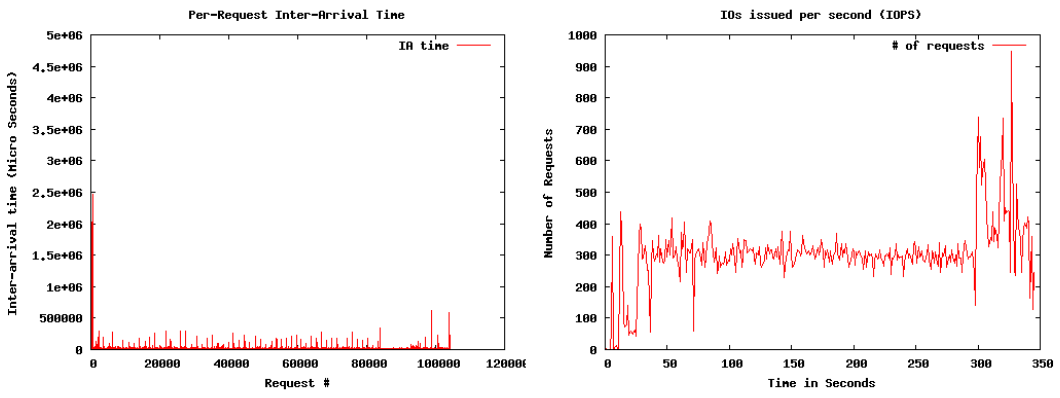


Figure 4.19: Blogbench overall interarrival latency.

ms and 2% of the interarrivals are above 15 ms. The offline interarrival time chart (see Figure 4.20a) shows that the application has a steady load. The high latency at the beginning of the plot is related to the ramp up process of the workload. In contrast, the IOPS chart (see Figure 4.20b) shows a steady load of 300 IOPS for most of the trace, but also a bursty behaviour at the end of the trace with peaks of over 900 IOPS.



(a) (b)

Figure 4.20: Blogbench overall interarrival time (a) and IOPS (b).

For the read accesses (see Figure 4.21), 57% of the I/Os arrived in under 1 ms and 89.5% in under 5 ms.

The histogram of the write accesses (see Figure 4.22) shows 45% of the I/Os arrived in under 1 ms and 80.4% under 5 ms. While the interarrival time diagrams for read and write access look similar, there are some differences. 12.5% of the read accesses have

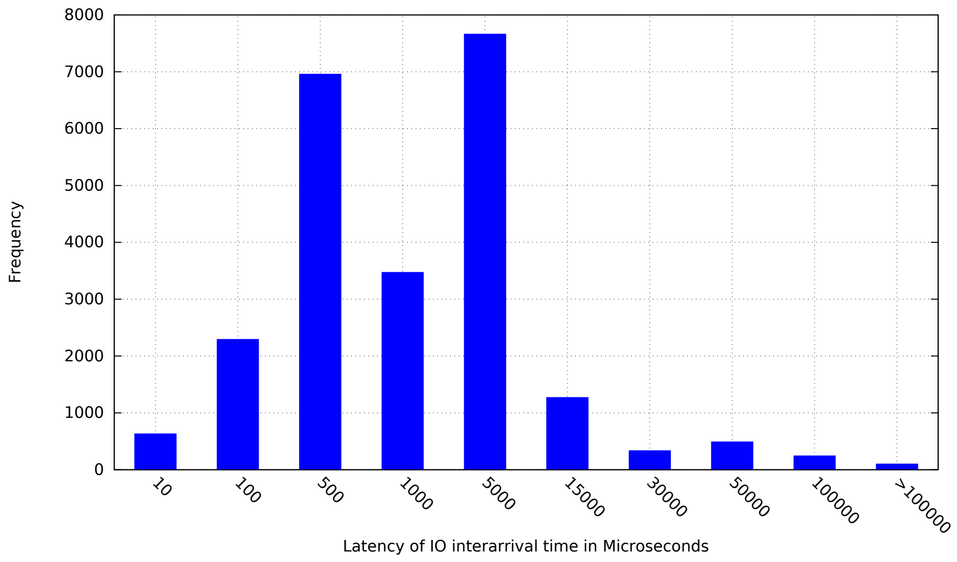


Figure 4.21: Blogbench read interarrival latency.

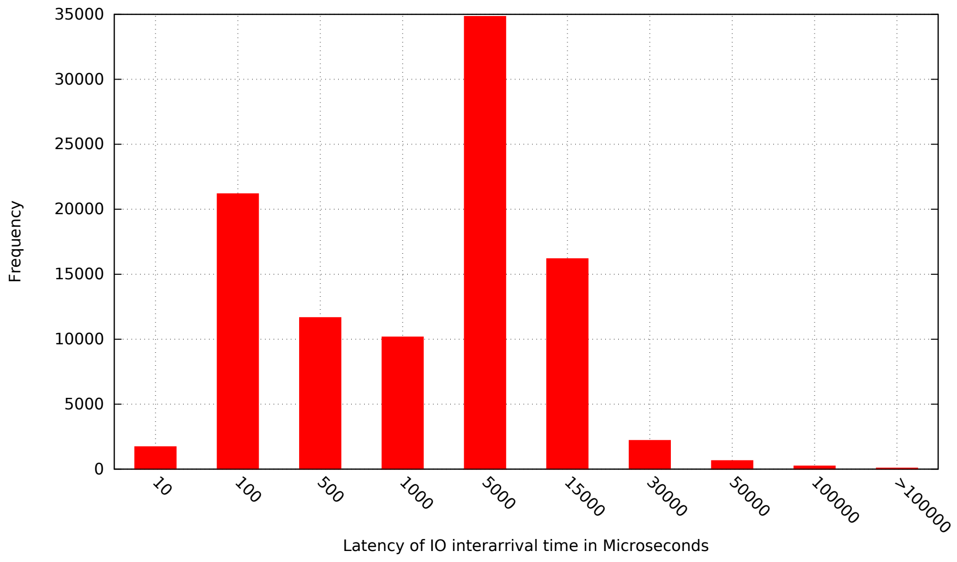


Figure 4.22: Blogbench write interarrival latency.

an interarrival time below 0.1 ms, whereas 23.2% of the write accesses were below 0.1 ms. This could be caused by larger file creation operations exceeding a single block. These have to be split up into multiple block accesses and take longer process.

4.3.2 Postmark

Postmark [53] is a workload designed to simulate the storage behaviour of a mail server, netnews (newsgroups) and web-based commerce. The workload behaviour maps the observed characteristics of ISPs (Internet Service Provider) who deployed NetApp filers to support such applications. Initially it creates a base pool of files. These are used in a subsequent phase in which more files can be created, files can be deleted, read and extended. Finally, at the end of the workload, all files are deleted.

A standard mail server will contain several thousands to millions of small files with varying sizes from one kilobyte to more than a hundred kilobytes. Emails can also come with attached files that increase the file size to many megabytes.

The default configurations uses file sizes between 5 and 512 kilobytes with an initial pool of 500 files and a total of 20000 transactions [135] [136]. The tested configuration used file sizes between 1KB and 16MB to better reflect the growth in file sizes used for Emails and attachments (pictures, multimedia files). The number of files was set to 8000 with a transaction count of 50000. The workload can configure the `write_block_size` and `read_block_size` and these were both set to 512 bytes.

During the traced run, 2658283 I/Os were recorded, 78% were reads and 22% were writes.

4.3.2.1 I/O length

The average I/O length of all recorded I/Os (see Figure 4.23) during the run was 225714 bytes. The smallest access size was 4096 bytes and the largest access size was 4MB. The default configuration of Postmark uses a `write_block_size` of 4096 bytes [53], while the test was performed with 512 bytes. The trace data indicates that this setting is not in line with the workload access pattern. 128KB is the most dominant access size, used in 71% of all transactions. An access size of 512KB was used in 10.8% of the accesses, while 4KB was used in 6.6%.

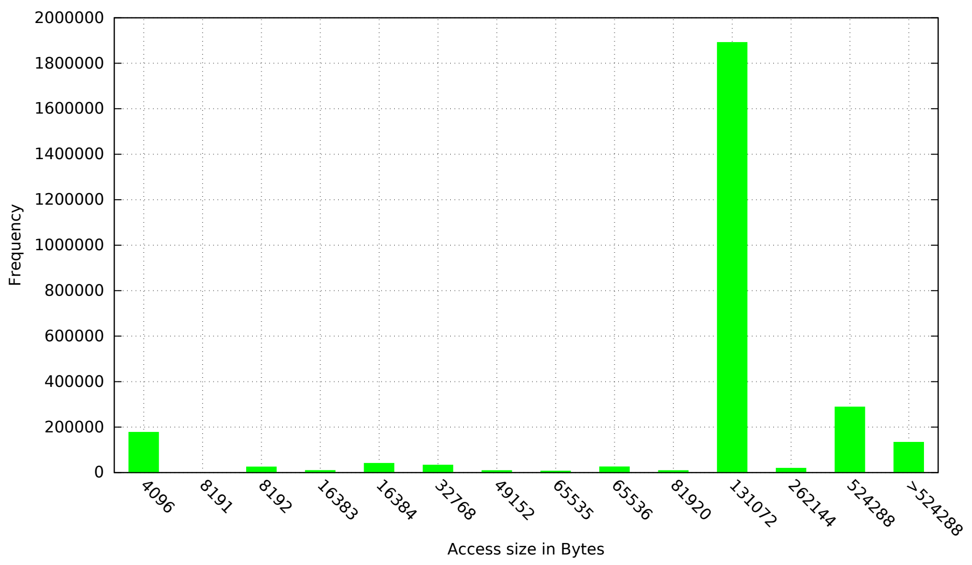


Figure 4.23: Postmark total I/O length.

The I/O length during reads (see Figure 4.24) showed a strong bias to 128KB access sizes. 91.1% of the read accesses during the trace used this access size. Larger accesses only account for about 0.5%. Therefore, the average is reported as 120KB which, as mentioned above, does not match the set `read_block_size` of 512 bytes (default: 4096 bytes [53]).

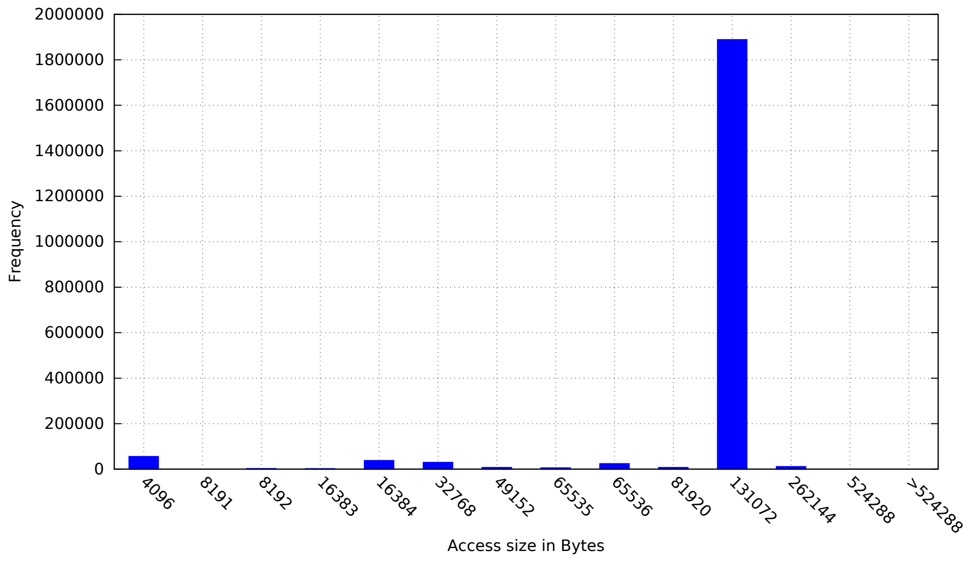


Figure 4.24: Postmark read I/O length.

The I/O length distribution for Postmark writes (see Figure 4.25) differs from the reads. The most common access size of 512KB is present in 49.1% of the write accesses. Larger write accesses (>512KB) are present in 22.6% of the accesses and 4KB accesses are present in 20.6%. The largest access size is 4MB and the average is 575KB.

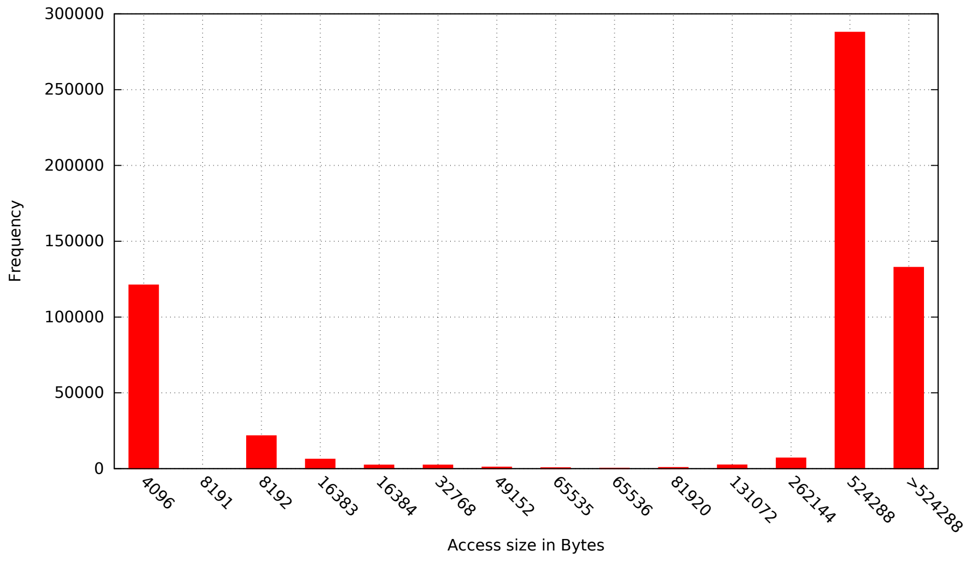


Figure 4.25: Postmark write I/O length.

Overall, the workload shows I/O access size characteristics that differ significantly between reads and writes. Reads consist mostly of 128KB accesses and write accesses consist of block sizes of 512KB and above. Moreover, write accesses of 4KB in size are frequent.

4.3.2.2 Seek Distance

The seek distance for Postmark (see Figure 4.26) shows very sequential behaviour. 86% of the accesses are done on the next block, without requiring any head movements, and less than 8.2% of the accesses have distances of more than 500000 blocks.

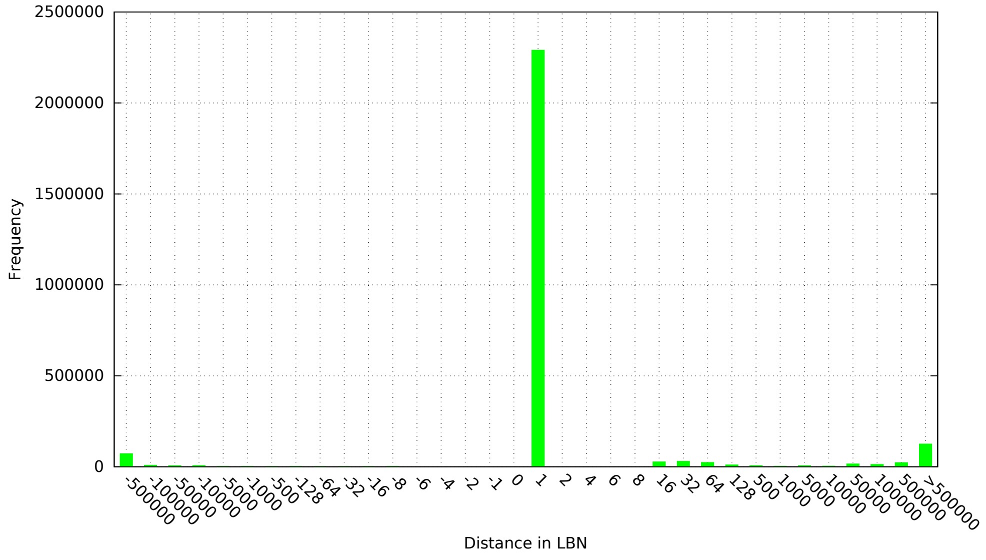


Figure 4.26: Postmark total distance.

The reads for Postmark (see Figure 4.27) are mostly sequential; 94% of the I/Os access the immediately following blocks.

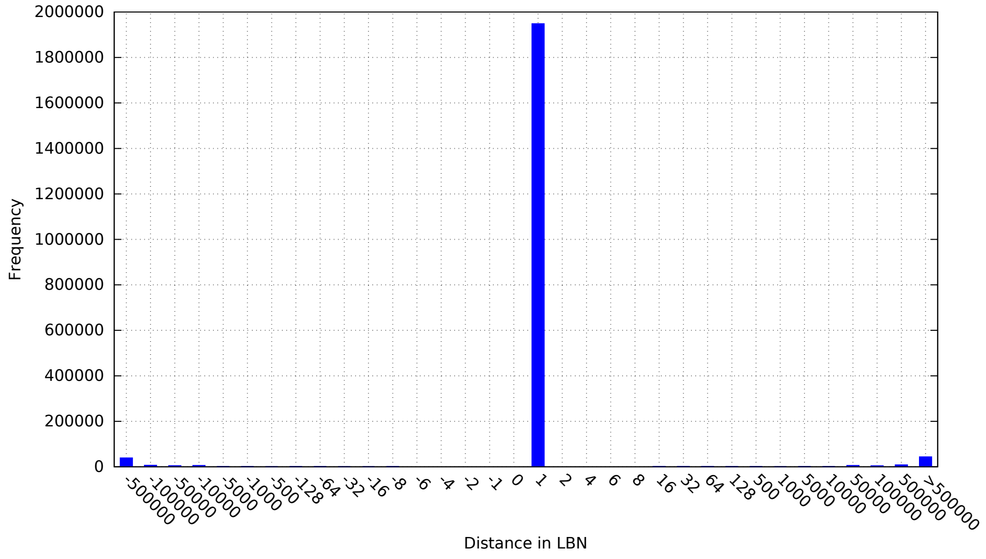


Figure 4.27: Postmark read distance.

For the Postmark writes (see Figure 4.28), the seek distance shows a high number of sequential accesses; 61.2% of the accesses are executed on blocks with a distance of less than 128 blocks. Accesses to more distant positions were recorded for 38.8% of the I/Os, which indicates a mix of sequential and random accesses.

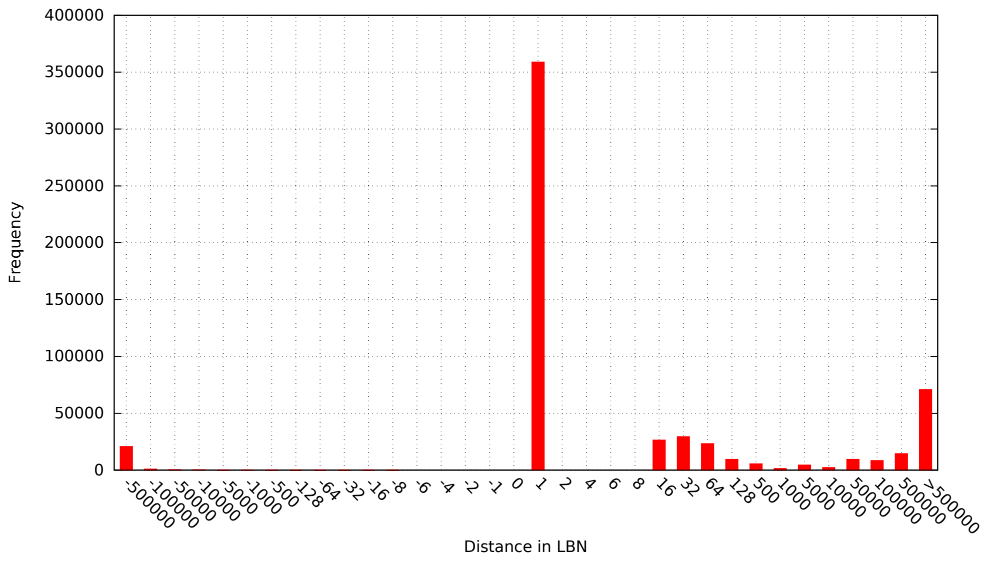


Figure 4.28: Postmark write distance.

Overall, Postmark is mixture of sequential and random accesses. Making tuning decisions without taking this mix into account, can lead to a configuration that performs worse since it does not consider the application characteristics.

4.3.2.3 Interarrival Latency

The interarrival latencies for the Postmark workload (see Figure 4.29) show a steady load with 74.2% of the I/Os arriving in under 1 ms and 87.7% in under 5 ms from the previous access. 12.3% of the I/Os arrive later than 5 ms after the previous access. This means Postmark is a steady load rather than being irregular or bursty.

The interarrival latencies for Postmark reads (see Figure 4.30) are clustered at low latencies; 82.3% of the reads arrive in under 1 ms and 91.9% under 5 ms.

The interarrival latencies for Postmark writes (see Figure 4.31) are less dispersed. In comparison to the reads, 43.3% of the I/Os arrive in under 1 ms. 29.1% arrive between 1 and 5 ms and 0.9% arrive after 100 ms.

The histograms for the interarrival latency of Postmark I/Os show that the workload creates a steady load on the storage system. Block accesses are dispatched very frequently for reads and less so for writes without becoming bursty.

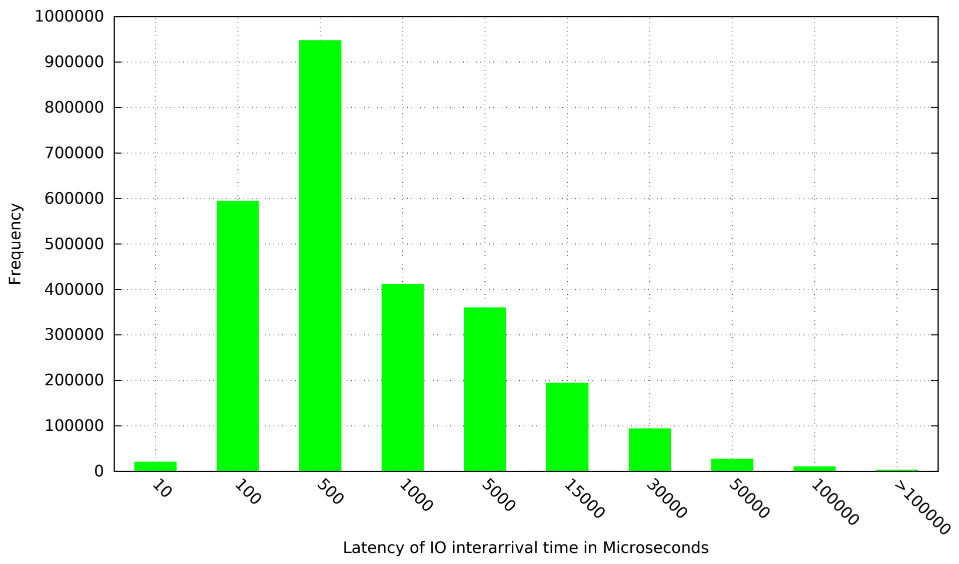


Figure 4.29: Postmark total interarrival latency.

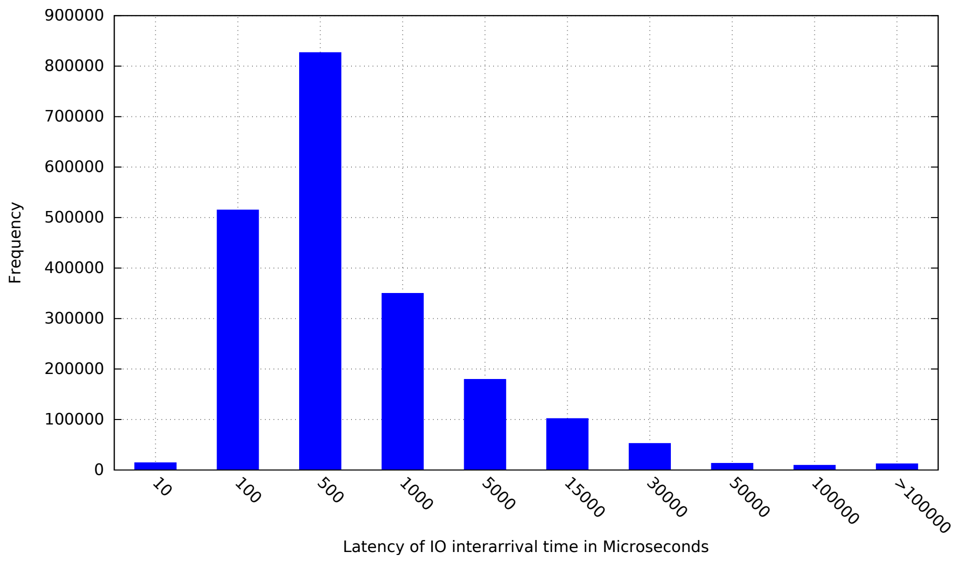


Figure 4.30: Postmark read interarrival latency.

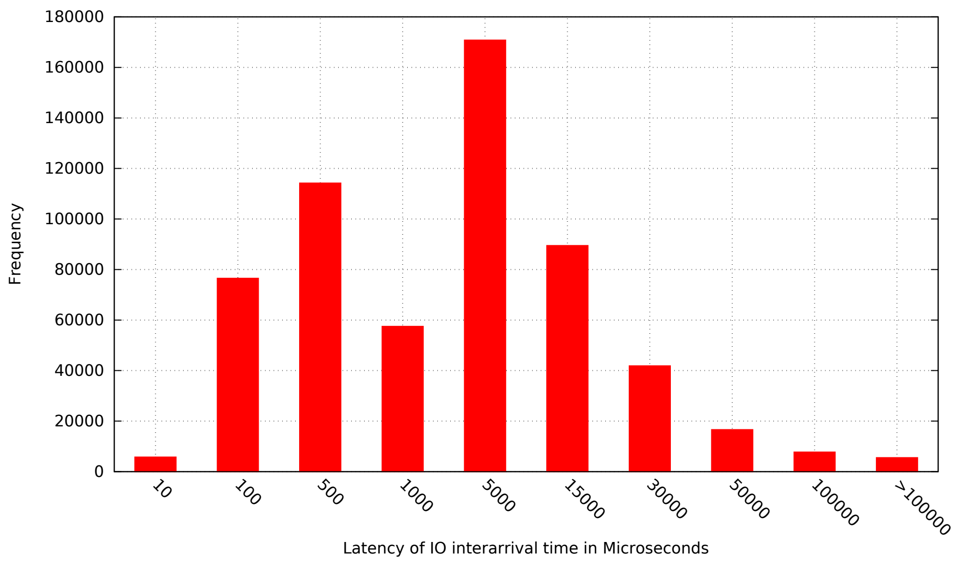


Figure 4.31: Postmark write interarrival latency.

4.3.3 DBENCH

DBENCH [74] is a tool that can target read and write operations at a number of different storage backends, such as iSCSI targets, NFS or Samba servers. Furthermore, it can be used to stress a file system to determine when that system will become saturated and to determine how many concurrent clients or applications can be sustained without service disruptions, including lagging [137] [138].

It is part of a suite of tools that contains DBENCH, NETBENCH and TBENCH. NETBENCH is used to stress a fileserver over the network. To simulate multiple clients the tool has to be executed on multiple machines, which can be problematic when deploying dozens or hundreds of clients. DBENCH simulates the load a fileserver experiences by making the file system calls that are typically seen by a fileserver that is stressed by NETBENCH, without using any network communications. TBENCH is used to test only the networking component without precipitating any file system operations; this can be used to check for network limitations, assuming the fileserver and the I/O are not the limiting components.

The developer of the workload specifically states that the load is not completely realistic as it contains many more writes than reads, which does not reflect a real office workload.

DBENCH simulates multiple concurrent clients with the same client configuration. The behaviour of the accesses changes according to the number of clients: the more concurrent clients, the more random the accesses.

In this trace a configuration with 48 clients was used to reflect a realistic SME size [139] (small- and medium-sized enterprise).

The total number of I/Os in the trace was 730578. Contrary to the DBENCH developer's statement regarding the read/write ratio, only 209 of the I/Os were read accesses, which is the equivalent of less than 0.03%. The expectation was that the read/write ratio would be in the order of 90% writes since the load generation file contained many more than the 209 read statements appearing in the trace. This suggests that the files were resident in memory rather than the disk and so were invisible to the tracing procedure.

Due to the discrepancy between the reads and writes, only the write charts will be analysed, as the read graphs contain insufficient data to make any accurate judgement.

4.3.3.1 I/O length

The accessed I/O lengths during the DBENCH trace (see Figure 4.32) shows a high utilization of 4KB I/O accesses. This access size was used in 54.7% of the write accesses. Cumulatively the access sizes from 8KB to 31KB make up 18.8% of the total. Accesses of 80KB and 128KB were seen in 8.1% and 8.4% of the respective accesses. Larger access sizes were see in under 6% of the total. The average accessed I/O size during the trace was 42KB.

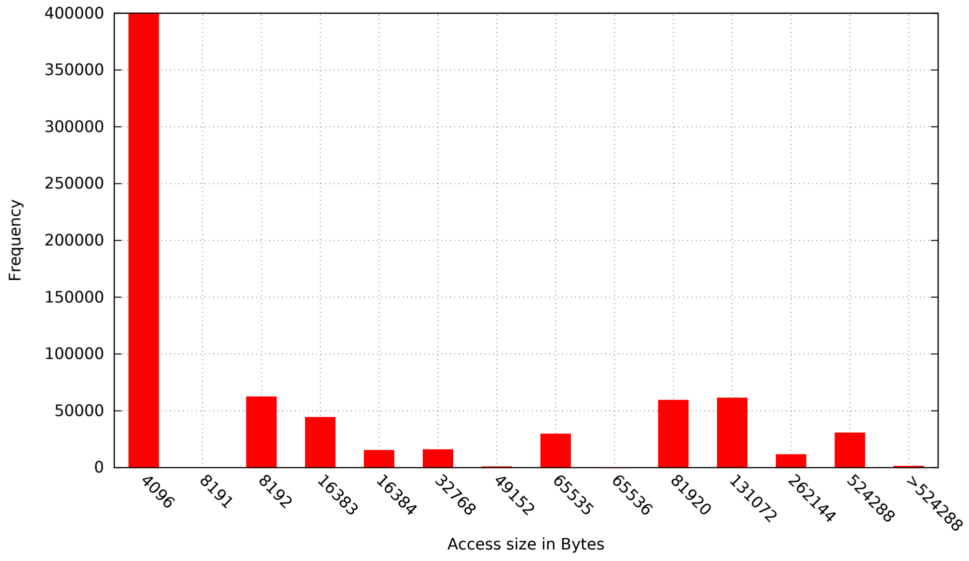


Figure 4.32: DBENCH write I/O length.

4.3.3.2 Seek Distance

The seek distance for writes during the DBENCH trace (see Figure 4.33) is highly random. Less than 5% of the accesses were made to neighbouring blocks. 51.5% of the accesses were to blocks further than 500000 blocks distant from the previous access. Lowering the client count would potentially result in a more sequential behaviour.

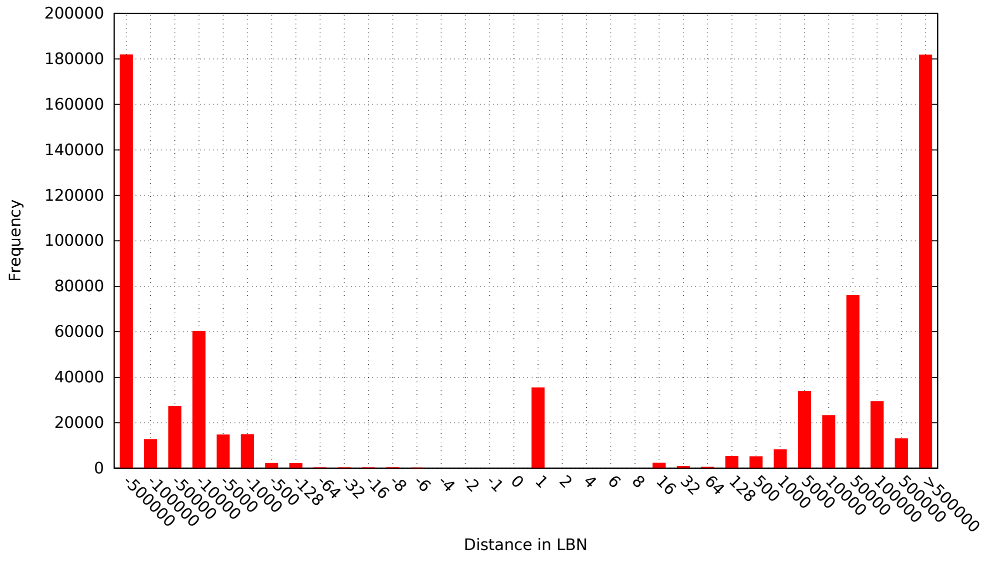


Figure 4.33: DBENCH write distance.

The offline block distribution of the trace is depicted graphically in Figure 4.34) and shows the three main areas where the file system writes.

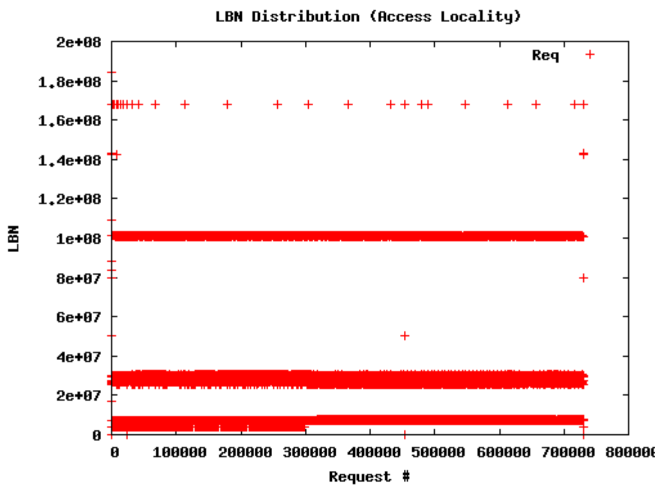


Figure 4.34: DBENCH write distribution.

4.3.3.3 Interarrival Latency

The interarrival latency during the DBENCH benchmark (see Figure 4.35) shows a strong bias to low interarrival latencies; 36.8% of the accesses arrived in under 10 µsec and 32.4% of the accesses in under 100 µsec. 83.4% of all write access arrived in under 500 µsec of each other.

The offline trace (see Figure 4.36) shows the corresponding high IOPS rate. The trace shows an average between 1000 and 1200 IOPS.

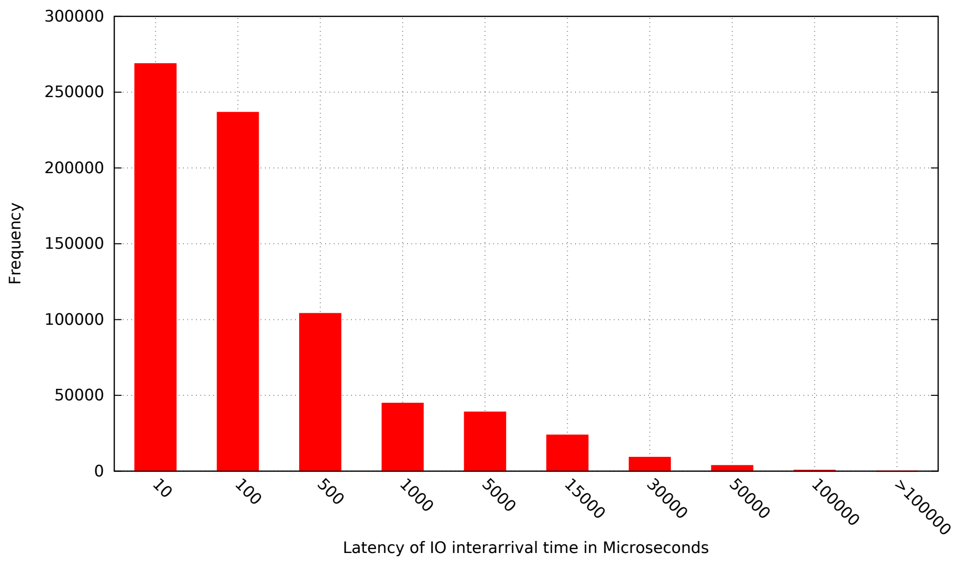


Figure 4.35: DBENCH write distance.

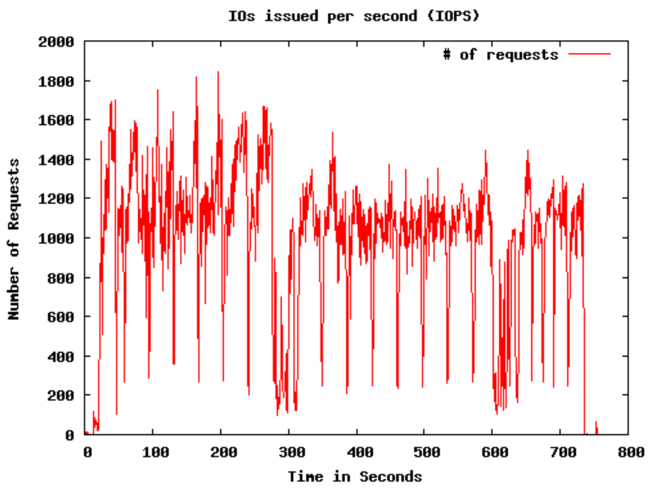


Figure 4.36: DBENCH write IOPS distribution.

4.3.4 Kernel Compile

The task of compiling the Linux Kernel is one of the workloads, used on cloud machines, that can be considered as continuous integration as described in Section 2.3.4. This task is mainly CPU intensive, but it also requires reading source files and writing the compiled code back to disk. The version of the workload chosen here compiles the Linux Kernel 4.3 [140] [141]. This Kernel version in its compressed form is 126 MB in size. The workload extracts the file first before starting the compilation. The extracted archive contains 55008 files in 3438 folders and sub-folders. In total the extracted Kernel source is 613.5 MB.

The workload extracts the Kernel once and compiles it 3 times to determine a representative average. Between each compilation run it deletes the compiled image. It uses multiple concurrent threads to speed up the process, in deference to the number of CPU cores available on the machine. On the machine used for gathering the trace the core count was 4.

4.3.4.1 I/O length

Overall there were 14008 I/Os recorded during the workload, 14.6% being reads and 85.4% being writes. The I/O length used most often overall (see Figure 4.37) is 4KB. This access size was present in 40.1% of all accesses. 8KB and 128KB block sizes were each present in 9.5% of all accesses; 32KB and 48KB blocks appeared in 7.85% of the total. The reported average was 111KB.

Two dominant access sizes were present in the Kernel compile read accesses (see Figure 4.38). 128KB access sizes were present in 52.6% of the read accesses. 4KB I/O sizes were used in 36.2% of the read accesses. The largest recorded access size was 256KB.

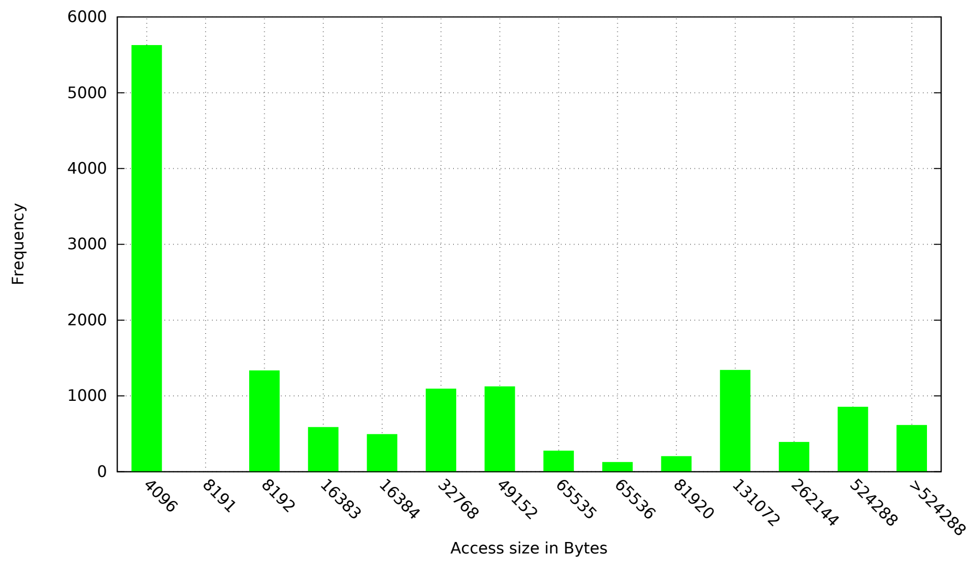


Figure 4.37: Kernel compile total I/O length.

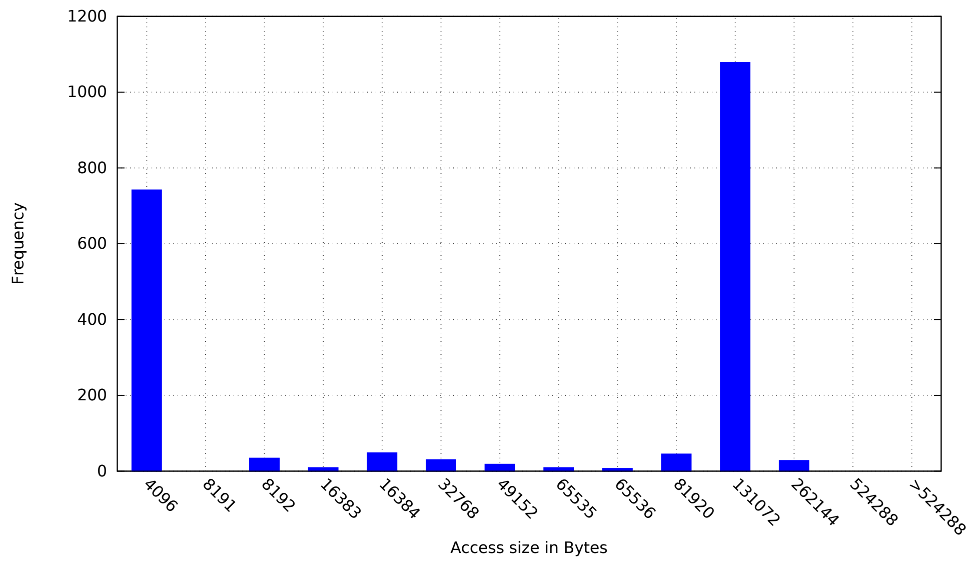


Figure 4.38: Kernel compile read I/O length.

The 4KB access size comprised 40.8% of the total write accesses (see Figure 4.39); 8KB, 32KB and 48KB accesses comprised between 8.7% and 10.8% of the total. Large accesses with 512KB were present in 7.1% of the total. The largest recorded access size was 4MB and the average was 120KB.

Even though the Linux Kernel is composed of more than 55000 files, the default configuration requires less. Therefore, not all files will be touched. Overall the compilation makes use of mostly 4KB blocks for write accesses and 4KB and 128KB for read accesses.

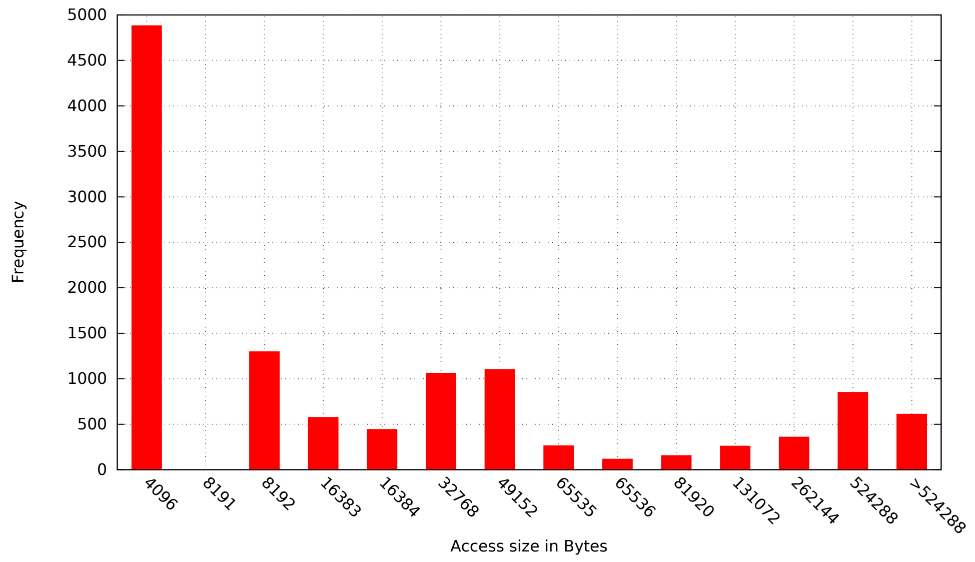


Figure 4.39: Kernel compile write I/O length.

4.3.4.2 Seek Distance

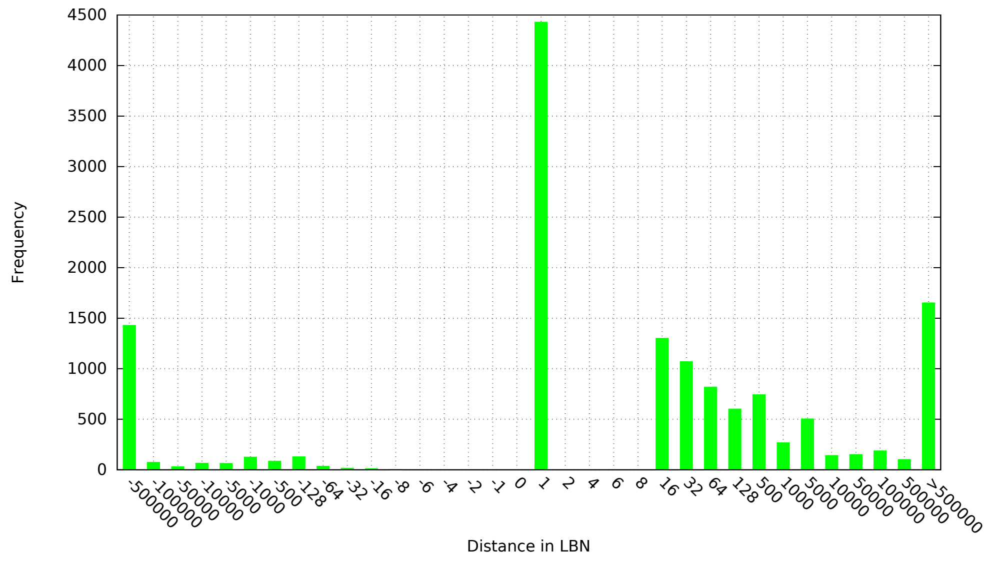


Figure 4.40: Kernel compile total distance.

The seek distance observed during the Kernel compilation (see Figure 4.40) shows a peak in the centre, indicating a sequential access pattern, but this comprises only 31.6% of all accesses. Accesses that differ in block addresses by more than 500000 in each direction account for 22.7%. Therefore, the workload is more random than it is sequential.

Reads during compilation (see Figure 4.41) exhibit a sequential access pattern, 75% of these accesses read the next block. More distant accesses are rare.

When writing, the Kernel compilation exhibits sequential access patterns in 24.6% of

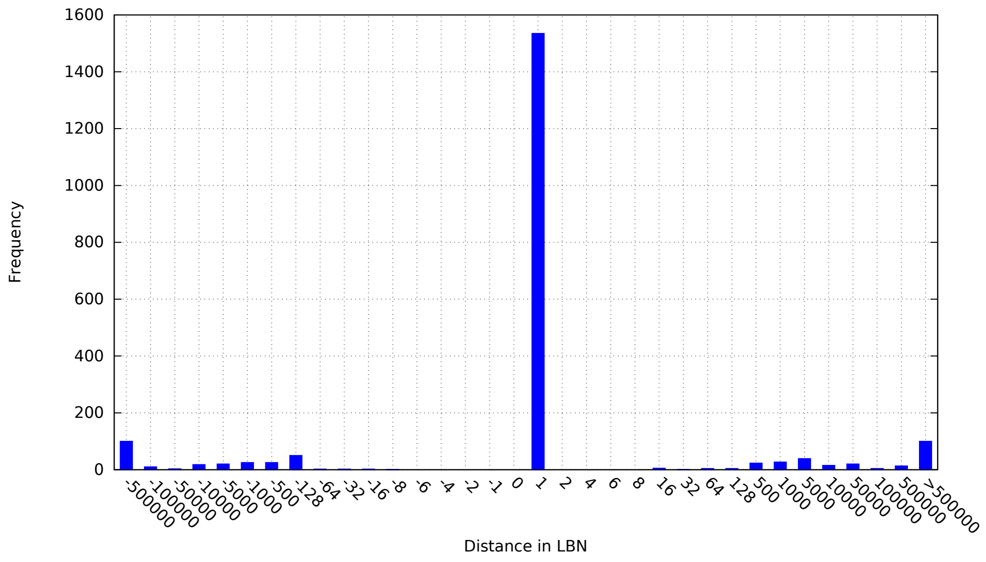


Figure 4.41: Kernel compile read distance.

the total (see Figure 4.42). Long distance jumps make up for 24.1% of that total.

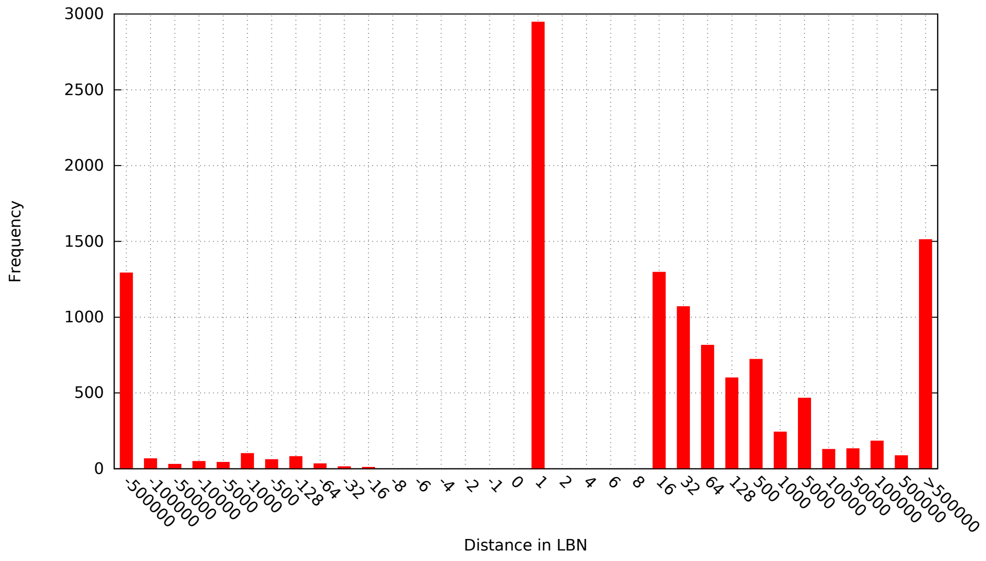
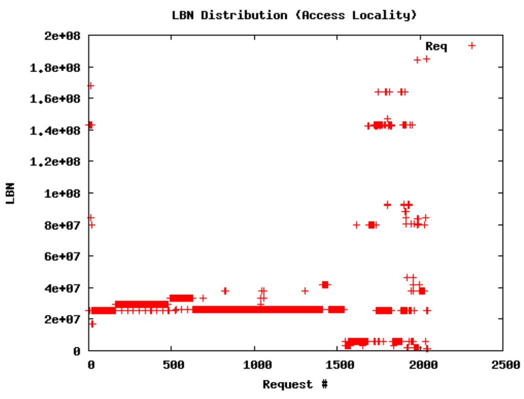
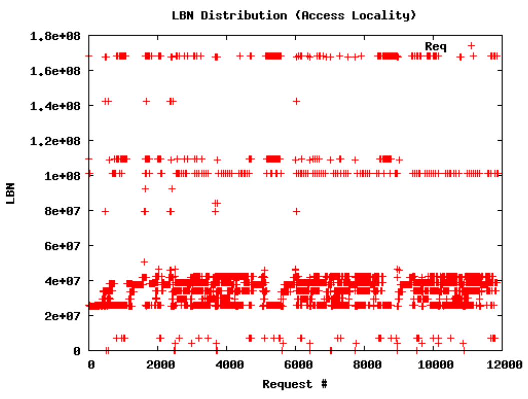


Figure 4.42: Kernel compile write distance.

The offline trace casts more light on the situation. It shows that reads (see Figure 4.43a) are sequential at the beginning of the process and random after 1500 accesses. The write diagram (see Figure 4.43b) does not give any further insight into the randomness of the write accesses.



(a) Read



(b) Write

Figure 4.43: Detailed Kernel compile seek distance.

4.3.4.3 Interarrival Latency

The interarrival latencies for the Kernel compilation (see Figure 4.44) show a leaning toward lower latencies; 64.3% of the accesses arrived within 1 ms of the previous, 83.5% in under 5 ms. The peak latency in the recorded trace was between 10 and 100 µsec accounting for 32.4% of the total accesses.

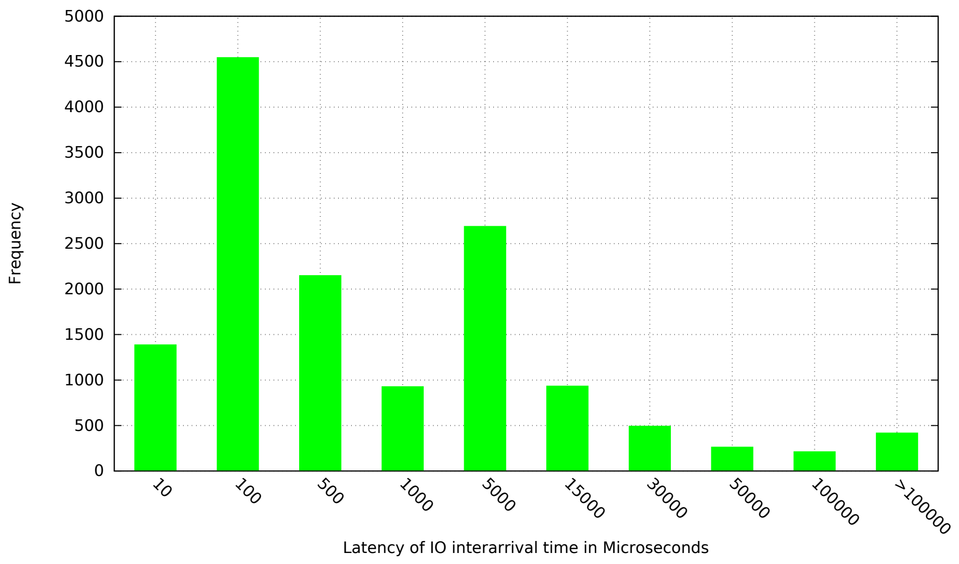


Figure 4.44: Kernel compile total interarrival latency.

The Kernel compilation read interarrival latencies (see Figure 4.45) reveal that 39.1% of the accesses arrived with a latency under 1 ms to the previous access; 85.7% of the accesses exhibit an interarrival latency of up to 5 ms. The large number of accesses having an interarrival latency of less than 10 µsec is remarkable since it shows that the workload has a high burst ratio.

During Kernel compilation writes, the interarrival latencies (see Figure 4.46) show a

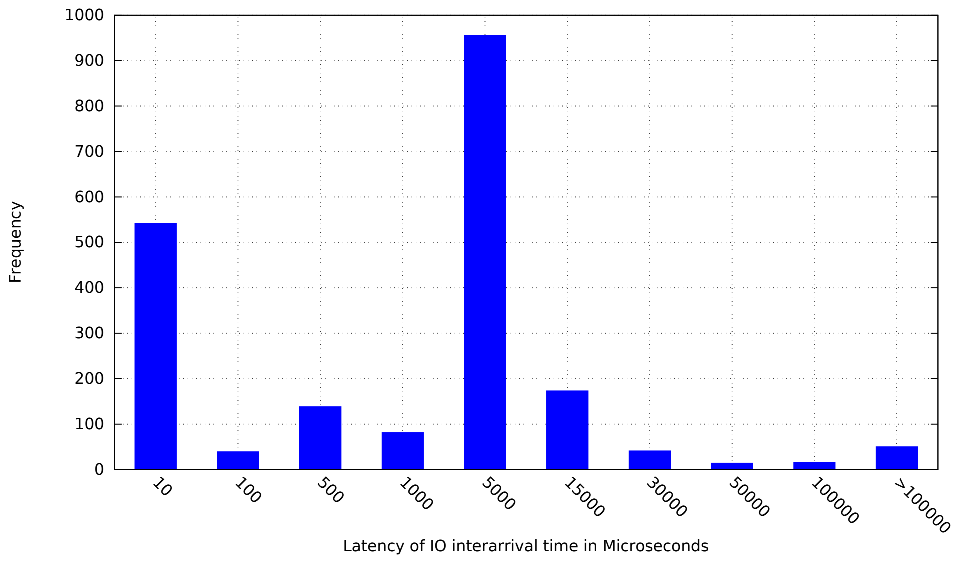


Figure 4.45: Kernel compile read interarrival latency.

peak at 100 µsec. 44.7% of the all accesses exhibit this latency whereas 68.3% have an interarrival latency of under 1 ms and 82.6% under 5 ms.

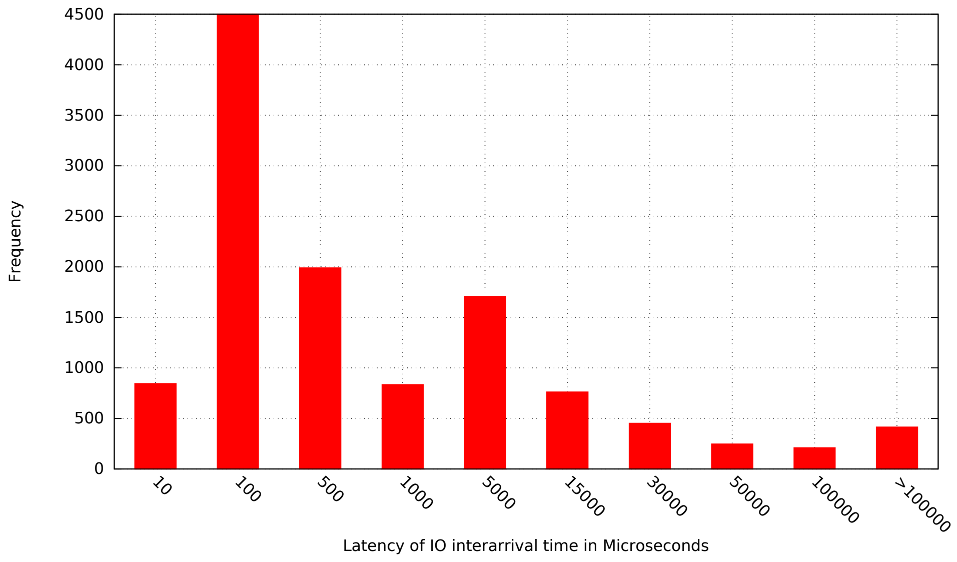
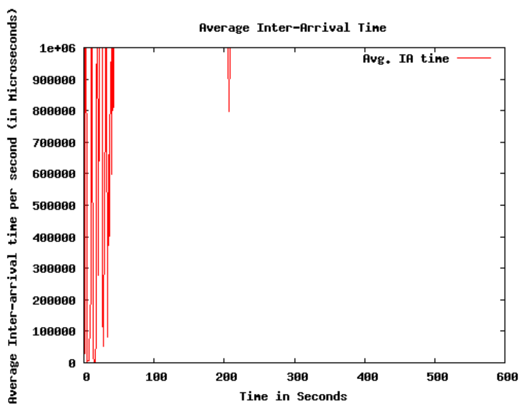
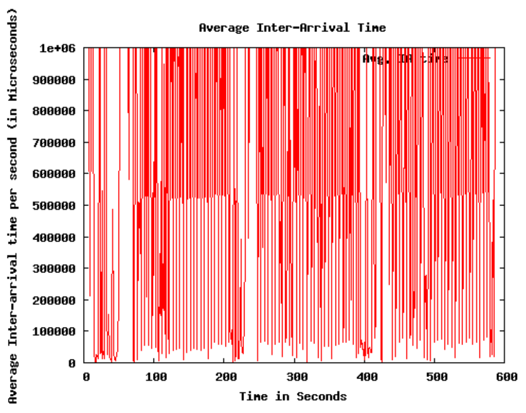


Figure 4.46: Kernel compile write interarrival latency.

The workload is easier to understand using the offline analysis. Reads (see Figure 4.47a) occur at the beginning of the trace when the Kernel archive is extracted and written to disk. The data is then either cached directly to memory or is read once and then cached to memory. Write accesses (see Figure 4.47b) happen at the beginning when the archive is extracted and written to disk and when the compiled code is saved. The three runs of the compilation are visible between 70-220, 240-420 and 434-600 seconds within the trace.



(a) Read



(b) Write

Figure 4.47: Detailed Kernel compile interarrival time.

4.3.5 pgbench

The pgbench [64] workload is itself a benchmark program used to test a PostgreSQL database server [65]. It runs the same sequence of SQL commands over and over, possibly with multiple concurrent database sessions, and then calculates the average transaction rate (transactions per second). The test it runs as default is loosely related to the TPC-B [21] benchmark which consist of five SELECT, UPDATE, and INSERT commands per transaction when used in read-write mode [64], as shown in Listing E.14.

When used in read-only mode the SQL statement is much shorter as presented in Listing E.15.

The workload supports different system components to be tested. This is achieved by setting a scaling factor to determine the size of the database. With a scaling factor of 1 the database contains 100000 entries and is 15MB in size. Depending on the component to be tested, this scaling factor may have to be changed. When used with a scale factor between 1-10 (15-150MB databases) only a small fraction of RAM is used. This can expose locking contention and problems with CPU caches and similar issues not visible when used with larger scales [142].

During the trace a scaling factor of 1228 was used, which is the equivalent of 0.3× size of RAM (in MB). This resulted in a database file of 18 GB and 122880000 entries [143] [144].

The workload supports two different types of database testing: fixed duration and number of transactions. For the fixed duration is easily replicated and the duration of the run is relatively predictable. The transaction based run exhibits an unpredictable runtime. Therefore the time based mode was used with a runtime of 3600 seconds.

4.3.5.1 I/O length

In total there were 2082155 disk accesses recorded. The I/O length distribution (see Figure 4.48) shows a peak at 8KB; 74.3% of the accesses were made with that access size. Accesses with an I/O size of 128KB were the second most common with 12.8% of the total. Other access sizes were not prevalent.

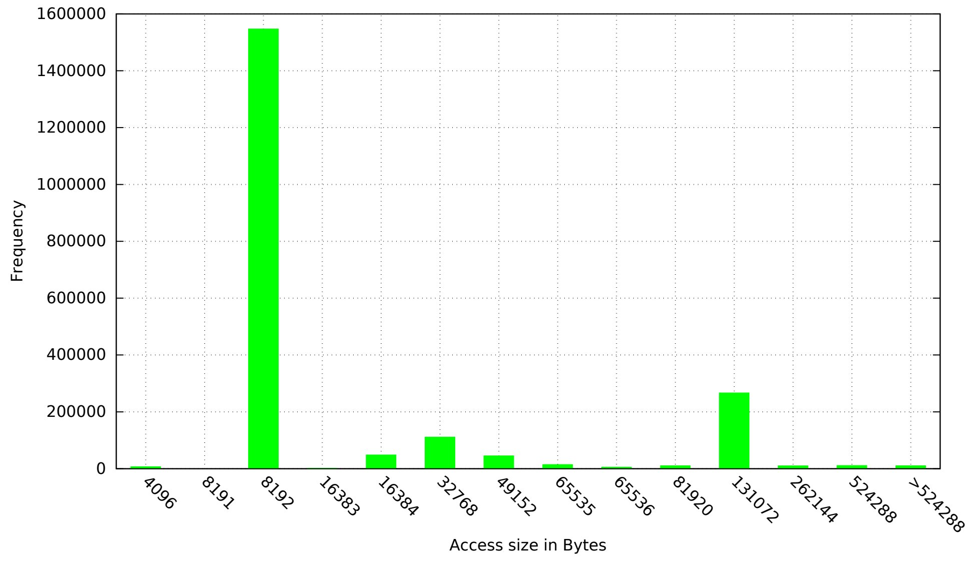


Figure 4.48: pgbench total I/O length.

A total of 977077 read accesses was recorded in the pgbench trace (see Figure 4.49). 8 KB block sizes were present in 68.8% of the read accesses and 128KB were present in 26%.

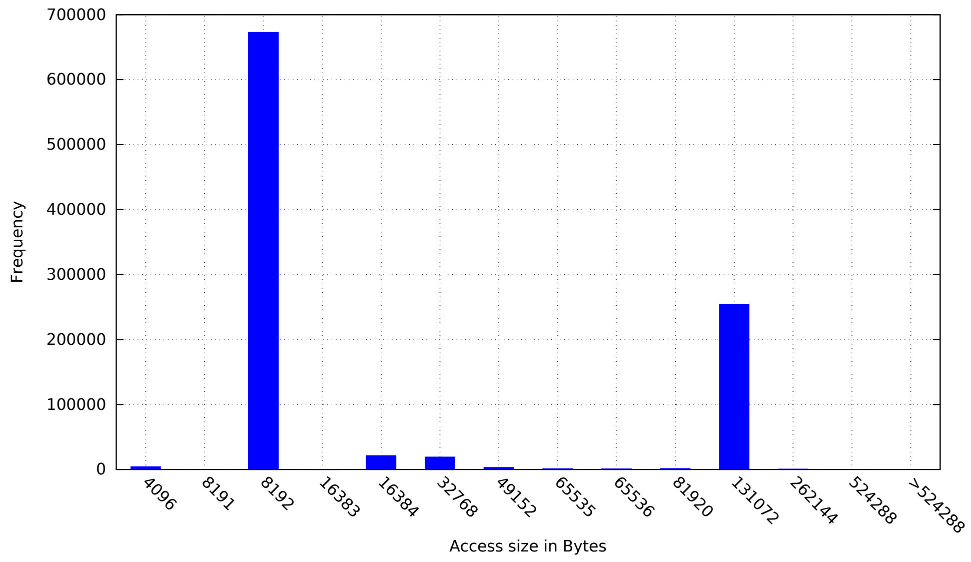


Figure 4.49: pgbench read I/O length.

The 8KB access size was present in 79% of the pgbench writes (see Figure 4.50). An access size of 32KB was present in 8.3% of the total write accesses.

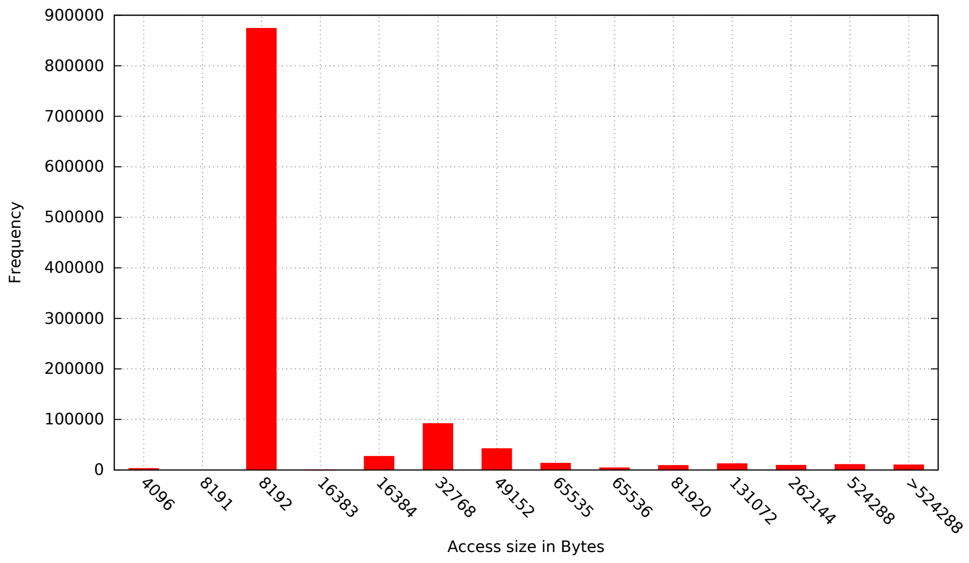


Figure 4.50: pgbench write I/O length.

Overall, the numbers show that for the pgbench workload, with normal load, comprising read and write accesses, the dominant access size was 8KB. The differences between reads and writes are not substantial apart from the frequency of larger block size accesses. An access size of 128KB was more frequently used for reads than for writes, which may have an impact on selecting the storage configuration that improves workload most.

4.3.5.2 Seek Distance

The seek distance during the recorded pgbench run (see Figure 4.51) shows three peaks. Far distance seeks of more than 500000 blocks in either direction are observed in 40.6% of the write accesses, and random writes with lower seek distances were observed in 47% of the write accesses. Accesses to the next block were made in 13.2% of all accesses.

The offline trace reveals that accesses were spread over a band of about 20 GB (see Figure 4.52) augmented by accesses at the end of the disk and around the 50 GB mark. Occasionally, accesses in between these positions and to lower disk blocks occurred.

Reads in pgbench (see Figure 4.53) are sequential for 26.4% of the total read accesses. 65.6% occurred with a distance of more than 500000 blocks to the previous write access, making them random.

The pgbench write accesses (see Figure 4.54) exhibit a mixed access pattern of random and sequential accesses. The seek distances recorded were mostly below 50000 blocks in both directions. 65.2% of the write accesses were random.

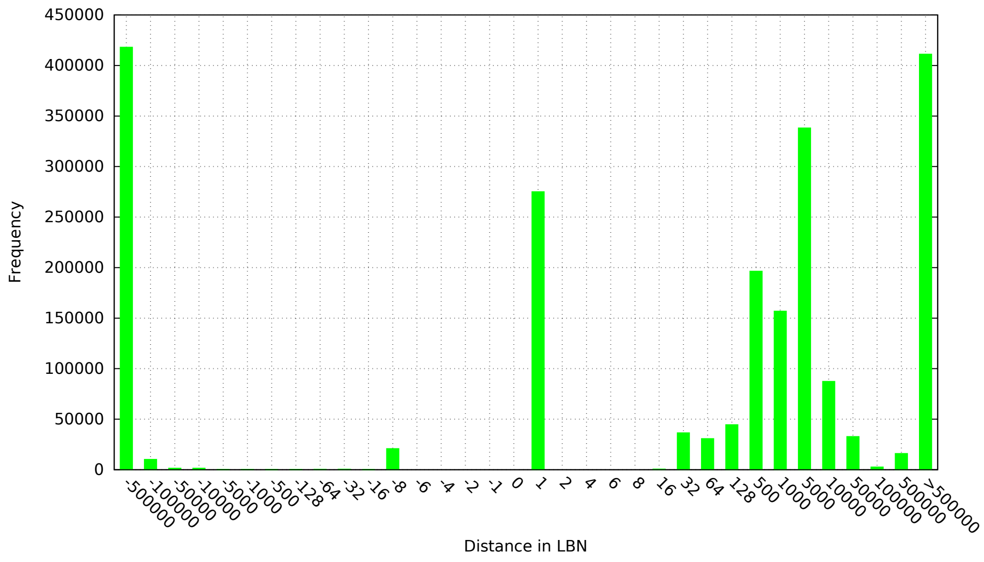


Figure 4.51: pgbench total distance.

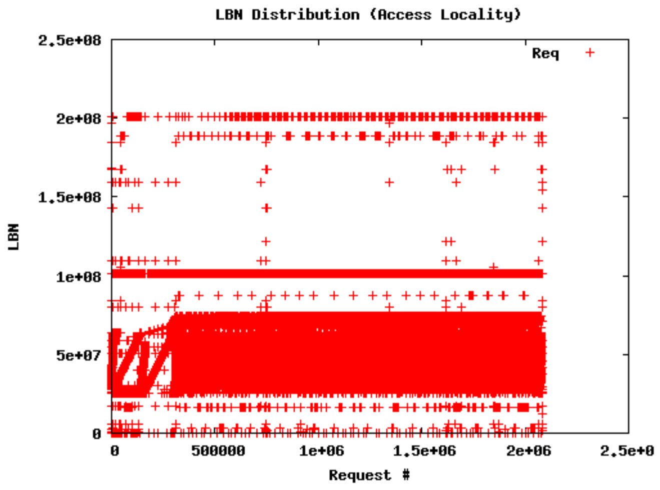


Figure 4.52: pgbench LBN distance offline.

Overall, pgbench displays mostly random accesses. The ratio between random and sequential accesses is almost identical for both access types, while the seek distance distributions differ. Read accesses access more distant blocks than write accesses. This may result in higher seek latencies, since the disk head may have to travel further.

4.3.5.3 Interarrival Latency

The interarrival latencies for the pgbench workload (see Figure 4.55) show most of the I/Os arriving within 15 ms of the previous access and 89% in under 5 ms. 42.4% of the pgbench accesses arrived with a latency of 500 µs or less to the previous access.

The offline trace of the pgbench interarrival latencies (see Figure 4.56) reveals that high interarrival latencies occur at the beginning of the trace when the system is initializing

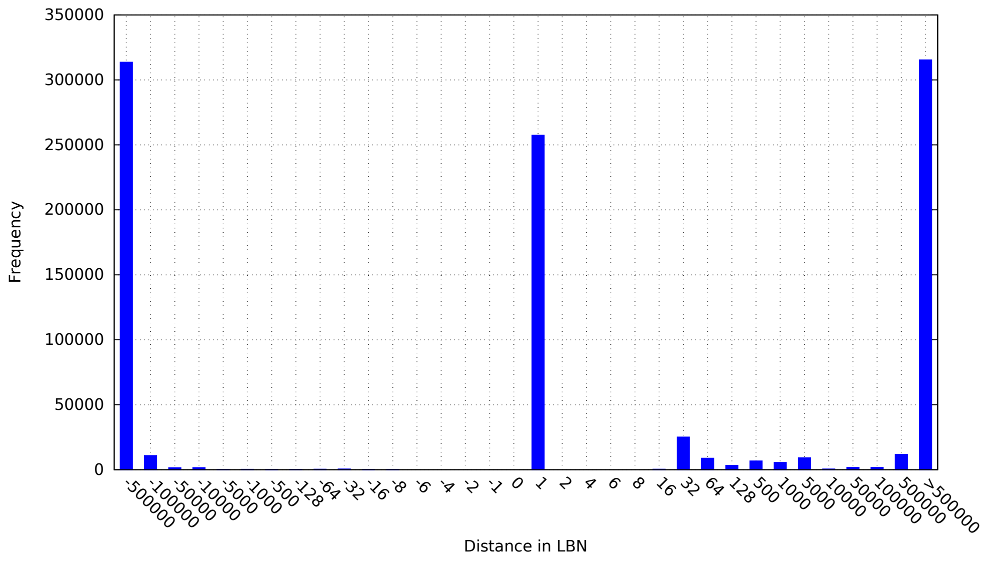


Figure 4.53: pgbench read distance.

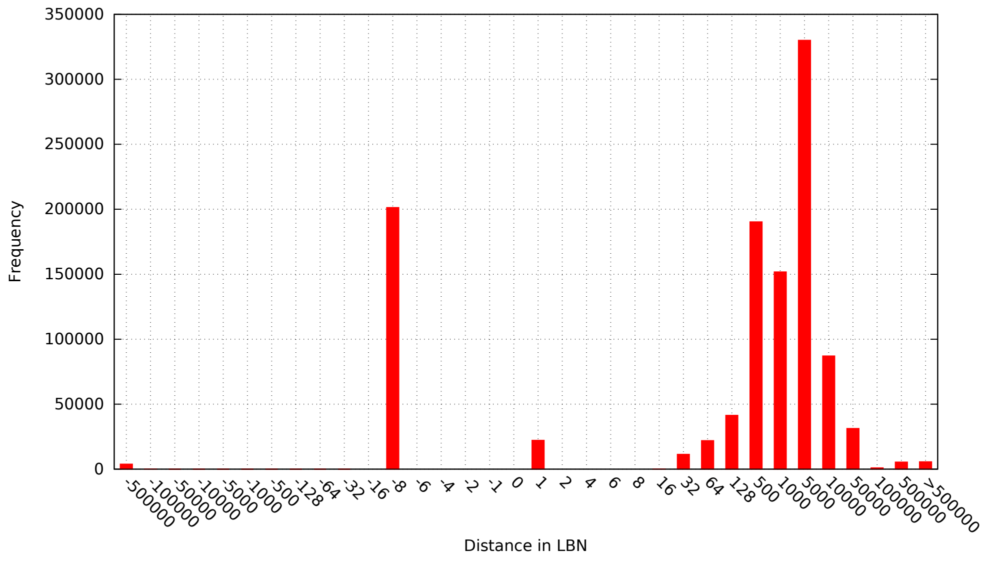


Figure 4.54: pgbench write distance.

the database. When the initialization is finished, the interarrival latencies are low, with occasional latency spikes.

The interarrival latencies of the pgbench read accesses (see Figure 4.57) were under 5 ms to the previous access in 93.8% of the total. Accesses with a latency of less than 10 µs were recorded for 11.5% of the accesses, indicating bursts of I/Os.

The interarrival latencies for pgbench write accesses (see Figure 4.58) show a peak for accesses between 1 ms and 5 ms; 45.7% of the writes arrived with this latency to the previous access. Low latencies with less than 100 µs were recorded in 3.1% of the I/Os.

Overall, the pgbench workload exhibits a fluctuating load to the storage system. Read

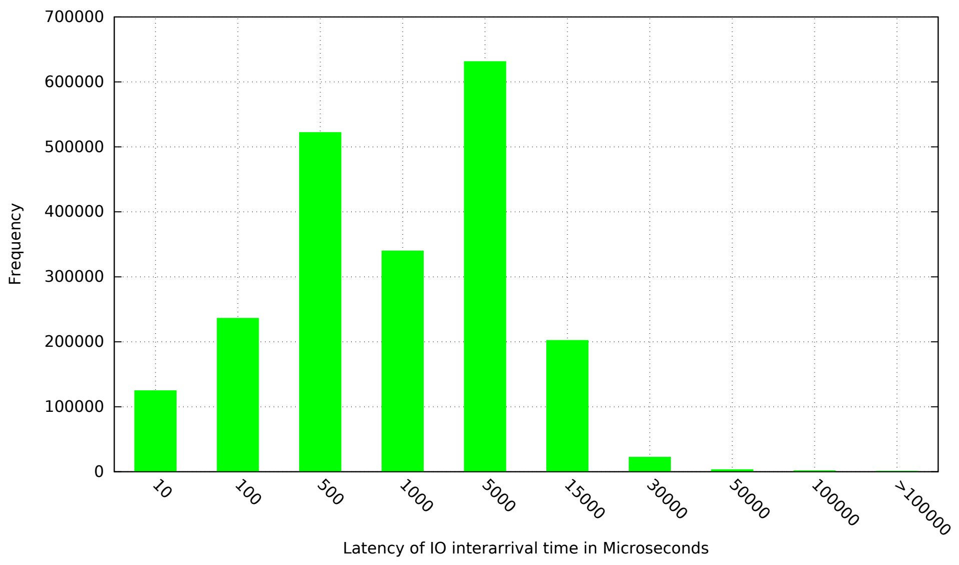


Figure 4.55: pgbench total interarrival latency.

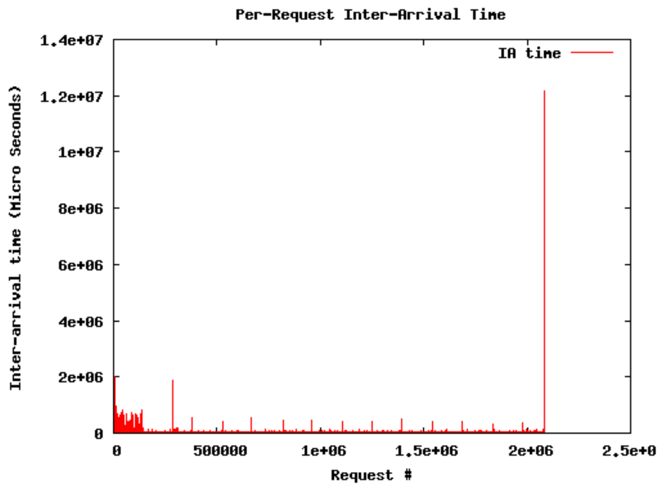


Figure 4.56: pgbench total interarrival latency offline.

accesses display a bursty behaviour. Write accesses are less bursty, but the latencies indicate a fluctuating load.

4.3.6 Summary

It can be seen that the workloads described in the foregoing sections exhibit a wide range of storage access patterns with access sizes from 4KB to 4MB. In addition, collectively they exhibit both random and sequential accesses when reading and writing. As such, this collection of workloads broadly represents typical cloud workloads and thus are relevant for validating the mapping procedure. The following chapter uses the trace information and characterizations determined in the foregoing sections and describes the empirical results associated with this validation.

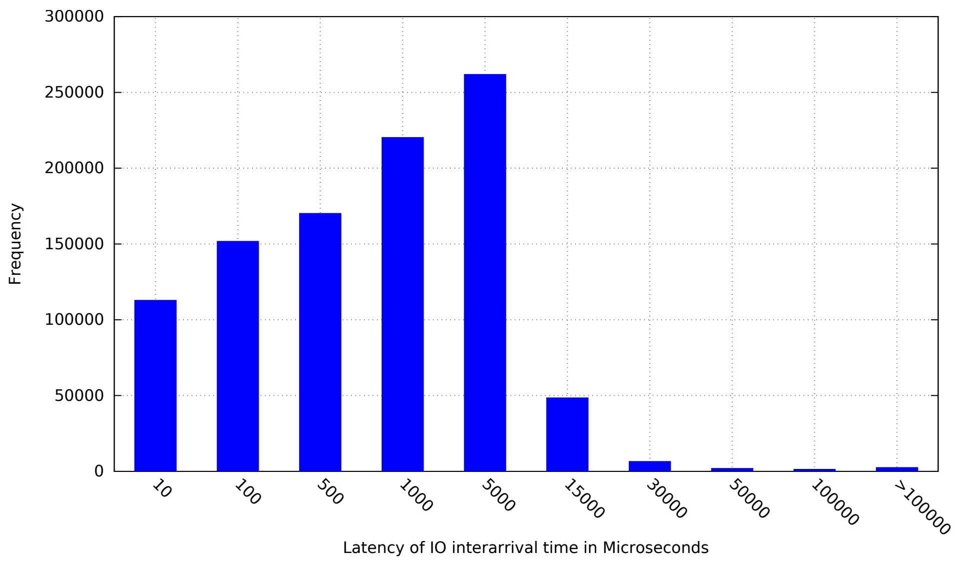


Figure 4.57: pgbench read interarrival latency.

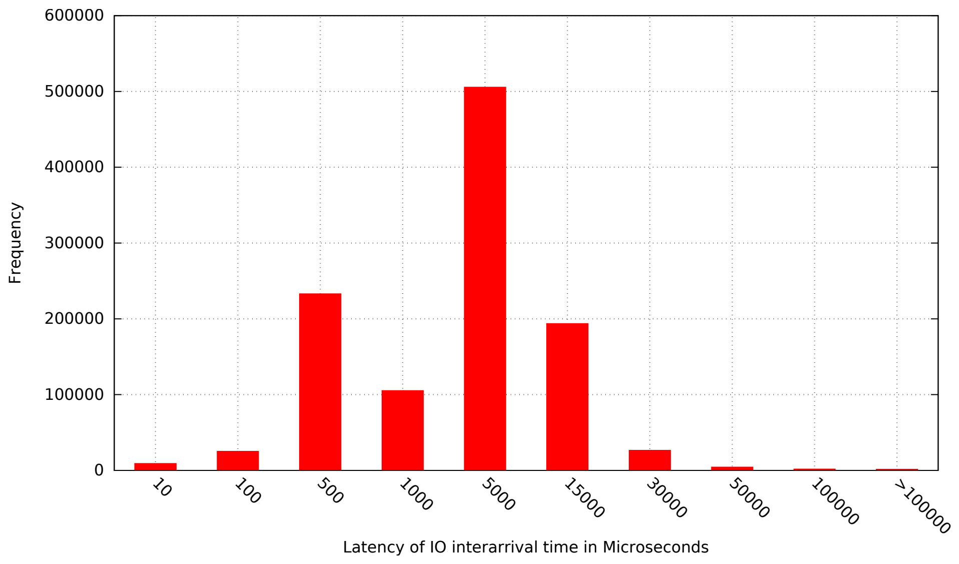


Figure 4.58: pgbench write interarrival latency.

Chapter 5

Verification of the Mapping Procedure

In Chapter 3 different Ceph configurations were analysed for their relative performance in comparison to the default configuration for different access patterns. In Chapter 4 different workloads were traced and analysed for their respective access sizes and randomness. In this chapter the extracted information is used to map the workload to performance enhancing Ceph configurations, as described in Section 2.2.4. The workload is subsequently run on the default and the best and worst performing configurations to investigate the effectiveness of the mapping procedure.

5.1 blogbench

To find a storage configuration tested in Section 3.4 that fits the blogbench workload, it is necessary to analyse the storage trace made in Section 4.3.1 and to map this onto an appropriate Ceph storage configuration.

5.1.1 Workload Analysis

The application trace revealed that during the workload run blogbench in its combined read and write mode executed 122338 I/O accesses. 98902 of them were write accesses (80.8%) while 23436 were read accesses (19.2%). As such a configuration that performs well for write accesses should generally perform well for the blogbench read and write workload.

The dominant access sizes recorded during the blogbench read and write run were mostly below or equal to 32KB, as shown in Figure 4.8. 85.7% of the access fell into this range with 4KB, 8KB and 32KB being the most common in descending order.

Table 5.1: Accesses of blogbench workload for the separate access sizes and randomness.

Total	4KB read	32KB read	128KB read	1MB read	32MB read	% random read
122338	13019	9064	1353	0	0	89.9
	4KB write	32KB write	128KB write	1MB write	32MB write	% random write
	61748	25373	7325	4456	0	71.7

As shown in Figure 4.11, the workload uses a randomized access pattern with read accesses (see Figure 4.12) showing more spread out accesses than the write accesses (see Figure 4.13). Sequential accesses were also in evidence, but they comprised only 3.4% of the accesses. Therefore, for increasing performance for a blogbench read and write workload, a configuration tuned for random read and write accesses should be chosen.

When the access sizes and distances are analysed and put into bins, as described in Section 2.2.4, the accesses are combined to create the five access sizes with their read and writes, as shown in Table 5.1. The information can then be used with the mapping algorithm to calculate the best performing configuration.

The workload shows a constant load, as depicted in Figures 4.19 to 4.22. There is no sign of bursty behaviour. Therefore, a configuration tuned for sustained random write throughput of 4KB and 32KB should provide the highest performance.

5.1.2 Mapping

Choosing an appropriate configuration requires more than considering the throughput diagrams of the 4KB random writes (see Figure 3.6). All information from the trace must be incorporated.

When the mapping algorithm considers only accesses of a specific size it creates two graphs as shown in Figure 5.2. The figure shows the performance of the different configurations when implying a purely sequential or a purely random access pattern for the specific workload access sizes. If the workload were to use random accesses only, Configurations X, T, P and K would result in increased performance in decreasing order, respectively. All other configurations would decrease performance relative to the default by up to 19.2% in the case of Configuration B. Configuration X would surpass the default configuration by 3.6%.

For a purely sequential access scenario only a single configuration (Q) is able to increase performance over the default configuration by a modest 0.3%. The other configurations all decrease performance with Configuration N performing worst with a decrease of 20%.

When the appropriate weights for the percentage of random reads and writes are ap-

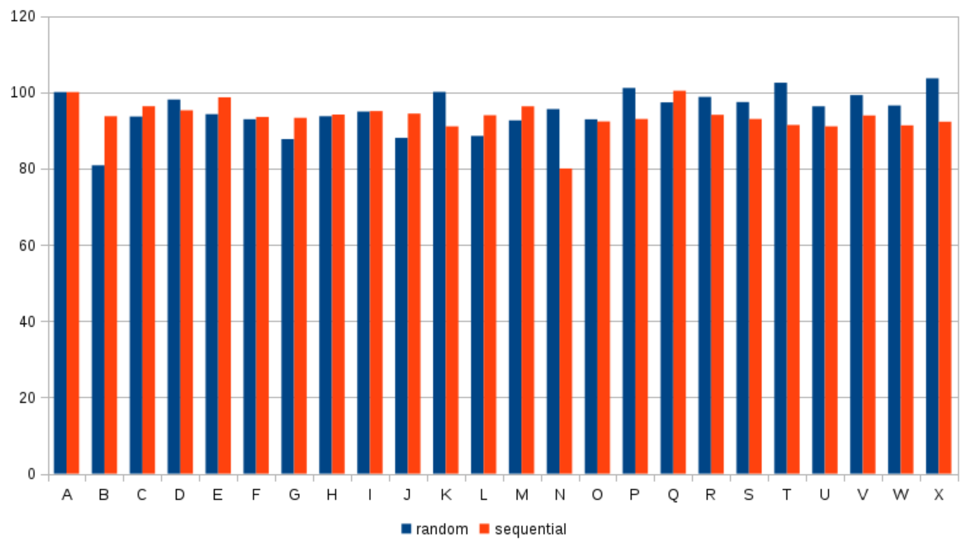


Figure 5.1: Configuration performance for the blogbench workload for sequential and random accesses.

plied the results change as shown in Figure 5.2. Only Configuration X now shows a performance increase of 0.9%, much less than the previous 3.6%. The performance of the other configurations is between 0.1% (T) and 16.3% (B) lower than the default configuration.

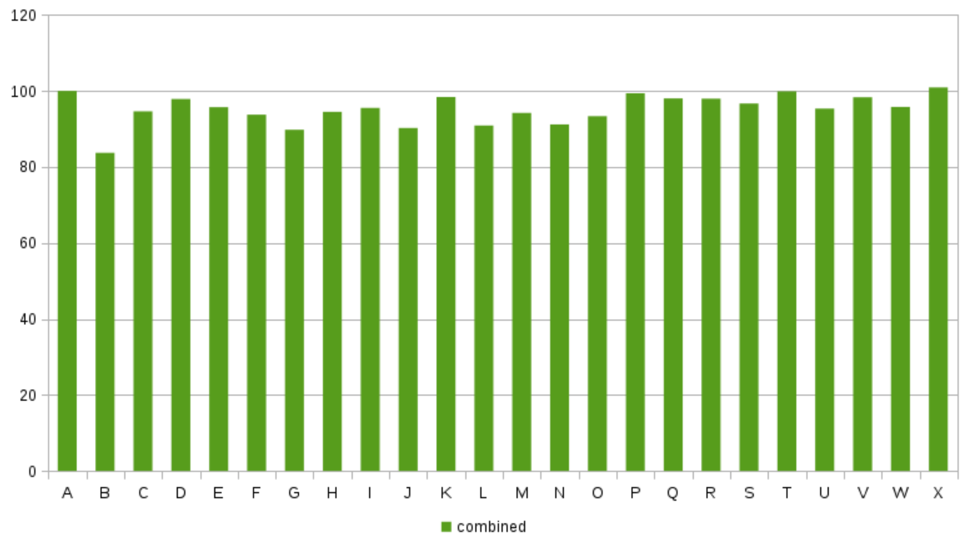


Figure 5.2: Configuration performance for the blogbench workload of combined sequential and random accesses with weights applied.

5.1.3 Results

The results of the mapping process were verified by replicating the workload concurrently on 12 virtual machines. The configurations tested were the default (Configuration

A), Configuration B (worst) and Configuration T (best). As mentioned in Section 5.1.1, the workload is CPU bound. Figure 5.3 shows the performance differences between the configurations, identified in the mapping process, when tested against the blogbench workload.

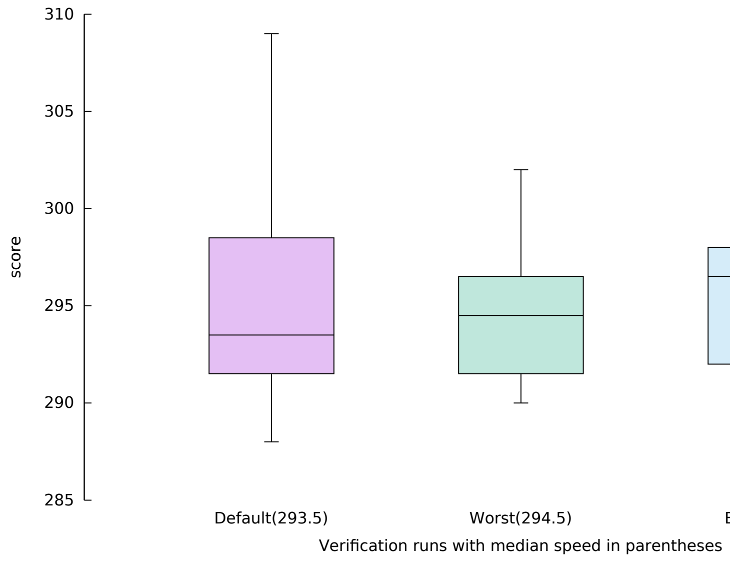


Figure 5.3: Verification of the proposed blogbench configurations.

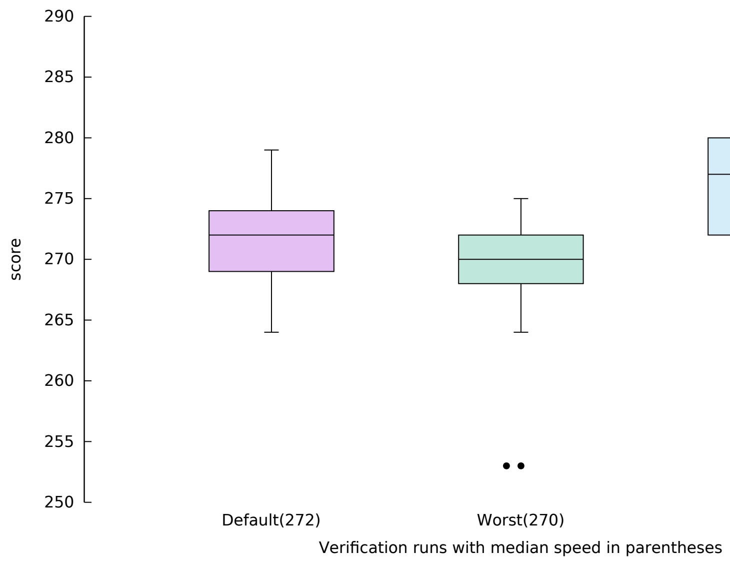


Figure 5.4: Verification of the proposed blogbench configurations using 18 VMs.

The mapping process predicted an increase of performance of 0.9% for Configuration X and a decrease in performance of 16.2% for Configuration B. The empirical results show an increase in performance of 1.0% for Configuration X and indeed and increase of performance also for Configuration B of 3.4%. The differences between the three different configurations are minute. Given the fact that the workload is CPU bound, it is highly likely that even with 12 VMs that it was not possible to generate enough load to keep the storage system busy in any of the three different configurations.

Following the methodology described in Section 2.2.4, the experiment was rerun but this time replicating the workload on 18 VMs (6 per host with no over-provisioning on the host) and the results are depicted in Figure 5.4. Configuration X has the highest performance increases of 1.8%. Configuration B has, yet again, the worst alternative configuration this time reducing performance by 0.7% compared to the default. Increasing the workload replication count from 12 to 18 (an increase of 50%) resulted in only a 39% increase in storage accesses emphasising the CPU bound nature of the workload.

5.2 Postmark

To find a storage configuration to fit the Postmark workload, it is necessary to analyse the storage trace made in Section 4.3.2 and to map this onto an appropriate Ceph storage configuration.

5.2.1 Workload Analysis

The application trace of the Postmark workload showed that the workload consisted of 2658283 I/O operations, of which 78% (2072124) were read accesses. This suggests that a configuration that performs well during read accesses could positively improve performance.

The access size diagram for all accesses (see Figure 4.23) shows a peak at 128KB accesses. Other access sizes that appear to be used frequently are 4KB, 512KB and accesses larger than 512KB. The separate charts for reads (see Figure 4.24) and writes (see Figure 4.25) reveal more detail. The dominant access size is 128KB which is used in 91.1% of the read accesses. Accesses with 4KB, 16KB, 32KB and 64KB also occur frequently, as does 256KB, which is the largest recorded access size.

The writes show three access sizes with high frequency. The 4KB accesses account for 20.6% (121081), the 512KB and accesses larger account for 49.1% (287870) and 22.6% (132748), respectively. These three access sizes combined are used in 92.4% of all write accesses. Therefore, the requirements of the reads and writes for the Postmark workload differ with reads using 128KB accesses while writes make use of mostly 512KB and larger block sizes.

The distance between the accessed blocks depicted in Figure 4.26 shows a highly sequential access pattern. This is confirmed for the read accesses shown in Figure 4.27 and mostly confirmed for the write accesses (see Figure 4.28) which show a higher number of random accesses but still mostly sequential.

When the access sizes and distances are analysed and put into bins as described in

Table 5.2: Accesses of Postmark workload for the separate access sizes and randomness.

Total	4k read	32k read	128k read	1MB read	32MB read	% random read
2658283	57970	77350	1936804	0	0	5.8
	4k write	32k write	128k write	1MB write	32MB write	% random write
	142759	11790	10992	420618	0	25.3

Section 2.2.4 the accesses are combined to create the five access sizes with their read and writes as shown in Table 5.2. The information can then be used with the mapping algorithm to calculate the best performing configuration.

5.2.2 Mapping

The Postmark workload consists of mostly sequential 128KB reads, therefore configurations depicted in Figure 3.15 should theoretically best resemble the read access pattern of the workload. Thus, Configurations Q, X and N could potentially increase performance. For the write accesses, configurations depicted in Figures 3.20 and 3.18 should theoretically best resemble the write access pattern of the workload. In both cases Configuration K performs best. Other configurations potentially outperform the default, most notably Configuration K.

When the trace information is put into bins (see Table 5.2) and used in combination with the mapping algorithm a different picture appears. Without applying any weights to the accesses many configurations would predict a performance increase as shown in Figure 5.5. If the workload consists of purely random accesses, only two configurations (B, N) show a performance degradation of up to 2.5%. All others predict increased performance, with Configuration K indicating an improvement of up to 13%. For sequential accesses the alternative configurations mostly result in reduced performance. A few configurations (Q, X, K, R, N, B) predict a performance increase over the default between 2% (configuration Q) and 0.2% (Configuration B).

When the weights of randomness is applied, a better representation for the predicted performance results, as shown in Figure 5.6. With these weightings, six configurations show a performance increase, Configurations Q and X perform best with a predicted performance improvement of 1.9% and 1.6%, respectively. Remarkably, the prediction for Configuration K drops from 13% improvement to 1% improvement. Configuration T is predicted to perform worst and reduce performance by 4.6%.

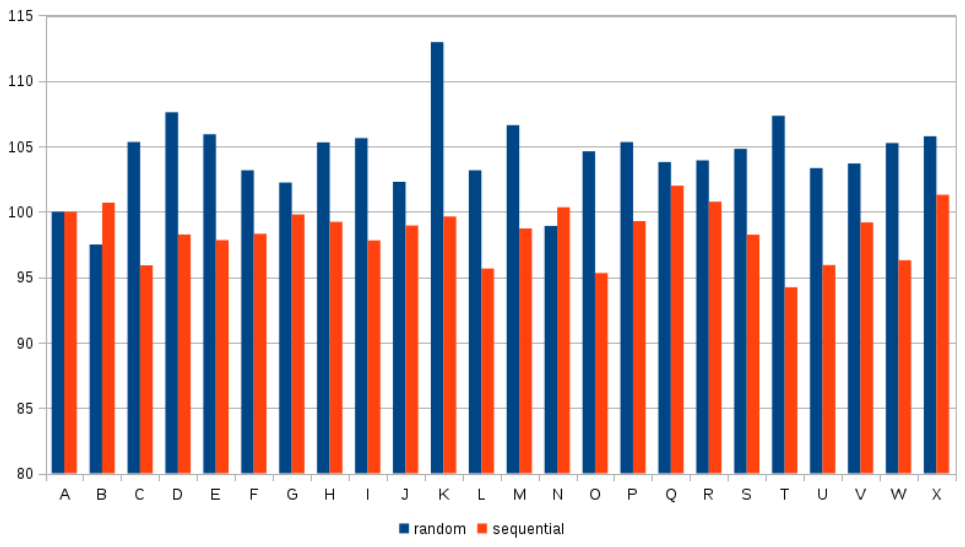


Figure 5.5: Configuration performance for the Postmark workload for sequential and random accesses.

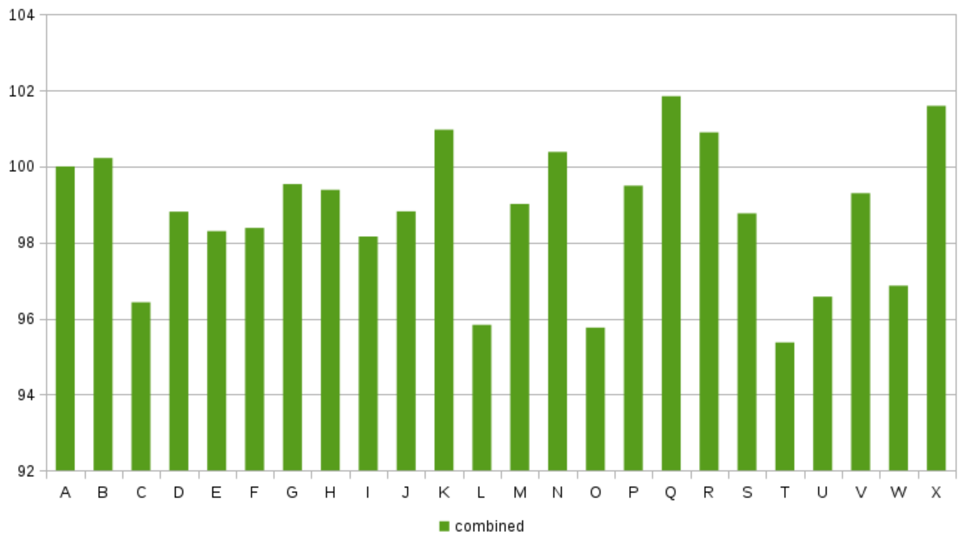


Figure 5.6: Configuration performance for the Postmark workload of combined sequential and random accesses with weights applied.

5.2.3 Results

The results of the mapping process were verified by replicating the workload concurrently on 12 virtual machines. The configurations tested were the default (Configuration A), Configuration T (worst) and Configuration Q (best).

In contrast to blogbench which is CPU bound, Postmark is I/O bound. Therefore, 12 concurrent replications of this workload resulted in significantly stressing the storage system. Perversely, the empirical results indicated less than one transaction per second (TPS) was executed (see Figure 5.7). The reason for this low I/O count, and almost

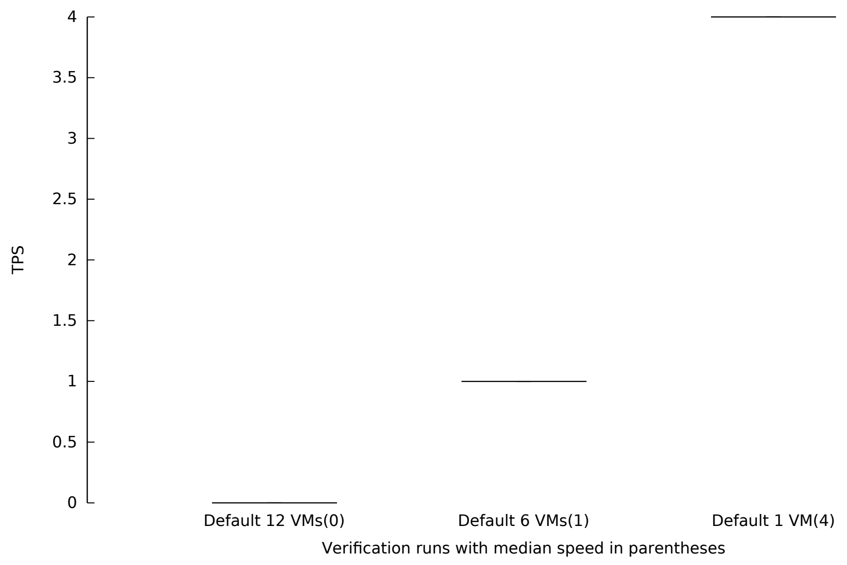


Figure 5.7: Verification of the different configurations under the postmark workload.

certainly erroneous reporting from the tracer, is that the system was too overloaded to be monitored correctly. With 6 concurrent virtual machines the throughput increased to 1 TPS per VM, which is not enough to see the predicted performance increase of around 2%. Testing with a single VM resulted in 4 TPS per VM, which again was too low to draw any meaningful conclusions. In summary, the storage backend was not fast enough to react to the Postmark access characteristics and consequently it was not possible to differentiate between the alternative configurations when differences in TPS per VM were so low.

5.3 DBENCH

The trace of the DBENCH workload shown in Section 5.3.1 is unusual since it consist of only write accesses. Only 0.03% of the accesses were reads. Nevertheless, they will be taken into account here for identifying candidate configurations, however, it is not expected that they will have a significant influence on the result.

5.3.1 Workload Analysis

The application trace of the DBENCH workload revealed that the workload produced 730578 storage accesses, of which only 209 accesses were reads. Therefore, a Ceph configuration that performs well for writes should positively influence the performance.

The dominant access size for the writes during the trace was 4KB, amounting to 54.7% of all write accesses, as shown in Figure 4.32. The second most used access size was 8KB.

Table 5.3: Accesses of dbench workload for the separate access sizes and randomness.

Total	4k read	32k read	128k read	1MB read	32MB read	% random read
730578	8	1	200	0	0	6.2
	4k write	32k write	128k write	1MB write	32MB write	% random write
	462157	75359	161292	31561	0	94.6

Both combined made up 63.3% of all write accesses, suggesting that a configuration that improves performance for 4KB accesses would improve workload performance.

The seek distance between successive write accesses shows a highly random pattern, as shown in Figure 4.33. 94.6% of the operations accessed blocks further than 128 blocks away. This is depicted in Figure 4.34 where accesses are shown in 4 regions of the disk.

When the access size information and seek distances are put into bins, as described in Section 2.2.4, a better picture emerges (see Table 5.3). These numbers can then be directly used in the mapping algorithm to determine the performance of alternative configurations relative to the default.

5.3.2 Mapping

The DBENCH trace reveals that the workload consists of mostly 4KB random writes, therefore the configurations best resembling this access pattern are shown in Figure 3.6. A characteristic of this figure is the closeness of the result; no configuration demonstrates a clear advantage over the default. Certain configurations (B, G, J, L) show performance degradation from 12 IOPS to 10.

When the binned data is used in the mapping algorithm, Configuration M predicts a performance improvement if all accesses were sequential, as shown in Figure 5.8. With the same assumption, Configuration N would perform worst, coming in at 21.7% below the default. If all accesses were random, multiple configurations (K, P, R, T, V, X) are predicted to outperform the default configuration by up to 3.6%. With the same assumption, Configuration L would perform worst, coming in at 13.9% below the default configuration.

With the appropriate randomness weights for reads and writes applied to the mapping algorithm, Configurations K, T and X show a predicted performance increase (see Figure 5.9), with X outperforming the default configuration by 7%. All other configurations decrease performance relative to the default configuration. The worst performing is configuration B, coming in at 13.3% below the default configuration.

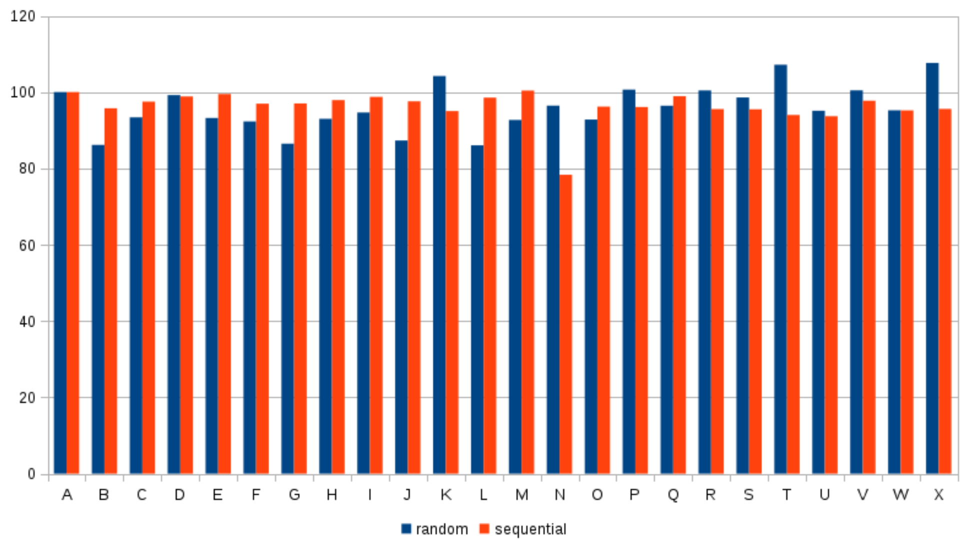


Figure 5.8: Configuration performance for the dbench workload for sequential and random accesses.

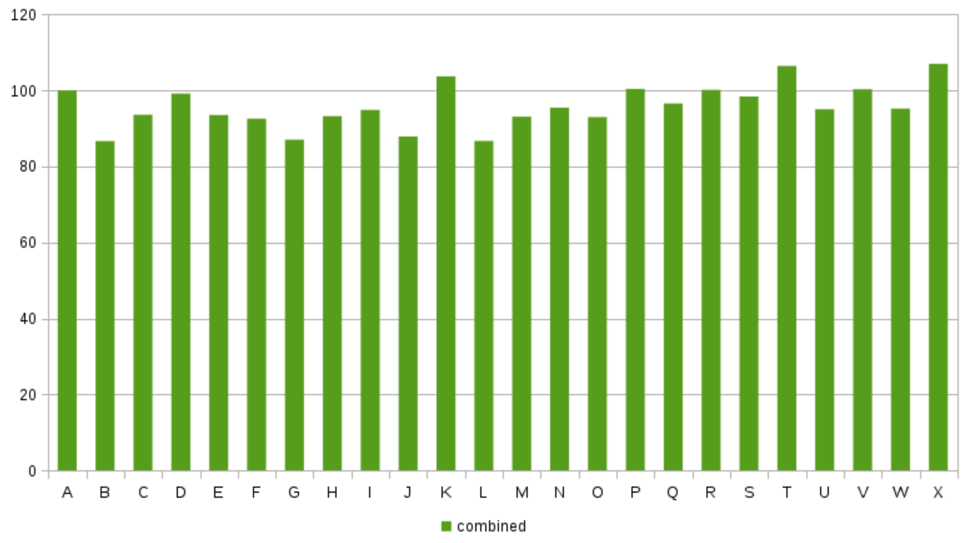


Figure 5.9: Configuration performance for the dbench workload of combined sequential and random accesses with weights applied.

5.3.3 Results

The results of the mapping process were verified by replicating the workload concurrently on 12 virtual machines. The configurations tested for the DBENCH workload with 48 clients were the default (Configuration A), Configuration B (worst) and Configuration X (best).

As depicted in Figure 5.10, Configuration X increased performance by 4.7% and Configuration B decreased performance by 16.7% relative to the default configuration. These results are in line with the predicted performance increases of 7% for Configuration X and the predicted decrease of 13.3% for Configuration B.

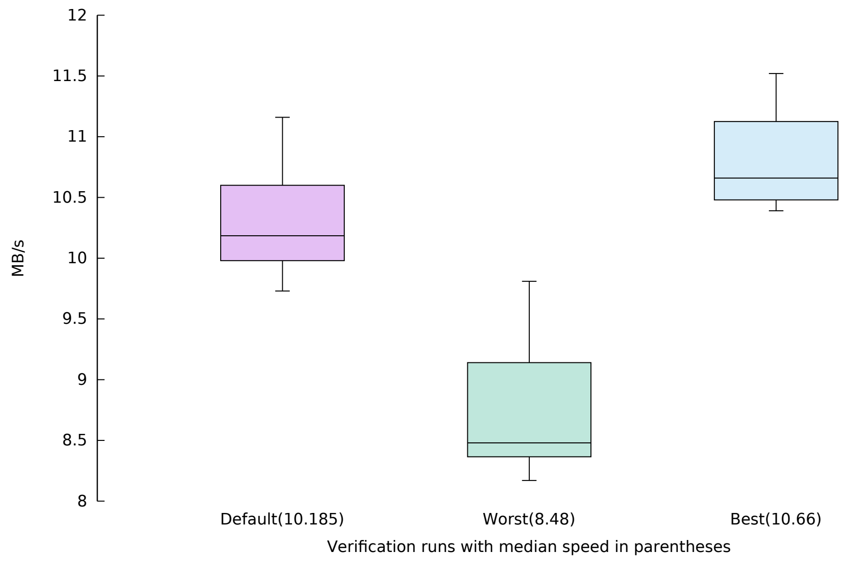


Figure 5.10: Verification of the different configurations under the dbench workload.

5.4 Linux Kernel Compile

Linux Kernel compilation can be used to represent the workload associated with application development and continuous integration. It makes use of many files across different folder, these are read from and subsequently written back to disk. To find a storage configuration tested in Section 3.4 that fits this workload, it is necessary to analyse the storage trace made in Section 4.3.4 and to map this onto an appropriate Ceph storage configuration.

5.4.1 Workload Analysis

The trace of the Linux Kernel compilation workload revealed that the workload produced a total of 14008 I/O operations; 14.6% were reads and 85.4% were writes. Such a distribution suggests that a configuration that improves write throughput may improve workload performance.

In the workload access size distributions, depicted in Figure 4.37, 4KB accesses were dominant and used in 40.1% of all accesses, while 8KB and 128KB each used 9.5% of all accesses, respectively. When looking only at the read accesses the distribution changes (see Figure 4.38). 128KB accesses are the most used access size with 52.6% while 4KB blocks were used in 36.2% of all read accesses. This is a significant difference to the write accesses shown in Figure 4.39. 4KB accesses are used in 40.8%, while 8KB, 32KB and 48KB block sizes each are used in ∼10% of all accesses. This means

Table 5.4: Accesses of Kernel compile workload for the separate access sizes and randomness.

Total	4k read	32k read	128k read	1MB read	32MB read	% random read
14008	776	105	1167	0	0	24.6
	4k write	32k write	128k write	1MB write	32MB write	% random write
	6176	3177	1147	1460	0	48.3

that the requirements for reads and writes are very different in their accesses sizes. This suggests that an optimal configuration will incorporate both the reading and writing requirements. In general, when an alternative configuration attempts to tune for multiple requirements, two approaches are possible:

1. Choose a configuration that differs in one parameter from the default but choose that parameter so as to balance the needs of conflicting requirements.
2. Alternatively create a new configuration in which multiple parameters are changed relative to the default and for which each parameter choice reflecting an orthogonal requirement.

Since the read and write requirements are not orthogonal to each other, approach 1 is taken here.

The read accesses display a highly sequential access pattern, as depicted in Figure 4.41, with 75.0% of the read accesses being made to the next contiguous block. Augmented by accesses with a seek distance of below 128, the total number of sequential like accesses is 75.8%. For write accesses the number of sequential accesses dropped to 51.7% of the total, as shown in Figure 4.42.

As shown in Figure 4.44, data is only read once but written three times. The reason for this difference is the way the trace and the benchmark was executed. The trace consists of three runs of compiling the Linux kernel. During the first run files are read from disk, but for each subsequent run some files may be available in the operating system cache.

When the access size information and seek distances are put into bins, as described in Section 2.2.4, a better picture emerges (see Table 5.4). These numbers can then be directly used in the mapping algorithm to determine the performance of alternative configurations relative to the default.

5.4.2 Mapping

The Linux Kernel compile workload consists mostly of a mixture of sequential and random write accesses with a 4KB block size. The difference in performance shown in

Figures 3.8 and 3.6 are not significant to make a recommendation for a configuration as they all perform similarly to each other. Adding 32KB write accesses, the second most common bin shown in Table 5.4, does not change this situation. A clear recommendation that can be made is to not use Configuration N since that configuration leads to a degradation in performance of up to 25% relative to the default.

When the binned data is used in the mapping algorithm, only a few configurations show a predicted performance increase when assuming only sequential or only random accesses, as depicted in Figure 5.11. If all accesses were random, four configurations (K, P, T, X) would show a performance increase by up to 5% relative to the default. If all accesses were sequential, only Configuration Q would be able to marginally outperform the default by 0.2%.

With the appropriate randomness weights for reads and writes applied to the mapping algorithm, none of the configurations would be able to outperform the default (see Figure 5.12). As predicted above, Configuration N would show the lowest performance relative to the default, reducing performance by 10.6%, and Configuration X would loose 0.5%. This suggests that keeping the configuration at default would result in the best performance for this workload.

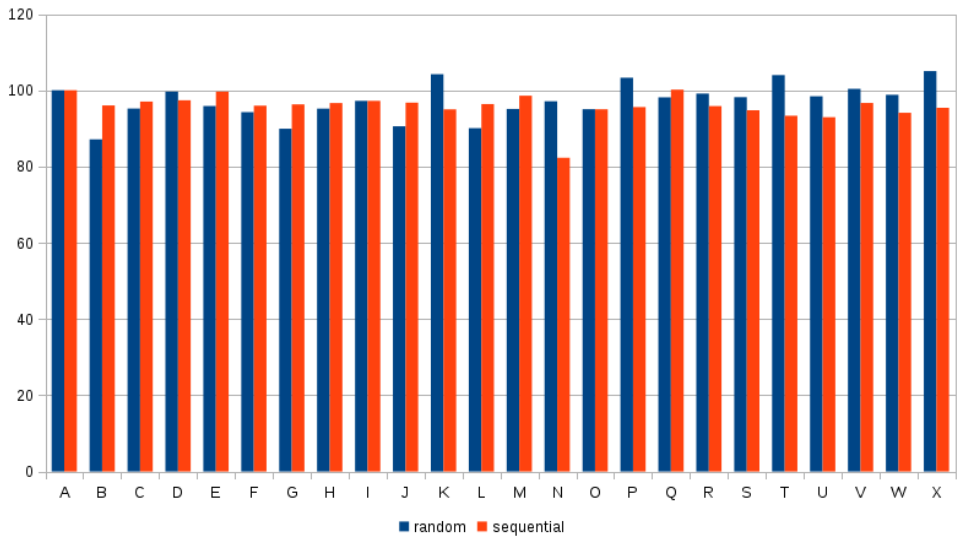


Figure 5.11: Configuration performance for the Linux Kernel compilation workload for sequential and random accesses.

5.4.3 Results

The results of the mapping process were verified by replicating the workload concurrently on 12 virtual machines. The configurations tested for the Linux Kernel compile workload were the default (Configuration A), Configuration N (worst) and Configuration X (best). The mapping algorithm tells us that Configuration X is the best

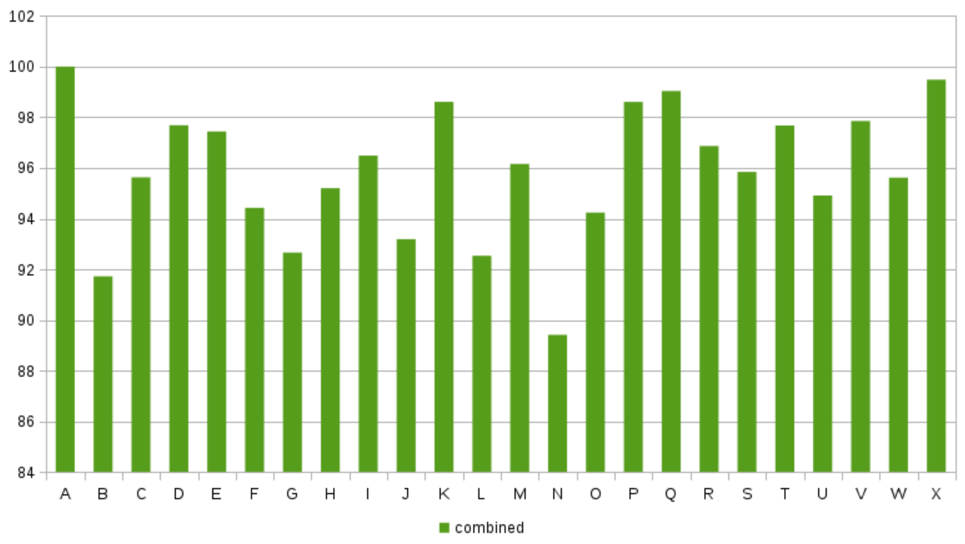


Figure 5.12: Configuration performance for the Linux Kernel compilation workload of combined sequential and random accesses with weights applied.

alternative configuration but is predicted to perform worse than the default. As mentioned in Section 5.4.1, the workload is CPU bound. Figure 5.13 shows the performance differences between the configurations, identified in the mapping process, when tested against the Linux Kernel compile workload.

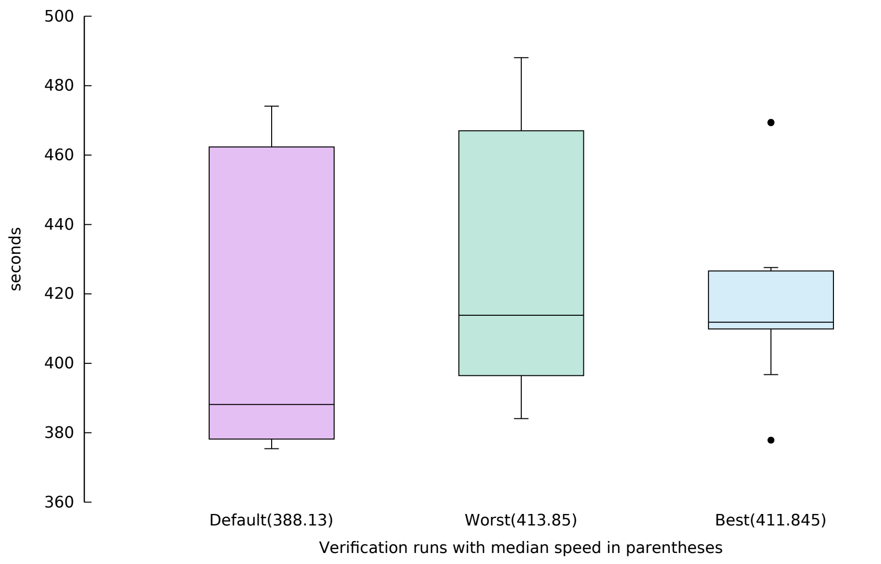


Figure 5.13: Verification of the proposed Linux Kernel compile configurations.

The results of the verification with 12 virtual machines confirm the performance predictions (see Figure 5.13), where no configuration is able to outperform the default. Configurations N and X perform almost identically, which does not match the performance predictions which suggested a performance decrease of 0.5% and 10.6% relative to the default, respectively.

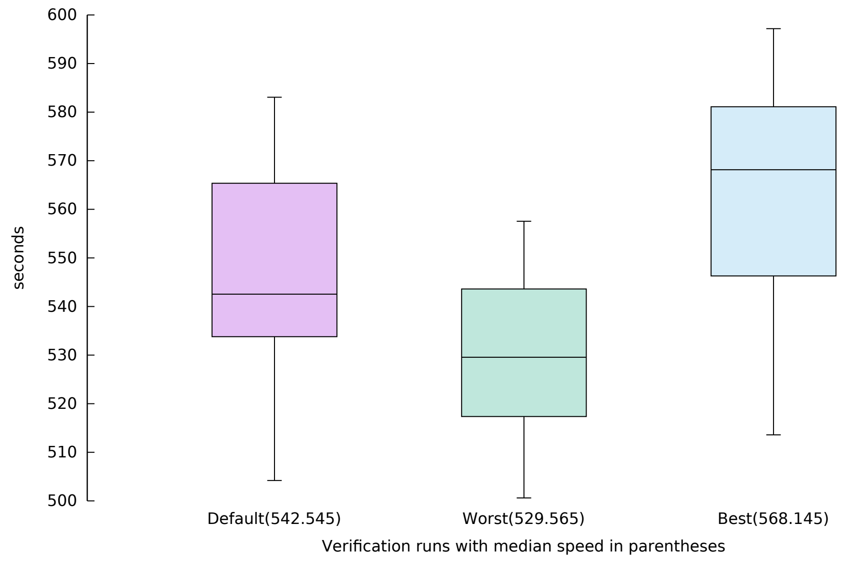


Figure 5.14: Verification of the proposed Linux Kernel compile configurations using 18 VMs.

Following the methodology described in Section 2.2.4, which attempts to get a better insight into CPU bound workloads, the experiment was rerun but this time replicating the workload on 18 VMs (6 per host with no over-provisioning on the host) and the results are depicted in Figure 5.14. These results were initially surprising since they completely inverted the predictions; the best alternative became the worst and the worst became the best. On closer inspection, an inaccurate assumption made in the methodology became apparent. It was assumed that increasing the workload from 12 VMs to 18 would simply change the granularity of the results so that trends would become more discernible. On reflection, this is a fallacy since it fails to take into account the operating system scheduling policies and the storage system software stack and the effects that these might have on the replicated workloads. When the replication count is higher, these software layers may well attempt to tune the access patterns in ways that are transparent to the tracing module. In effect, the workload of 18 VMs must be considered as being fundamentally of a different character. It is assumed that the differences become explicit in the experiment run here and not in the blogbench workload because Linux Kernel compile is not as CPU bound as blogbench. Therefore, to perform a valid prediction one would have to recreate the baselines mentioned in Section 2.2 using 18 VMs and to make predictions based on that new data.

Therefore, while the relative position of the best and worst predicted configurations was verified by experiment, these predictions were not able to better the default for the 12 VMs.

Table 5.5: Accesses of pgbench workload for the separate access sizes and randomness.

Total	4k read	32k read	128k read	1MB read	32MB read	% random read
2082155	676499	43187	257391	0	0	70.0
	4k write	32k write	128k write	1MB write	32MB write	% random write
	876568	160753	47187	20570	0	76.7

5.5 pgbench

To find a storage configuration tested in Section 3.4 that fits the pgbench workload, it is necessary to analyse the storage trace made in Section 4.3.5 and to map this onto an appropriate Ceph storage configuration.

5.5.1 Workload Analysis

The application trace of the pgbench workload revealed that it produced 2082155 I/Os in the trace length of 4360 seconds. 46.9% (977077) of these accesses were read operations. Making a recommendation based on the read and write ration is therefore not possible.

In the workload access size distributions, depicted in Figure 4.48, 8KB accesses were dominant and used in 74.3% of all accesses. 128KB accesses were recorded for 12.8% of the accesses. For read accesses (see Figure 4.49), these two access sizes are still the dominant ones, but the frequency is changed to 68.8% (8KB) and 26% (128KB), respectively. For write accesses (see Figure 4.50), an access size of 8KB was used in 79.1% of all write accesses. 128KB accesses were only used in 1.1% of the accesses while an access size of 32KB occurred in 9.3% of all write accesses. Configurations that improve performance for these block sizes should therefore be able to improve performance.

As shown in Figure 4.51, the workload displays random access characteristics; 82.5% (1717658) of the I/Os accessed blocks further than 128 blocks distant. This pattern is depicted in Figure 4.52 were the accesses are spread out over more than 1/3 of the 100GB virtual disk. The read accesses displayed 70% random accesses (see Figure 4.53) while write accesses were 76.7% random (see Figure 4.54).

When the access size information and seek distances are put into bins, as described in Section 2.2.4, a better picture emerges (see Table 5.5). These numbers can then be directly used in the mapping algorithm to determine the performance of alternative configurations relative to the default.

5.5.2 Mapping

When the pgbench workload accesses are put in the appropriate bins, 69.2% of the read accesses are mapped to an access size of 4KB. With a randomness of 70% the 4KB random reads should represent the workload read accesses best (see Figure 3.5). The default configuration, Configuration E, Configuration Q and Configuration R performed best for this access pattern with only small differences between them. A configuration that should not be considered for the workload is Configuration B, since it performs significantly worse than all other configurations.

The write accesses use mostly random 4KB accesses, but with small differences between them no meaningful recommendation is possible (see Figure 3.6). The same is true for the 32KB accesses (see Figure 3.10). As a consequence, a recommendation can not be made for the write accesses based on these two dominant access sizes.

When the binned data is used in the mapping algorithm, only Configuration T and Configuration X are predicted to show a performance improvement if all accesses were random, as shown in Figure 5.15. If all accesses were sequential, no alternative configuration is predicted to improve performance over the default. With the appropriate randomness weights for reads and writes applied to the mapping algorithm, none of the configurations is predicted to outperform the default configuration (see Figure 5.16). The smallest decrease in performance of 1.6% is predicted for Configuration Q, and the largest performance decrease of 16.9% is predicted for Configuration B. Therefore, the prediction would suggest that for the pgbench workload the configuration should not deviate from the default, since all other configurations predict a performance reduction.

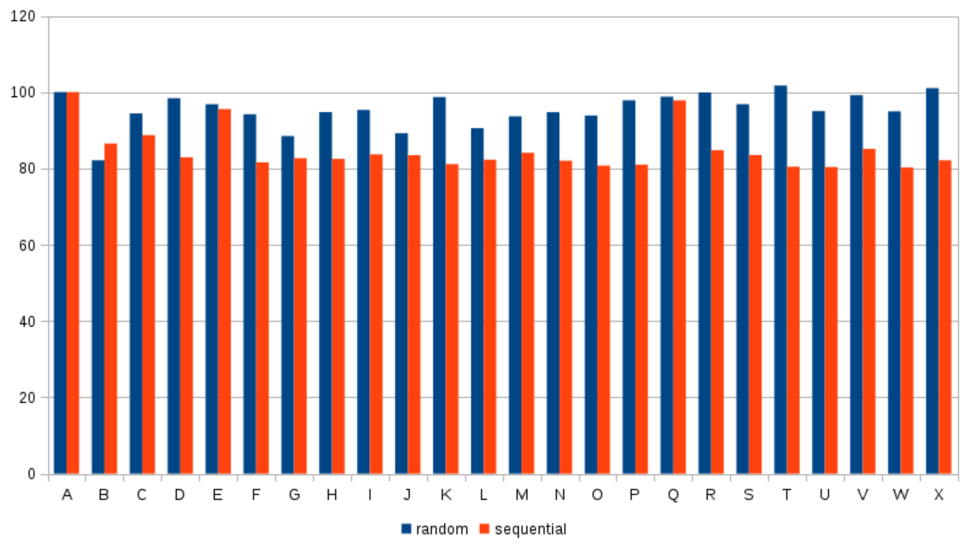


Figure 5.15: Configuration performance for the pgbench workload for sequential and random accesses.

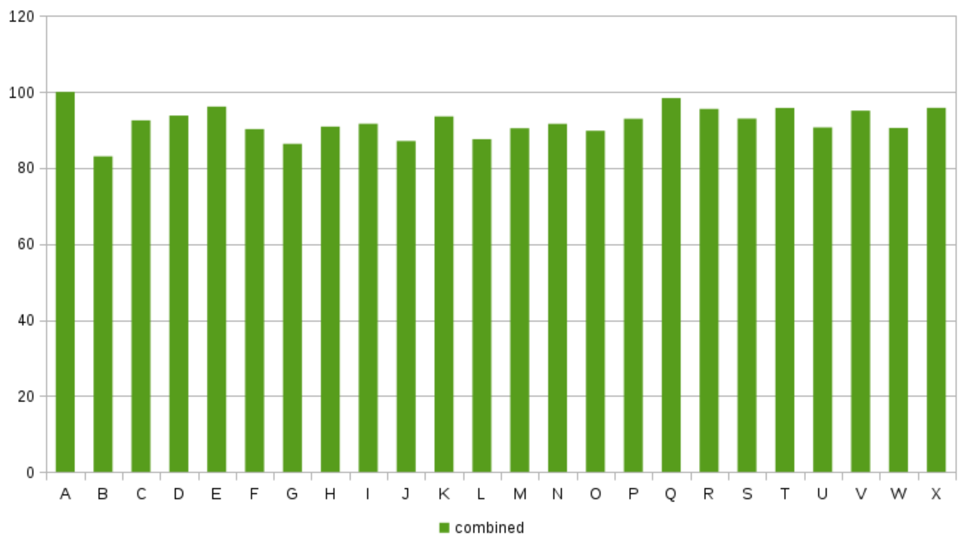


Figure 5.16: Configuration performance for the pgbench workload of combined sequential and random accesses with weights applied.

5.5.3 Results

The results of the mapping process were verified by replicating the workload concurrently on 12 virtual machines. The configurations tested for the pgbench workload were the default (Configuration A), Configuration B (worst) and Configuration Q (best).

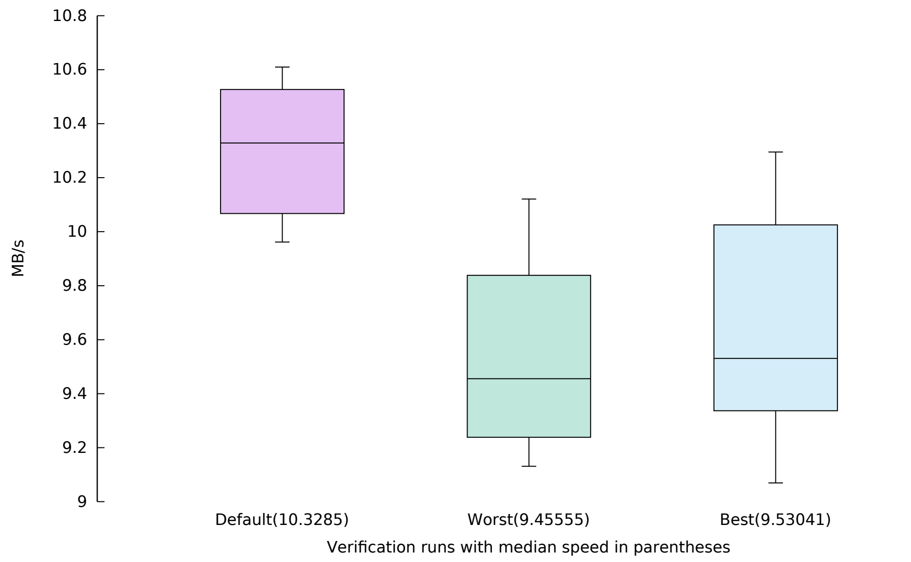


Figure 5.17: Verification of the proposed pgbench configurations.

As depicted in Figure 5.17, the default configuration performed best for the pgbench workload and confirmed the prediction made in the mapping. Configuration Q reduced performance by 7.7%, which is higher than the projected loss of 1.6%. Configuration B reduced performance by 8.5%, which is better than the projected loss of 16.9%.

Workload	CPU bound	default	predicted best	predicted worst	measured best	measured worst	Δ best	Δ worst
blogbench	✓	293.5	0.9%	-16.2%	294.5	296.5	0.3%	1.0%
blogbench (18 VMs)	✓	272	0.9%	-16.2%	277	270	1.8%	-0.3%
Postmark	N	0	1.9%	-4.6%	N/A	N/A	N/A	N/A
DBENCH	N	10.185	7%	-13.3%	10.66	8.48	4.6%	-16.7%
Linux Kernel compile	✓	388.13	-0.5%	-10.6%	411.845	413.85	-6.1%	-6.6%
Linux Kernel compile (18 VMs)	✓	542.545	-0.5%	-10.6%	568.145	529.565	-4.7%	2.4%
pgbench	N	10.3285	-1.6%	-16.9%	9.53041	9.45555	-7.7%	-8.4%

5.6 Summary

In this chapter a mapping between the workload characteristics and the differently performing configurations was performed and tested empirically. In this process, a configuration, that would improve workload performance, was found in four of the five examined workloads. In one case (pgbench), the default configuration was predicted to be the best performing, which was shown in the empirical examination. In two cases (blogbench, Linux Kernel compile) the workload was CPU bound and performance differences could not be conclusively attributed to performance differences of the underlying storage configuration. For one workload (Postmark) the performance variations arising from different storage configurations could not be evaluated due to the limited performance of the storage system and small predicted performance differences. For the DBENCH workload, the results of the empirical verification of the mapped configurations is in line with the predicted performance gains and losses.

The relative positions of the chosen configurations were in line with the predictions for all workloads, suggesting that the prediction of the best performing and the worst performing configurations had merit. As mentioned in Section 5.4.3 the anomaly identified for 18 VMs in the Kernel compile workload can be explained as an inaccurate procedural assumption rather than a poor prediction.

The default configuration is performing well across all workloads. This resiliency to the changes made can be attributed to three different reasons. The first reason lies in the scale of the testbed. It is possible that the testbed was not too small to highlight scaling limitations of specific parameters. The second reason lies in the used hardware technology. Ceph has been designed to operate well with mechanical hard drives. When using SSDs the system characteristics change and bottlenecks in the system become apparent that do not affect mechanical hard drives. In deployments that use SSDs for the disk journals or when using only flash drives, these configuration parameters become serious bottlenecks that limit performance [41]. The third reason can lie in the choice of parameters used in the parameter sweep. It is possible that the parameters are highly connected to other parameters rather than being independent. A configuration that has evolved over time can be ruled out, since the parameter values have not been changed during the development of the system since it was uploaded to the version control platform GitHub [111].

Chapter 6

Conclusion

Ceph is a highly configurable distributed storage system. Finding an optimal configuration is a non-trivial task and requires deep knowledge of the inner workings of the system and empirical experiments to create a configuration that improves storage performance. Since the effects of changes to the Ceph configuration are not well documented, a structured process has to be applied to find a configuration that improves performance in a given testbed and environment.

In Appendix B the *ad hoc* approach taken to improve the storage system found in the literature is explored on the testbed constructed for this investigation. The procedure adopted in the literature for testing proposed new configurations is extended to get a better insight into storage configuration performance by increasing the sample size for the baseline performances from two, as used by the related work, to four. In the literature different configurations are proposed without clear reasons describing why those specific changes were made. Consequently, it is impossible to objectively discern the rational behind these changes nor is it clear how changes to the system should be proposed and implemented in a structured manner.

The results from the author's testbed show that while certain configurations result in improvements in restricted circumstances, none of the configurations proposed in the literature is capable of improving storage performance across a range of access patterns. Surprisingly, it was found in the test performed here that in some cases these alternative configurations resulted in a performance disimprovement. When a new configuration for Ceph is proposed in the literature, it is invariably presented as being a general improvement across a broad range of access patterns and sizes. The empirical study performed here indicated that this was not the case. For the DBENCH workload, for example, it was shown that proposed configuration changes resulted more often in a storage system that performed worse than the default configuration.

Ad hoc configurations of this kind to improve performance are prevalent in the literature and are accepted by the community on the basis of results derived from synthetic workloads and considering at most two access sizes. The insights gained from the empirical studies performed here derive from the fact that a real workload with varying access patterns and sizes were used while investigating these proposed configuration alternatives.

Consequently, it became clear that an approach to performance improvement based on workload characterization would form a stronger basis for proposing alternative configurations. The work presented here thus considers the construction of a mapping algorithm to identify appropriate storage system configurations based on workload characteristics derived from actual trace data. Moreover, the experiments performed here are done over a greater range of access sizes (four in the case of the empirical investigation of the literature and five when exploring the effectiveness of the proposed mapping procedure) and so show a more complete picture of the stresses placed on the storage system and on the way it reacts over this extended range.

The extended range of experiments resulted in a total of 20 different combinations of access sizes and patterns to accurately determine the performance differences under these access patterns typically used by cloud workloads. For generating the storage trace, multiple tools were investigated and presented. In this work, five workloads were chosen as representable cloud workloads and storage traces were captured and analysed for their respective access patterns and sizes. Further metrics were captured and analysed but not used in the presented work.

The presented mapping algorithm creates a performance prediction by combining the extracted workload characteristics and the storage performance of the different access patterns and sizes to increase overall workload performance. The empirical verifications of these predictions were subsequently tested for correctness. The results observed in those empirical verifications did not exactly match the predicted changes, but the relative position to the default configuration was in line with the predictions, suggesting the predictions had merit. The inability to produce 100% accurate predictions is not surprising, since such predictions depend on very many factors, most of which are resident in the greater Ceph environment, and hence outside the controls and tracing tools of the testbed.

In the methodology, an assumption about the scaling behaviour of VMs and the corresponding storage load was made to observe the impact of different storage configurations for workloads that are CPU bound. The experiments performed with an increased VM count did not exhibit results that were in line with the performance predictions. As mentioned in Section 5.4.3, this is most probably due to the change in characteristics when scaled to a number that does not match the baseline performance experiments, since the scaling changes the characteristics of the I/Os that arrive at the storage system. Further work would therefore be necessary to investigate scaling effects of workloads when using a distributed file system, such as Ceph. This might be achieved by testing the different storage configurations with multiple VM counts to extract scaling characteristics to improve the mapping process to allow for predictions at different scales.

The process of creating the different Ceph configurations required logical partitioning of the configuration space. Thus, three partitions, namely functional component, Ceph environment and greater Ceph environment, were created to indicate where the changes could be made and the effects that these changes could have within the system. Changes to the functional component are limited to that component, but would not affect other parts of the system, whereas changes to the Ceph environment affect all instances of the functional components. The greater Ceph environment captures components that are not part of Ceph but host the Ceph cluster that is subjected to the constraints and capabilities of these components.

In this work, the impact of changes to the Ceph environment on pool performance was investigated and used in the process of mapping various workloads to Ceph configurations in an attempt to maximize performance of that pool. It was thought that changes made to environment parameters would have a bigger effect on performance than changes to the parameters of the functional components. Moreover, these environment parameters interface closer with the workload characteristics and hence changes to those parameters could be meaningfully inferred from those characteristics.

The impact of changes to the greater Ceph environment were explored and empirically tested for two cases. In the first, changing the file system on a subset of the storage devices lead to the creation of heterogeneous pools, pools that differ in their underlying configuration but are part of the same storage cluster and share the same Ceph environment and configuration of the functional component. In the second case, the performance differences resulting from changes to the operating system I/O scheduler were analysed. The results of these explorations were not used in the mapping process, but the results indicate that a correct setup of the greater Ceph environment could result in significant performance improvements for different cluster sizes, hardware and workload.

The contributions of this thesis can be summarized as follows:

- A methodology for mapping workloads to Ceph configurations was created.
  - A structured process was used rather than an *ad hoc* process as used in the literature.
  - Alternative storage configurations across 20 access sizes and access patterns were examined to determine the baseline performances for those access sizes and patterns to increase the precision of the evaluation.
  - A workload characterization for multiple representative cloud workloads was

performed based on access I/O size and randomness.

  - A mapping of these cloud workloads to appropriate Ceph configurations was performed and performance differences were evaluated.

- An experimental exploration of related work was undertaken to determine the performance impact of configurations proposed in the literature on performance.
- The Ceph configuration space was envisioned as three logical partitions:
  - the greater Ceph environment, including the operating system, the I/O scheduler, hardware used and the workload, i.e., all components outside of the control of Ceph.
  - the Ceph environment, capturing all parameters of the Ceph configuration that are not part of the functional components.
  - the functional component, representing the entity to be improved, such as the Ceph pool.
- Heterogeneous Ceph pools were created that share one Ceph environment and one functional component configuration, but contain different components in their greater Ceph environment, such as:
  - the file system deployed on each of the OSDs,
  - the I/O scheduler and scheduler queue size of the operating system.

6.1 Future Work

The work presented here suggests multiple opportunities for extending and improving the proposed methodology with view to workload driven performance improvement.

Modify the pool configuration and keep both Ceph environments

In this work the impact of changes in the Ceph environment on pool performance was investigated. Alternatively, the Ceph environment and greater Ceph environment could be fixed and changes could be made to the functional component. As stated above, in this work the changes to the Ceph environment were investigated since they were perceived to offer greater opportunities for improvement, potentially resulting in higher performance improvements. Changes to the functional components themselves, although limited in number, could alter the performance of those functional components. The impact of such changes are worthy of investigation, since they could be made for each pool separately without changing the characteristics of other pools. Furthermore, these changes could be applied to and could change characteristics of tiered pools, which may be a cost effective way to improve performance for certain workloads.

Testing with multiple VM counts for better performance representation

The verification of the mapping procedure revealed a false assumption. With an increased number of concurrent VMs, the load did not scale linearly and accurate predictions would require the construction of baselines corresponding to this new scale. Therefore, an extended baseline performance collection would be required to improve mapping quality and coverage. Furthermore, a larger baseline collection might allow for the extraction of trends for specific storage configurations, such as Configuration X performing well for random write 4KB accesses and various VM counts.

Tune for metrics other than throughput, such as latency or I/O queue depth

In this work tuning was performed with the intention of improving storage throughput to improve workload performance. While this is a highly influential factor, it might not improve performance for all workloads. Workloads that require low latency storage would not necessarily experience improved performance, since I/Os are held back by the I/O scheduler to increase storage throughput at the cost of increased latency. A tuning based on I/O latency could therefore be a better approach for such workloads, since the workloads are not constrained throughput.

Another opportunity for tuning could include the manipulation of the I/O queue depth for workload accesses. Since higher queue depths allow for better reorganization of I/Os, the performance of the distributed file system may differ in a similar fashion. A baseline performance analysis of multiple queue depths, such as 1, 16 and 32, might result in a better understanding of the behaviour of different storage configurations when presented with I/Os of various queue depths.

In both of these different approaches, the impact of specific parameter changes may vary and significantly impact performance. Especially when tuning for latency, multiple queues are in the storage path (VM, storage host, Ceph, storage controller, disk) and reorganize operations and potentially add latency to each request. Applying changes in one place might therefore not result in an optimal, but a partial improvement.

Investigation of the impact of orthogonal configurations when tuning for reads and writes separately on overall performance

In this work an approach of improvement for overall workload performance was taken. Rather than aiming for overall best performance, one could tune for reads and writes separately and combine the respective configurations afterwards. Since configurations tend to improve certain characteristics of the storage system, the combined configurations may conflict and may result in improvements in one access pattern being nullified by disimprovements in the other.

Investigating the impact of different randomness points in the workload characterization on the mapping process

The definition of randomness was adopted from the literature where it was defined as being a head movement of 128 LBNs from one access to the next. If this value were changed, the randomness of the workload accesses in the mapping process would change accordingly, recategorizing the sequential and random designation of a workload, thus altering the randomness weights applied in the prediction algorithm and resulting different candidate configurations.

Checking for the impact of configurations constructed from combining performance improving configurations

In the empirical verification of the predicted workload storage configurations, configurations were constructed by changing a single parameter so that it differed from the default. In this way multiple configurations were constructed and each was used as the target for the mapping process. The chosen configuration was invariably that which was predicted to give the best performance. In fact, in many cases multiple configurations were predicted to improve performance. In future, rather than choosing a single configuration as the output of the mapping process, a combination of all performance improving configurations could be chosen instead. As mentioned previously, the aggregation of all of these alternatives may not necessarily result in aggregating all of their individual performance improvements. For a given workload a combination of multiple parameter changes, each resulting in a performance improvement, may conflict due to some specific characteristics of that workload and so the results are a priori unpredictable. Associating workload characteristics with combinations of parameters that constructively and destructively interact would be a fruitful area of further investigation. In addition, it may also be fruitful to consider the construction of alternative configurations by altering multiple parameters simultaneously so that each differs from their default.

Balance the environment configurations so as to support the needs of all pools simultaneously

Since a storage system is rarely used for a single workload, future work could evaluate the impact of specific configurations that increase performance for a single workload on other pools and workloads combined, with the aim of achieving a configuration of the Ceph environment and greater Ceph environment that leads to an optimal global configuration, supporting a great number of different workloads simultaneously.

Remove testbed networking limitations

The limitations of the testbed put constraints on cluster performance. In a future evaluation the testbed could be equipped with sufficient network bandwidth to support appropriate cluster throughput, since the cluster network constrains the replication throughput required during write accesses. In this work a reduced replication count was used to reduce the impact, but storage clusters that experience large numbers of write accesses should be designed with sufficient network bandwidth, since performance gains might otherwise not become apparent.

6.2 Epilog

The work presented here strives to provide a structured approach to performance enhancements in the Ceph storage system over the *ad hoc* approach taken in the literature. Rather than basing alternative configurations on synthetic benchmarks that deliver performance improvements over a narrow range of access sizes and access patterns, efforts were made to firstly characterize workloads for access sizes and patterns. These characterizations allowed for the construction of a mapping process which predicted configurations that yielded performance improvements. Although the work presented in this dissertation was limited by the constraints of the testbed available to the author, the empirical investigation demonstrated that performance improvements largely resulted from the suggested alternative configurations. In a number of situations the predictions resulted in a performance disimprovement. Nevertheless, the predicted configuration was the best of the alternatives available. This disimprovement, and indeed the limited improvements found in this investigation can be explained by the vast number of parameters outside of the control of the Ceph system, which directly influence the performance of that system. Thus, to ensure worthwhile improvements when investigating alternative Ceph configurations a holistic view of the installation from the greater Ceph environment down to the parameters of the Ceph functional components, should be considered.

The Ceph system is, as shown in this work, very complex. Considering this complexity, the performance of the default configuration is surprising, since it has not changed since the system was created. The opaqueness of the effects and impacts of changes to the configuration make the task of creating better configurations difficult and non-trivial. However, there is evidence in this work that justifies the application of a tuning process to improve performance for a given workload type. Moreover, it is anticipated that these improvements will become more pronounced at scale as modest improvements aggregate to deliver cumulative gains.

Appendix A

OpenStack Components

A.1 OpenStack Compute - Nova

OpenStack compute (Nova) is a major component in an Infrastructure-as-a-Service deployment. It interacts with the identity service for authentication, the image service to provide disk and server images and the OpenStack webinterface. OpenStack Nova consist of multiple components:

nova-api accepts and responds to end user compute API calls. It supports the OpenStack Compute API, the Amazon EC2 API, and a special Admin API for privileged users to perform administrative actions.

nova-cert serves Nova Cert services for X509 certificates used for EC2 API access.

nova-compute is the worker daemon that creates and terminates Virtual machines via a hypervisor API, such as `libvirt` when using QEMU/KVM.

nova-conductor is a mediator between the nova-compute service and the database to prevent direct database accesses of the compute service.

nova-consoleauth provides authentication tokens used in the webinterface to get console access through the vncproxy in the webinterface.

nova-scheduler determines the location where a new VM should be created based on scheduling filters.

nova-vncproxy provides VNC access to running VM instances, accessible through the webinterface Horizon.

A.2 OpenStack Network - Neutron

The OpenStack networking service Neutron is responsible for creating and attaching virtual network interfaces to virtual machines. These virtual interfaces are connected virtual networks. It is possible to create private isolated networks for each user and to assign floating IP addresses to individual VMs to expose them to the public network. VMs that have no floating IP can access the public network through routers.

Neutron is designed to be highly modular and extensible. It consists of the Neutron server that accepts and routes API requests to the appropriate Neutron networking plugins and agents where they will be processed. The plugins and agents differ in their implementation and function depending on the vendor and technology used. Network equipment manufacturers provide plugins and agents that support physical and virtual switches. Currently Cisco virtual and physical switches, NEC OpenFlow, Open vSwitch, Linux bridging and VMware NSX plugins and agents are available, but plugins and agents for other products can be developed through the well documented specification.

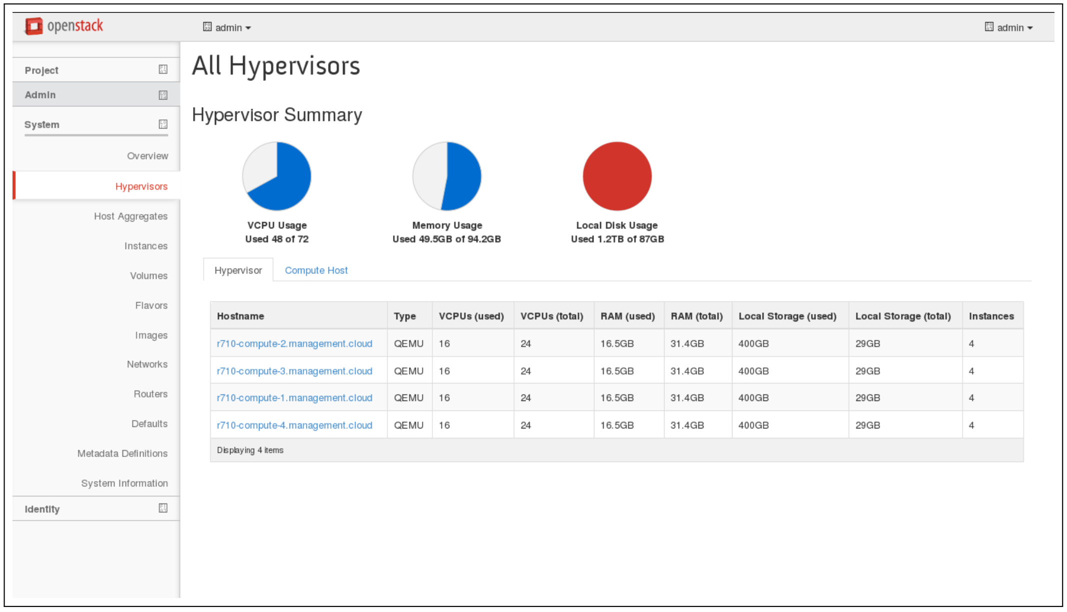
A.3 OpenStack Webinterface - Horizon

The OpenStack Dashboard is a modular Django web application that gives access to the different components of OpenStack (see Figure A.1). Users can create and terminate VMs and access the VNC console of running VMs. Furthermore, it gives access to the different OpenStack storage services (Glance, Cinder, Swift) to upload and download VM images, block device images and data objects.

A.4 OpenStack Identity - Keystone

The OpenStack identity service Keystone provides a single point of integration for managing authorization, authentication and a catalogue of services. It can integrate external user management systems, such as LDAP, to manage user credentials. Authentication in OpenStack is required for users accessing OpenStack services and for the communication and interaction of the OpenStack services themselves. Without appropriate authentication, communication between the components is rejected.

Keystone is typically the first component to be set up as all other OpenStack services depend on it. It is also the first component users interact with, since access to services is only granted when the user is properly authenticated and is allowed to access the specific component. To increase system security, each OpenStack service has multiple types of service endpoints (admin, internal, public) that restrict operations.



All Hypervisors

Hypervisor Summary

VCPU Usage
Used 48 of 72

Memory Usage
Used 49.5GB of 94.2GB

Local Disk Usage
Used 1.2TB of 87GB

Hostname	Type	VCPUs (used)	VCPUs (total)	RAM (used)	RAM (total)	Local Storage (used)	Local Storage (total)	Instances
r710-compute-2.management.cloud	QEMU	16	24	16.5GB	31.4GB	400GB	29GB	4
r710-compute-3.management.cloud	QEMU	16	24	16.5GB	31.4GB	400GB	29GB	4
r710-compute-1.management.cloud	QEMU	16	24	16.5GB	31.4GB	400GB	29GB	4
r710-compute-4.management.cloud	QEMU	16	24	16.5GB	31.4GB	400GB	29GB	4

Displaying 4 items

Figure A.1: OpenStack horizon webinterface displaying resource utilization of the deployed VMs on a cluster level and for each host individually.

A.5 OpenStack Storage Components

OpenStack has three different storage services: Glance, Cinder and Swift. Each of them has different requirements and is used in different ways.

A.5.1 OpenStack Image Service - Glance

The OpenStack image service Glance is an essential component in OpenStack as it serves and manages the virtual machine images that are central to Infrastructure-as-a-Service (IaaS). It offers an RESTful API that can be used by end users and OpenStack internal components to request virtual machine images or metadata associated with them, such as the image owner, creation date, public visibility or image tags.

Using the tags of the images should allow an automatic selection of the best storage backend for an individual VM. When an image is tagged with an database tag the storage scheduler should be able to automatically select the appropriate pool to host the VM, as it does for other components, such as the amount of CPU cores or the memory capacity. If the tag is absent, the image will be hosted on the fall-back backend, which uses a standard or non-targeted configuration for general purpose scenarios. When users use the correct tag they benefit, since the operator can potentially increase the number of users that can be hosted without risking overall storage performance degradation.

A.5.2 OpenStack Block Storage - Cinder

Cinder, the OpenStack block storage service, is used to either create volumes that are attached to virtual machines for extra storage capacity that show up as a separate block device within the VM, but it can also be used directly as the boot device. In that case the image from Glance will be converted/copied into a Cinder volume. After it has been flagged as bootable it can be used as the root disk of the VM. The backends for Cinder cover a great variety of systems [145], including proprietary storage systems such as Dell EqualLogic or EMC VNX Direct, distributed file systems, such as Ceph or GlusterFS, and network shares using Server Message Block (SMB) or NFS.

Normally it would be necessary to have at least two dedicated storage systems available when all three storage services are desired. Having two distinct storage systems does not allow the flexibility to deal with extra capacity demands on individual services as the hardware has to be partitioned when the system is rolled out. Currently there are two storage backends available that can be used for all three services within one system with the support for file-level storage, which is required for supporting live-migration between compute hosts. These are Ceph (see Section 1.2) and GlusterFS [1]. By using one of these storage backends it is possible to consolidate the three services on a single storage cluster and to keep them separated through logical pools.

A.5.3 OpenStack Object Storage - Swift

The OpenStack object storage service Swift offers access to binary objects through an RESTful API. It is therefore very similar to the Amazon S3 object storage. Swift is a scalable storage system that has been in use in production for many years at Rackspace[2] and has become a part of OpenStack. It is highly scalable and capable of managing petabytes of storage. Swift comes with its own replication and distribution scheme and does not rely on special RAID controllers to achieve fault tolerance and resilient storage. It can also be used to host the cloud images for the image service Glance.

A.5.4 Other Storage Solutions

There are other solutions available that can be used to provide storage within OpenStack. These do not have to be pure software solutions, but can be hardware solutions, such as Nexenta, IBM (Storwize family/SVC, XIV), NetApp and SolidFire. These have integrated interfaces that can communicate directly to OpenStack Cinder [145] with varying feature implementations.

[1] www.gluster.org
[2] www.rackspace.com

The software solutions aim at commodity hardware that are based on the principle that hardware fails. They are distributed file systems that have built-in replications. Among them are Lustre [146] [147], Gluster [148], GPFS [149] and MooseFS [150].

Appendix B

Empirical Exploration of Related Work

While studying the related literature, the author undertook an empirical study of the configurations presented by Intel [7], UnitedStack [8] and Han [9] for their performance impacts in the authors testbed.

B.1 Cluster configuration

The basic configuration of the Ceph cluster was kept to the default values at the beginning and changed for the specific runs. The Placement Group count for the storage pool was set to 1024. To reduce the limitation of the backend storage network the replication count was set to 2. The authentication was set to CephX to test the cluster in a realistic OpenStack setup. The OSDs were deployed with the data and the journal (5 GB) partition being on the same disk. Neither memory nor CPU utilizations on the storage cluster hit 100 percent during the tests and were not a limiting factor. The tested Ceph configuration can be seen in Table B.1.

The Ceph `rbd_cache` is behaving like a disk cache but uses the Linux page cache. It is on the client side, which requires it to be deactivated when used with shared block devices as there is no coherency across multiple clients. It can lead to performance improvements but requires memory on the compute hosts. The multiple `debuging` parameter control the logging behaviour of different Ceph components. It can be controlled in different logging levels, but it is an overhead to the storage cluster.

The `journal_max_write_bytes` and `journal_max_write_entries` parameters control the amount of bytes and entries that the journal will write at any one time. By increasing these values allows the system to write a larger block instead of multiple small ones. The `journal_queue_max_bytes` and `journal_queue_max_ops` are working a similar way for the journal queue.

`filestore_max_inline_xattr_size` and `filestore_max_inline_xattr_size_xfs` set the maximimum size of an extended atrributes (XATTR) stored in the file system per object. This parameter can be set individually for the supported file systems (XFS, btrfs, ext4). `filestore_queue_max_bytes` and `filestore_queue_max_ops` set the maximum number of bytes and operations the file store accepts before blocking on queuing new operations. The `filestore_fd_cache_shards` and `filestore_fd_cache_size` control the file descriptor cache size. The shard setting breaks it up into multiple components that do the lookup locally. `filestore_omap_header_cache_size` sets the cache size of the object map. The used cluster contained over 210.000 objects.

B.2 4KB results

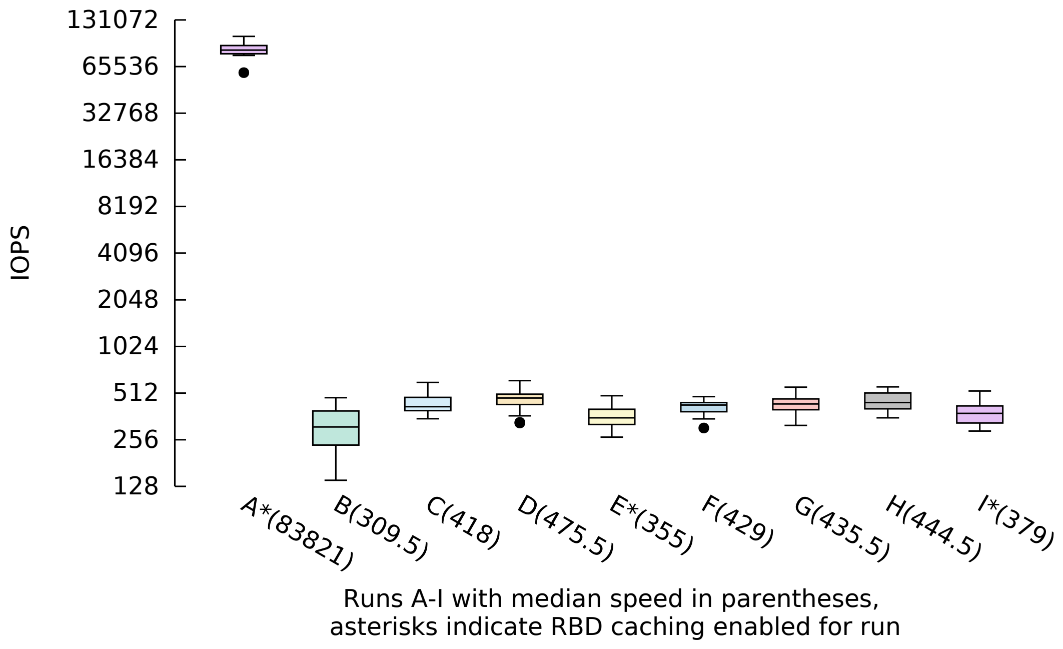


Figure B.1: FIO 4KB random read.

In the 4KB random read test (see Figure B.1) the increased journal with doubled journal operations (D) improves the IOPS from 309.5 to 475.5, which is an improvement of 53.5%. Without the OPS modifier (C) the performance improves by 35%. The other configurations are performing in between these two configurations. The cache had a negative impact of up to 25% (E). The default configuration shows the gains from local caching as the disk subsystem is not capable of delivering over one million IOPS.

In the 4KB random write test (see Figure B.2) the performance could be improved between 3% and 23%. It has to be noted, that this is a maximum improvement from

Parameter	**A (default)**	**B (no debug)**	**C (larger journal)**	**D (C + x2 OPS)**	**E (D + RBD Cache)**	**F (Filestore shards)**	**G (F + XATTR)**	**H (OMAP)**	**I (H + RBD cache)**
rbd_cache	true	false	false	true	false	false	false	false	true
debuging	true	false	false	false	false	false	false	false	false
journal_max_write_bytes	10485760	10485760	1048576000	1048576000	1048576000	1048576000	1048576000	1048576000	1048576000
journal_max_write_entries	100	100	1000	2000	1000	1000	1000	1000	1000
journal_queue_max_bytes	33554432	33554432	1048576000	1048576000	1048576000	1048576000	1048576000	1048576000	1048576000
journal_queue_max_ops	300	300	3000	6000	3000	3000	3000	3000	3000
filestore_max_inline_xattr_size	0	0	0	0	0	0	0	0	0
filestore_max_inline_xattr_size_xfs	65536	65536	65536	65536	65536	65536	65536	0	0
filestore_queue_max_bytes	104857600	104857600	104857600	104857600	104857600	104857600	1048576000	1048576000	1048576000
filestore_queue_max_ops	50	50	50	50	50	50	500	500	500
filestore_fd_cache_shards	16	16	16	16	16	32	32	128	128
filestore_fd_cache_size	128	128	128	128	128	64	64	64	64
filestore_omap_header_cache_size	4096	4096	4096	4096	4096	4096	4096	409600	409600

Table B.1: Ceph parameters used in the benchmarks.

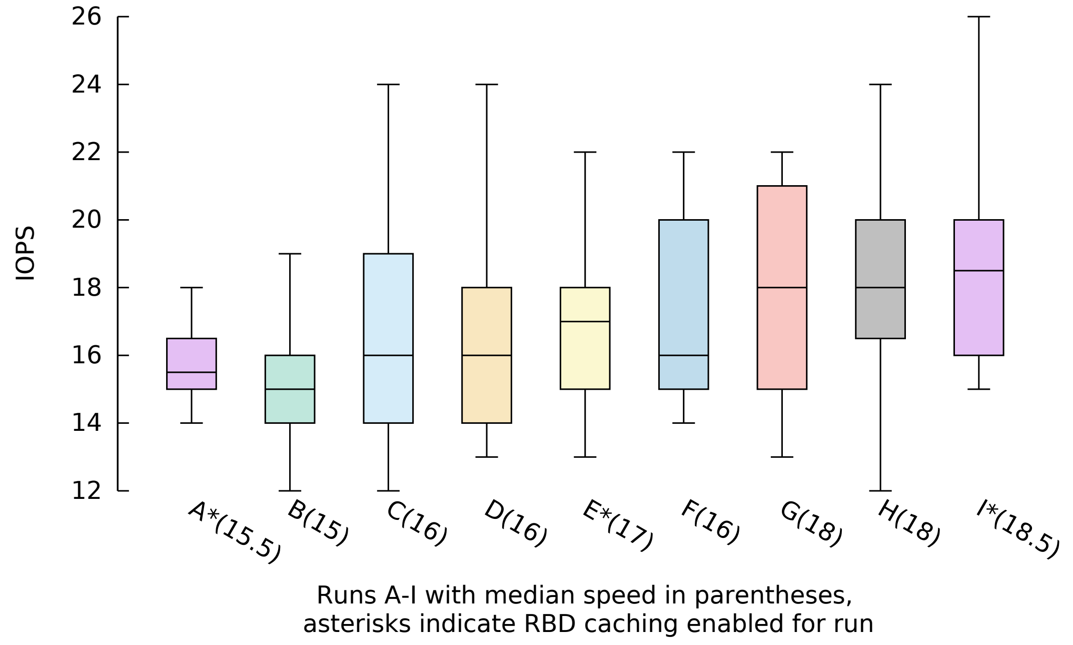


Figure B.2: FIO 4KB random write.

15 to 18.5 IOPS, which is a physical limitation of conventional hard drives. In such a workload solid state drives would be the recommended solution.

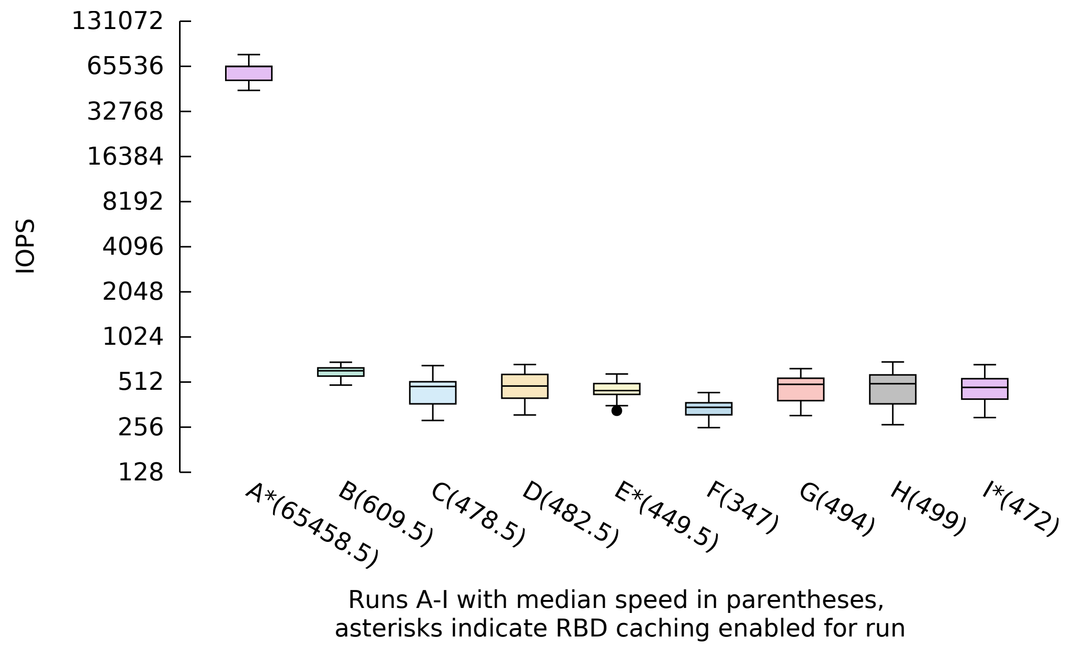


Figure B.3: FIO 4KB sequential read.

In the 4KB sequential read test (see Figure B.3) the performance is degrading when the parameters are applied. The measured penalty is between 19% (H) and 43% (F). Caching reduces the performance in both cases (E and I), whereas the default configuration (A) improved the performance.

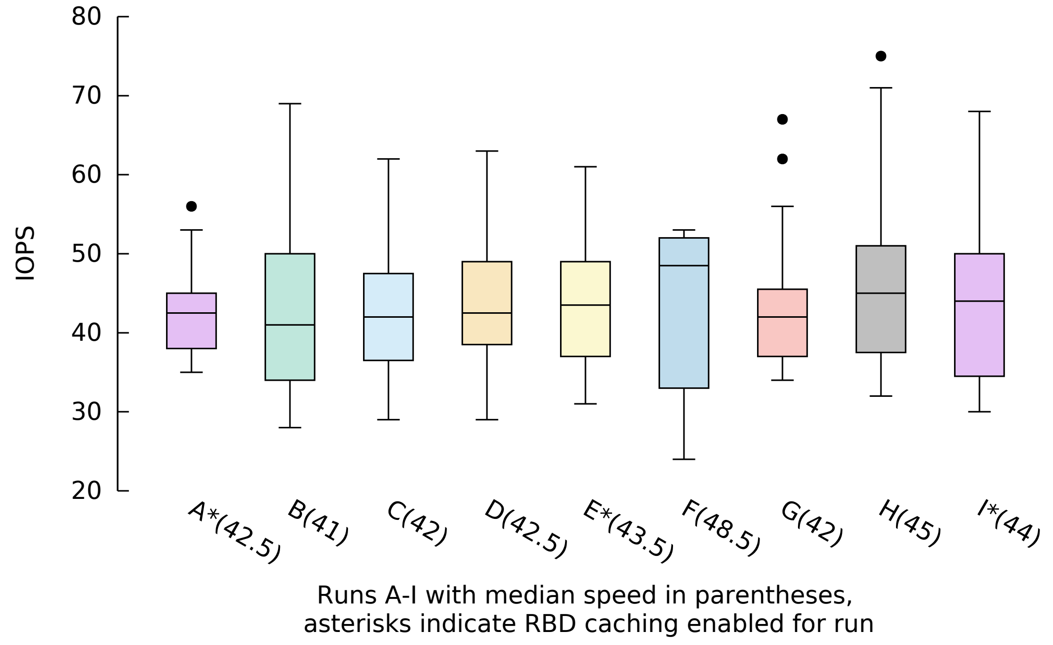


Figure B.4: FIO 4KB sequential write.

In the 4KB sequential write test (see Figure B.4) the shard parameter changes (F) have the largest impact and improve the performance by 18%. The cache has a negligible impact. Like in the random write test (see Figure B.2) the performance is limited by the used hardware.

B.3 128KB results

In the 128KB random read test (see Figure B.5) the throughput improves by 56.5% (F) to 80% (D) in comparison to the baseline. Using the RBD cache results in a performance penalty of 7.5% (I) and 10% (E).

In the 128KB random write test (see Figure B.6) the XATTR configuration (G) improves the performance by 35%. All other tested configurations increase the performance by at least 6.5%. The RBD cache improved the performance slightly and resulted in less widely spread results. The performance of the default configuration (A) is right in the middle of all the tested configurations with an 14.5% increase.

The results for the 128KB sequential read test (see Figure B.7) the performance increases by 18.5% (F) to 41%. Caching is reducing the performance for E and I in comparison to D and H, whereas the default configuration benefits greatly from it.

The results for the 128KB sequential write test (see Figure B.8) the throughput increases by 10-13% for multiple configurations (C, G-I). The configuration with the higher journal OPS (D) is at a similar level to the baseline (B), whereas the added

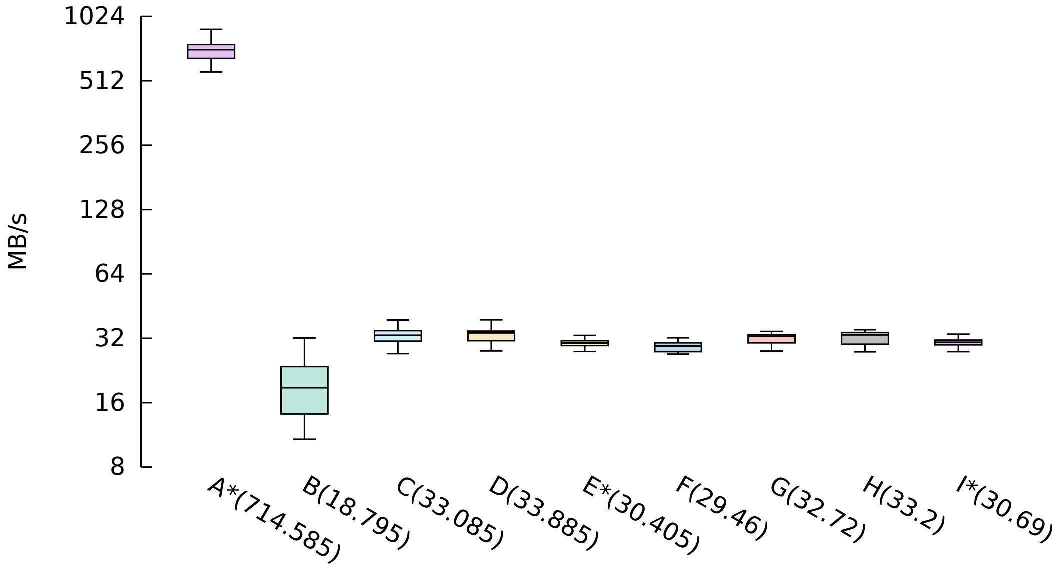


Figure B.5: FIO 128KB random read.

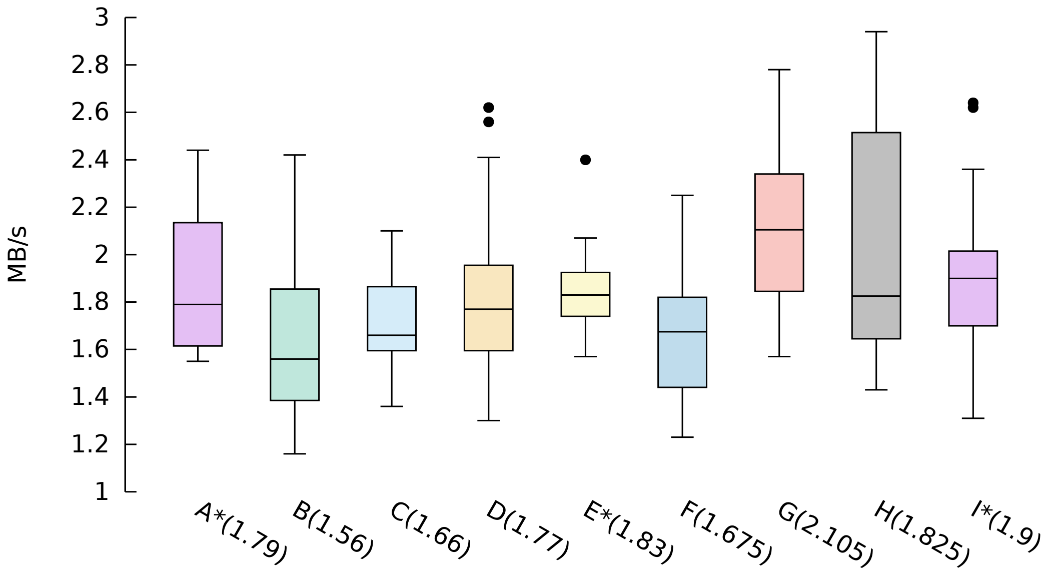


Figure B.6: FIO 128KB random write.

cache improves the gain to 5%. While the shards configuration with the XATTR setting (G) is amongst the top performing, without the XATTR modification (F) it is under performing and resulting in a deficit to the baseline of 2.5%.

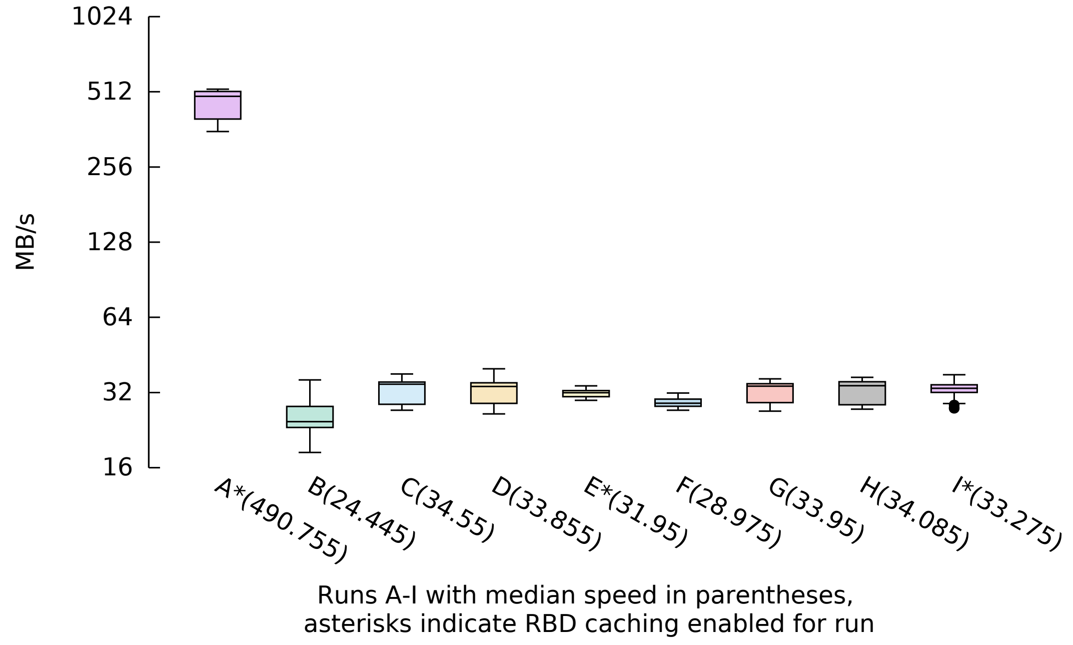


Figure B.7: FIO 128KB sequential read.

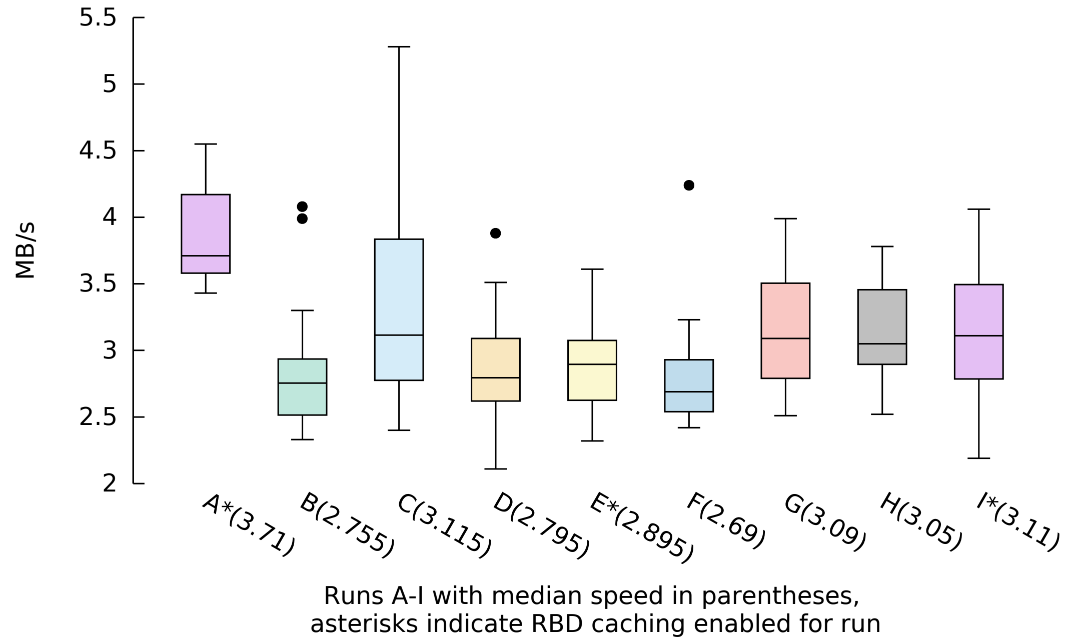


Figure B.8: FIO 128KB sequential write.

B.4 1MB results

The results from the 1MB random read test (see Figure B.9) show a direct improvement when the debugging logging is deactivated. Increasing the journal and its operations from its default value improves performance across all other runs by 5.5% (F) to 26.8% (C). The effect of caching was only present in the baseline, not in the tuned configura-

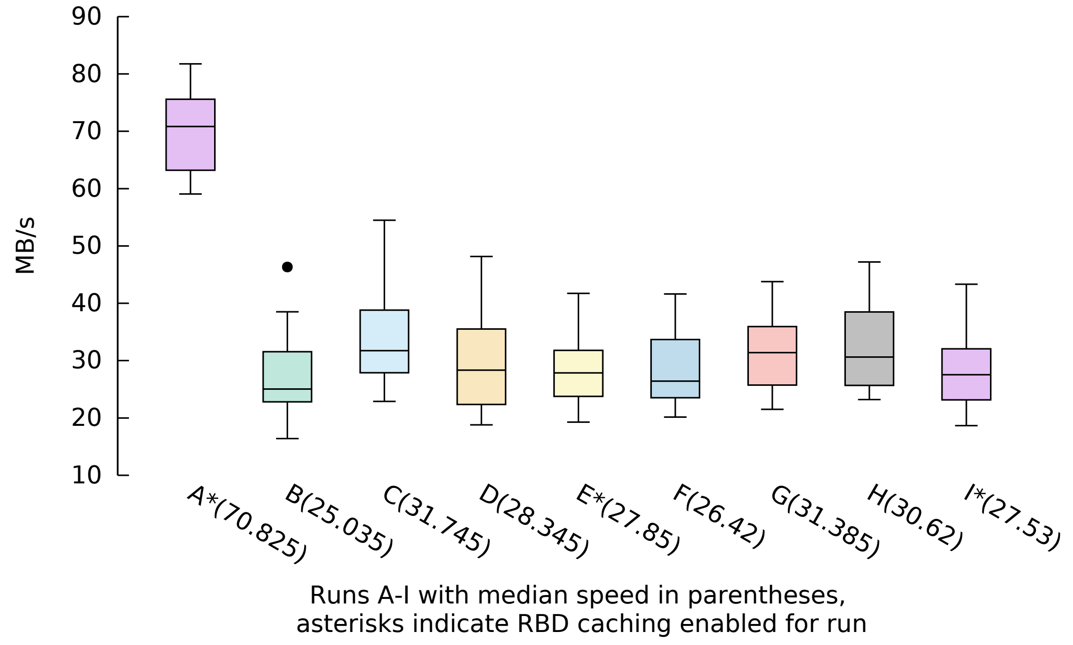


Figure B.9: FIO 1MB random read.

tions where it lowers the performance.

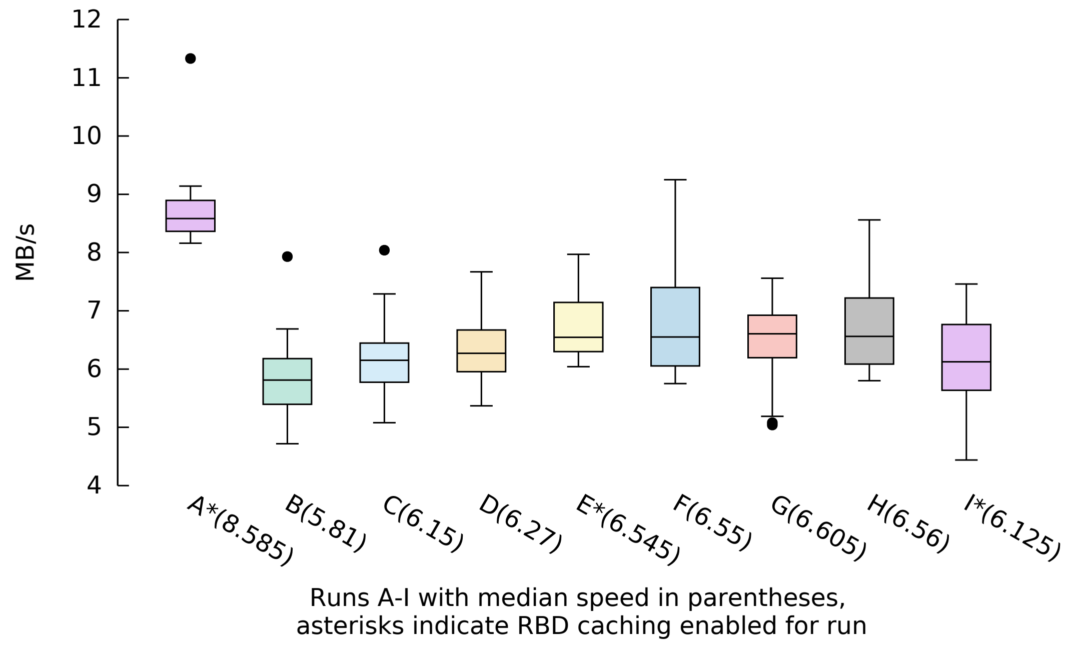


Figure B.10: FIO 1MB random write.

The changes made to the configuration for the 1MB random write test (see Figure B.10) improves the performance in all cases over the baseline results between 5.5% (I) and 13.5% (G). Caching was influencing the results heavily in the default configuration (+47.5%), but only slightly improves the performance for E (+4%) and decreases it for I (-6.5%).

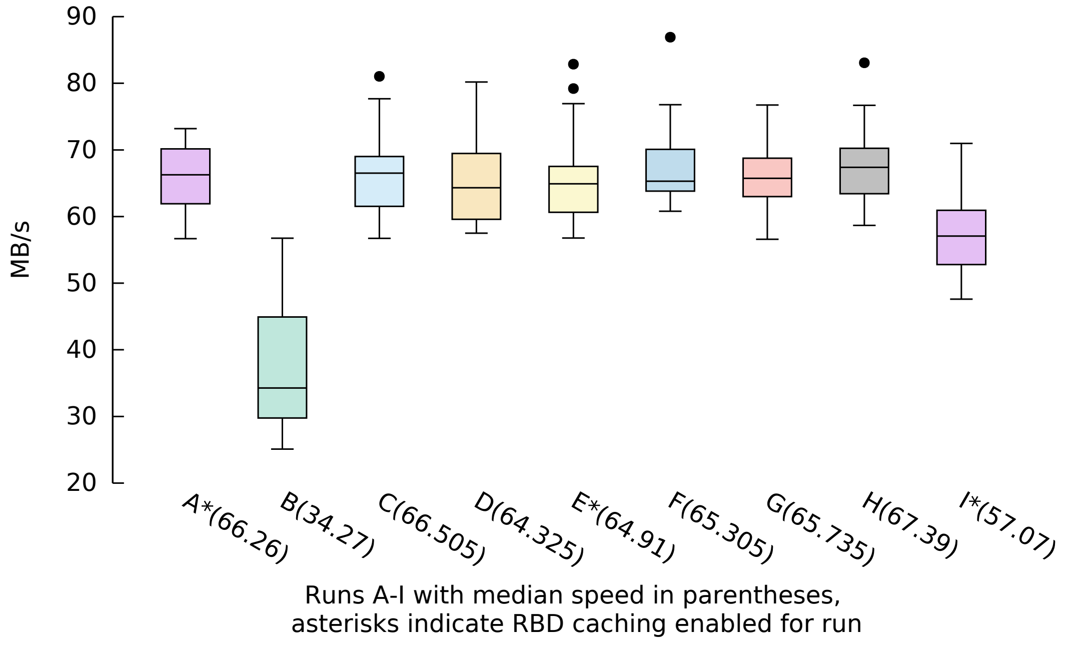


Figure B.11: FIO 1MB sequential read.

The differences in the 1MB sequential read tests (see Figure B.11) between the tuned configurations are only 4.5%. Only the for I the performance was decreased by 11% in comparison to D, which is still 66.5% faster than the baseline configuration, while the same configuration without the caching (H) improved the performance by 96.5%.

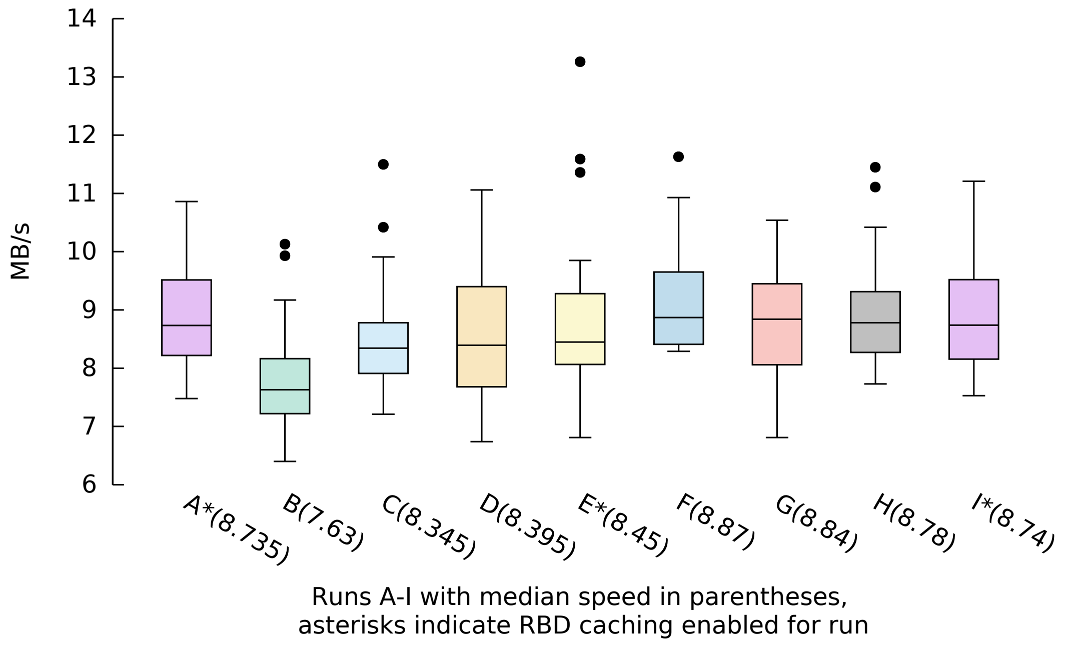


Figure B.12: FIO 1MB sequential write.

The effects of the tuning for the 1MB sequential write tests (see Figure B.12) show that when the shards and the journal were altered (F and G) it lead to an improvement of up to 16%. The caching was not having an impact in this access type.

B.5 32MB results

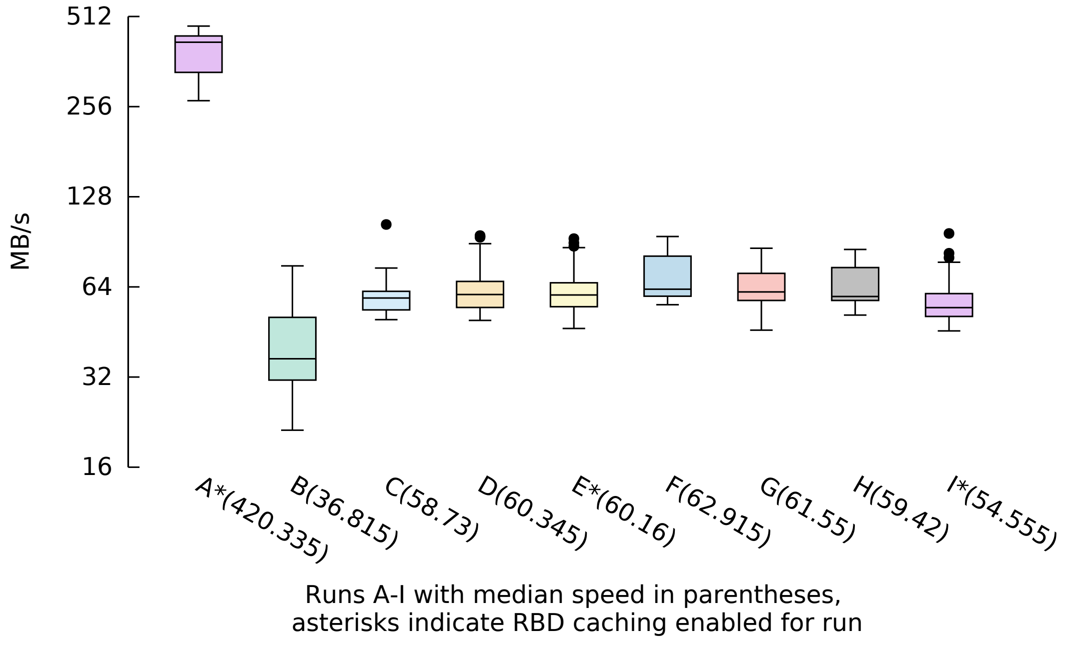


Figure B.13: FIO 32MB random read.

The effects of the tuning for the 32MB random read tests (see Figure B.13) show improvements between 48% (I) and 70.5% (F). The default configurations show the effect of the local caching by exceeding the network bandwidth by a factor of 5.6.

In case of the 32MB random read tests (see Figure B.14) caching improves the performance by 7% (A), not change it at all (I) or decreased it by 11% (E) in comparison to their configuration without the cache. The highest performance could be achieved by configuration F (+39%). The same configuration with XATTR (G) only achieved an increase of 21%.

In the 32MB sequential read benchmark (see Figure B.15) the baseline setting without the cache shows a performance of 35.4 MB/s. All tested deviations from the baseline configuration achieved between 61.5% (I) and 77% (F). The cache lowered the performance for E and I in comparison to D and H by up to 6.5%.

For the 32MB sequential write benchmark (see Figure B.16) an improvement of up to 32% is observed with changed shards settings (F and G). With OMAP cache changes this increase dropped to 27.5%. Caching reduces the performance for E but increased it for A and I by up to 13%.

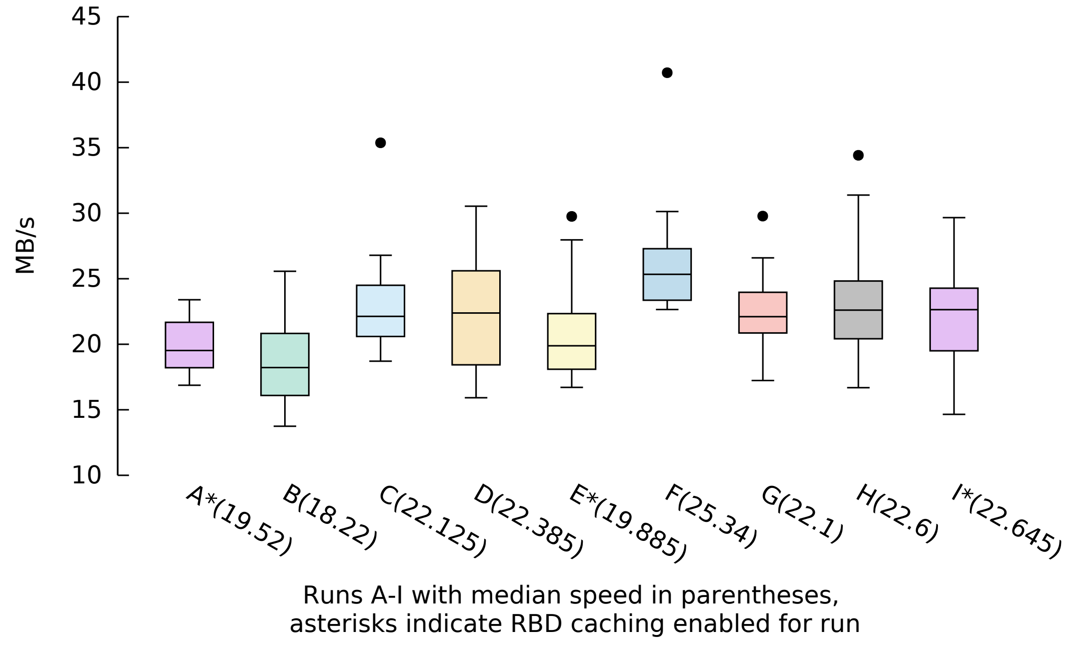


Figure B.14: FIO 32MB random write.

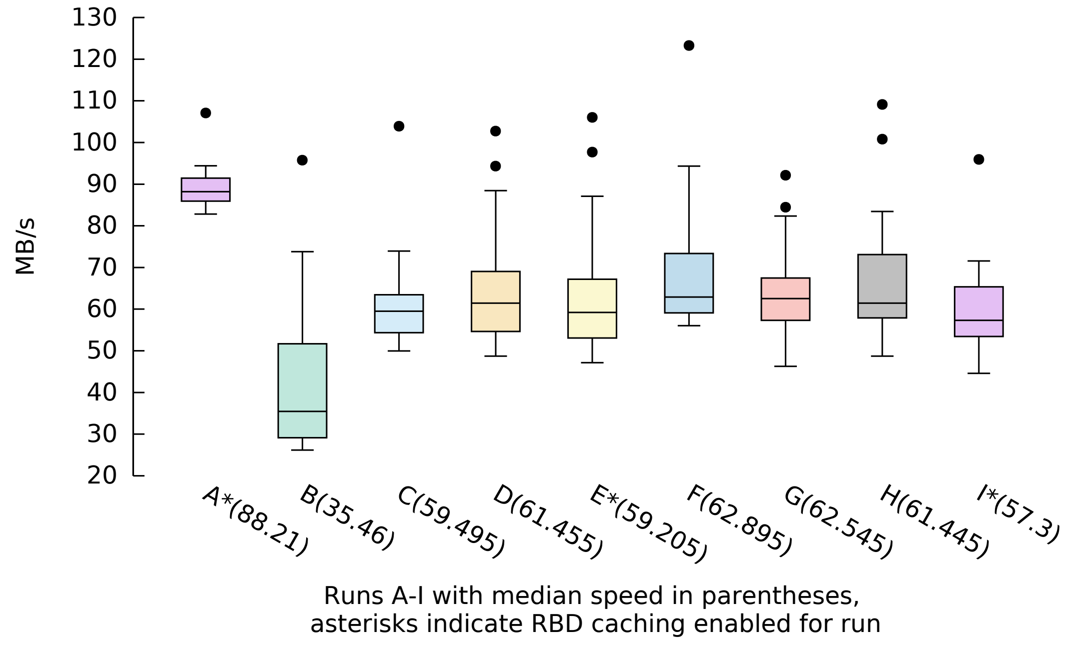


Figure B.15: FIO 32MB sequential read.

B.6 DBENCH

While the results for the DBENCH benchmark for the client count of 1, 6 and 12 are very similar with the baseline and the OMAP configurations coming out on top with similar results to it and the other configurations experiencing a penalty of up to 11%, the results for 48 (see Figure B.17) and 128 clients (see Figure B.18) were much

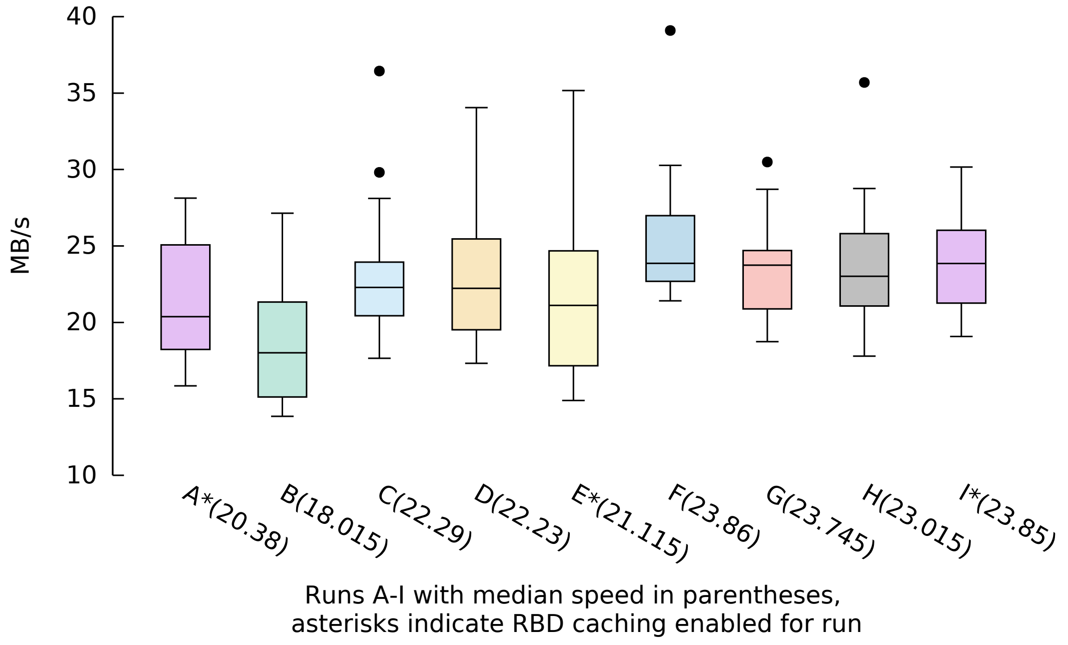


Figure B.16: FIO 32MB sequential write.

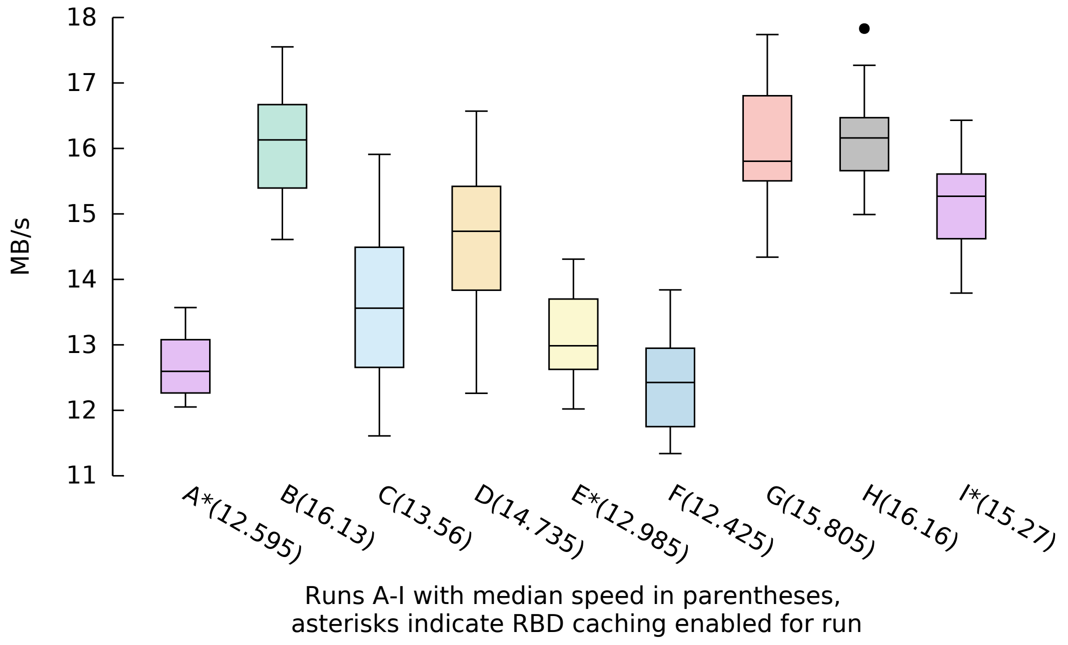


Figure B.17: DBENCH 48 Clients.

more interesting. While the OMAP configurations stay in a similar position in the 48 client run, the default (A), journal (C) and shard configurations (F) loose up to 23% of throughput. In the 128 client scenario the OMAP configuration (H) increases the performance by 14%, while the same configuration with RBD caching (I) has an equal performance to the baseline. The only other configuration able to improve the performance is the XATTR configuration (G) with 5%. The highest loss is present in

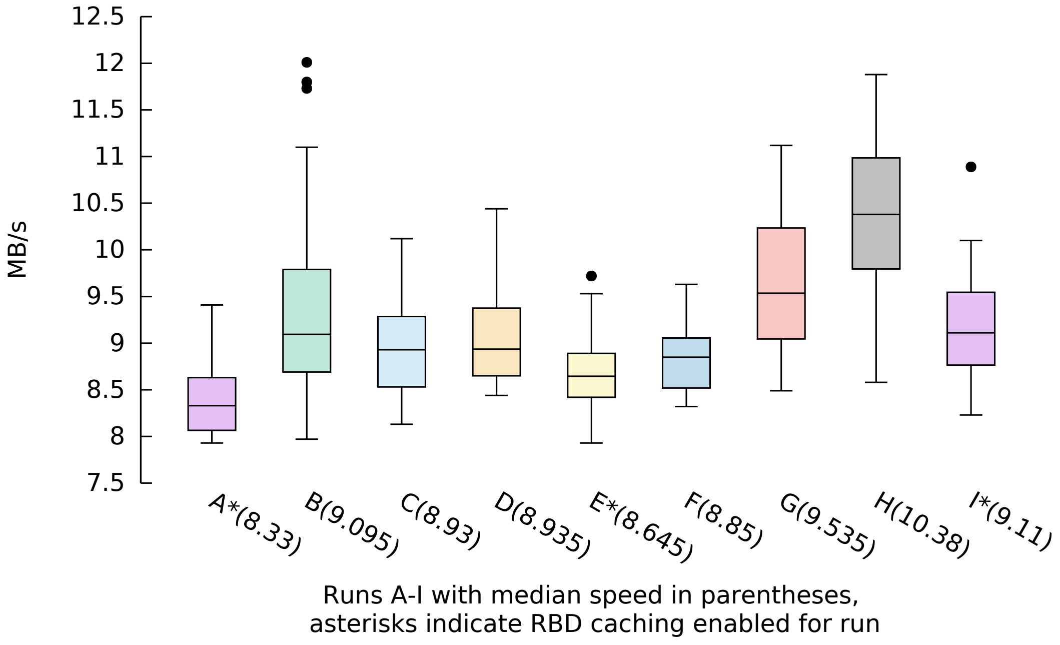


Figure B.18: DBENCH 128 Clients.

the default configuration (A) with -8.5%.

B.7 Conclusion

The results show, that the suggested configurations deliver performance improvements across a limited number of access patterns and sizes only. However, these limitations are not articulated fully in the literature and most likely derive from the fact that synthetic benchmarks are used over a limited range of access sizes and patterns. A more detailed analysis of storage configurations in combination with real workloads is therefore required to determine accurately the performance of the alternative configurations.

Appendix C

Puppet Manifests

Ceph has also been upgraded to the next version. The new version is now Giant (0.87). During the evaluation phase Inktank, the developer of Ceph, got acquired by RedHat. RedHat is offering Ceph now alongside GFS as the storage solution for OpenStack in their own cloud suite.

C.1 Ceph Manifest

The Ceph manifest was not compatible with Foreman, as Foreman can not deploy manifests that use only a `define` statement without a class that calls them, which is working without any problems with Puppet. The benefit of using a define statement is that it can be called multiple times. To use such a behaviour with Foreman it is necessary to create a wrapper class that calls the `define` with a `create_from_resource` construct. When used in such a way the manifests work and can be used. The code excerpt in Listing C.1 is used to create the Ceph OSDs by using a device hash that contains the devices that are intended to become OSDs.

```
#
# Class wrapper to avoid scenario_node_terminus problem
# in Foreman
#
class ceph::osds (
    $device_hash = undef
)
{
    if $device_hash{
        create_resources('ceph::device', $device_hash)
    }
}
```

Listing C.1: Ceph OSD Puppet manifest.

Such a wrapper class is also necessary for the monitor and metadata servers, as both have the same `define` construct as the OSD manifest.

C.2 OpenStack Manifests Initial

The chosen OpenStack manifests from Puppetlabs were compatible with Foreman without any changes, but they had to be tweaked in a couple of places to make them work properly. The changes ranged from small fixes, like changing an Integer to a String in case of the port numbers of the webservices, to larger fixes that involved changes that were generating dependency cycles.

The dependency cycle manifested itself with multiple Puppet versions (3.5 and 3.6). The affected manifest was responsible for setting up RabbitMQ for Nova. The problem originated from double checking for the command line tool `rabbitmqctl`. It was used as a provider and a requirement in the manifest. This construct was used in multiple places. By commenting the `Class[$rabbitmq_class]` statement in all occurrences, as shown in Listing C.2, the dependency cycle got broken and the manifest was running properly.

```
class nova::rabbitmq(
  ...
) {
  # only configure nova after the queue is up
  Class[$rabbitmq_class] -> Anchor<| title == 'nova-start' |>
  if ($enabled) {
    if $userid == 'guest' {
      $delete_guest_user = false
    } else {
      $delete_guest_user = true
      rabbitmq_user { $userid:
        admin     => true,
        password  => $password,
        provider  => 'rabbitmqctl',
        #require   => Class[$rabbitmq_class],
      }
  ...
}
```

Listing C.2: OpenStack Nova RabbitMQ Puppet manifest with commented requirement line.

As the setup requires multiple services to run on the controller, but not all of them, it was necessary to assign to it the `openstack::role::controller` and the `openstack::role::storage` manifests. This was not possible due to duplicate declaration conflict of underlying components. A way to solve this problem is to alter the controller manifest by adding the `glance-api` and `cinder-volume` profiles. The alternative would be to modify the all-in-one manifest and delete the compute and network components. The resulting `openstack::role::controller` file is presented in Listing C.3.

```
class openstack::role::controller inherits ::openstack::role {
  class { '::openstack::profile::firewall': }
  class { '::openstack::profile::rabbitmq': } ->
  class { '::openstack::profile::memcache': } ->
  class { '::openstack::profile::mysql': } ->
  class { '::openstack::profile::mongodb': } ->
  class { '::openstack::profile::keystone': } ->
  class { '::openstack::profile::ceilometer::api': } ->
  class { '::openstack::profile::glance::auth': } ->
  class { '::openstack::profile::cinder::api': } ->
  class { '::openstack::profile::nova::api': } ->
  class { '::openstack::profile::neutron::server': } ->
  class { '::openstack::profile::heat::api': } ->
  class { '::openstack::profile::horizon': }
  class { '::openstack::profile::auth_file': }
  class { '::openstack::profile::glance::api': }
  class { '::openstack::profile::cinder::volume': }
}
```

Listing C.3: OpenStack controller Puppet manifest with the different components required for that node type.

Another big change was the configuration of the Neutron server service. This service is responsible for setting up the networks internal network and the external IPs for the VMs. The service can be configured to run with many different providers, such as linuxbridge, Open VSwitch (OVS) and ML2. In this case the Neutron server was set up to use the OVS plugin. All parameters were set by the manifest for OVS, but the creation of the shared networks failed with a `u'Invalid input for operation: gre networks are not enabled.'` error message. This error originated from a misconfiguration of the Neutron server service. For all other provider options the manifest changed the configuration of the service to use the correct plugin configuration, just not in the case of OVS. The service was always set up to start with the ML2 plugin. The modified version is shown in Listing C.4.

```
class neutron::plugins::ovs (
  ...
) {
  ...
  # ensure neutron server uses correct config file
  if $::osfamily == 'Debian' {
    file_line { '/etc/default/neutron-server:NEUTRON_PLUGIN_CONFIG':
      path    => '/etc/default/neutron-server',
      match   => '^NEUTRON_PLUGIN_CONFIG=(.*)$',
      line     => "NEUTRON_PLUGIN_CONFIG=/etc/neutron/plugins/openvswitch
    /ovs_neutron_plugin.ini",
      require => [
        Package['neutron-plugin-ovs'],
        Package['neutron-server'],
      ],
      notify  => Service['neutron-server'],
    }
  }
}
```

Listing C.4: Modified Neutron OVS puppet manifest.

A change that originated from the use of CEPH as a storage system had to be made to the OpenStack manifest for the Glance API profile. As standard this manifest was using a file backend that had to be changes to rbd (see Listing C.5).

```
class openstack::profile::glance::api {
  ...
  #Use Ceph instead of File storage
  #class { '::glance::backend::file': }
  class { '::glance::backend::rbd':
    rbd_store_pool      => 'images',
    rbd_store_user      => 'glance',
  }
  ...
}
```

Listing C.5: Modified Glance manifest to use the images pool of the Ceph cluster.

A similar change had to be made to the Cinder volume service manifest that uses iSCSI as standard (see Listing C.6).

```
class openstack::profile::cinder::volume {
  ...
  #Use Ceph instead of iSCSI
  #class { '::cinder::volume::iscsi':
    #iscsi_ip_address  => $management_address,
    #volume_group      => 'cinder-volumes',
  #}
  class { '::cinder::volume::rbd':
    rbd_pool    => 'volumes',
    rbd_user    => 'cinder',
    rbd_secret_uuid => 'f4443b20-1e87-4d94-9a54-6f06cff8a07e',
  }
}
```

Listing C.6: Cinder volume manifest with changes to use Ceph pool as the storage device.

Due to CEPH the configuration for libvirt had to be changed as well. It required again the addition of a CEPH specific class to configure libvirt to use a CEPH storage backend instead of a local disk (see Listing C.7).

```
class openstack::profile::nova::compute {
  ...
  class { '::nova::compute::rbd':
    libvirt_rbd_user             = 'cinder',
    libvirt_rbd_secret_uuid      = 'f4443b20-1e87-4d94-9a54-6f06cff8a07e
   ',
    libvirt_images_rbd_pool      = 'volumes',
    libvirt_images_rbd_ceph_conf = '/etc/ceph/ceph.conf',
    rbd_keyring                  = 'client.cinder',
  }
  ...
}
```

Listing C.7: OpenStack Nova manifest for changing libvirt to use Ceph storage backend.

C.3 OpenStack Manifests Final

As the development of OpenStack did not stop during the thesis, OpenStack reached a new version codename Kilo. This came with new features and was the preferred version for the thesis. With the new software release the Puppet manifests changed as well. They changed far beyond changing the version numbers of packages to be installed. The whole structure changed and key components were dropped. The `openstack` class with its profiles was dropped. Instead all the individual components had to be installed manually by assigning the manifests to the individual hosts.

All the puppet manifests are now developed under the OpenStack GitHub repository [151] as is OpenStack itself. The manifests are regularly updated and are widely used. Splitting the manifests up without a super manifest that installs all the dependencies as well leads to a very complex setup on the individual hosts.

The basic manifests that are necessary on all three types of nodes (controller, compute, ceph) are shown in Table C.1. They include the Ceph manifests, the network configuration for the individual and bonded interfaces and the Ubuntu Cloud Archive manifest. The latter gives access to the special cloud repository with updated OpenStack packages, which is directly maintained by Canonical, the company behind Ubuntu.

The Keystone manifests (see Table C.2) are only deployed on the controller. The same is true for the MySQL server and the necessary MariDB repository manifest. The message queuing service RabbitMQ is also only deployed on the controller.

There are 16 manifests in total that have to be assigned to the OpenStack nodes (see Table C.3). The selection of manifests differs between the nodes. While the

Table C.1: Basic Puppet manifests assigned to the individual machines types.

Manifest Class	**Controller**	**Compute**	**Ceph**
ceph::profile::client	✓	✓	✓
ceph::profile::params	✓	✓	✓
network	✓	✓	✓
uca_repo	✓	✓	X

Table C.2: Keystone Puppet manifests are only assigned to the controller node.

Manifest Class	**Controller**	**Compute**	**Ceph**
keystone	✓	X	X
keystone::client	✓	X	X
keystone::db::mysql	✓	X	X
keystone::endpoint	✓	X	X
keystone::roles::admin	✓	X	X
mariadbrepo	✓	X	X
mysql::server	✓	X	X
rabbitmq	✓	X	X

Table C.3: Nova Puppet manifests are only assigned to the OpenStack nodes.

Manifest Class	**Controller**	**Compute**	**Ceph**
nova	✓	✓	X
nova::api	✓	X	X
nova::cert	✓	X	X
nova::client	✓	✓	X
nova::compute	X	✓	X
nova::compute::libvirt	X	✓	X
nova::compute::neutron	X	✓	X
nova::compute::rbd	X	✓	X
nova::conductor	✓	X	X
nova::consoleauth	✓	X	X
nova::cron::archive_deleted_rows	✓	X	X
nova::db::mysql	✓	X	X
nova::keystone::auth	✓	X	X
nova::network::neutron	✓	✓	X
nova::scheduler	✓	X	X
nova::vncproxy	✓	X	X

`nova::compute` manifests, that configure the hypervisor for VNC access, rados block devices and networking, are only deployed on the compute node, most of the other manifests are unique to the controller node. The only manifests that are assigned to both node types are the `nova`, `nova::client` and `nova::network::neutron` manifests.

Out of the 14 Neutron manifests only two are deployed to both OpenStack node types (see Table C.4). While the `neutron` class is the main class that does the general configuration, the `neutron::agents::ml2::ovs` class configures the Neutron network for the tunnels between the compute node and the controller/network node. The manifests

Table C.4: Neutron Puppet manifests are mostly deployed to the controller node.

Manifest Class	**Controller**	**Compute**	**Ceph**
neutron	✓	✓	X
neutron::agents::dhcp	✓	X	X
neutron::agents::l3	✓	X	X
neutron::agents::lbaas	✓	X	X
neutron::agents::metadata	✓	X	X
neutron::agents::metering	✓	X	X
neutron::agents::ml2::ovs	✓	✓	X
neutron::client	✓	X	X
neutron::db::mysql	✓	X	X
neutron::keystone::auth	✓	X	X
neutron::plugins::ml2	✓	X	X
neutron::quota	✓	X	X
neutron::server	✓	X	X
neutron::server::notifications	✓	X	X

Table C.5: Glance image service manifests are only deployed to the controller node.

Manifest Class	**Controller**	**Compute**	**Ceph**
glance	✓	X	X
glance::api	✓	X	X
glance::backend::rbd	✓	X	X
glance::client	✓	X	X
glance::db::mysql	✓	X	X
glance::keystone::auth	✓	X	X
glance::notify::rabbitmq	✓	X	X
glance::registry	✓	X	X

only assigned to the controller set up the Neutron server and the different agents, such as the metadata agent that is required by the cloud-init scripts described in Section 3.2.1.

The manifests for the image service Glance are only deployed on the controller node (see Table C.5). In a system with a dedicated Glance node these would be spread between it and the controller.

The Puppet manifests for the block storage Cinder are only deployed on the controller node (see Table C.6), including the `cinder::volume` classes. They configure the individual storage pools within Cinder. As this deployment uses Ceph as a backend for Cinder it requires the `cinder::volume::rbd` to use RADOS block devices.

Table C.6: Cinder manifests with connection to the Ceph storage cluster.

Manifest Class	**Controller**	**Compute**	**Ceph**
cinder	✓	X	X
cinder::api	✓	X	X
cinder::backends	✓	X	X
cinder::client	✓	X	X
cinder::db::mysql	✓	X	X
cinder::keystone::auth	✓	X	X
cinder::quota	✓	X	X
cinder::scheduler	✓	X	X
cinder::volume	✓	X	X
cinder::volume::rbd	✓	X	X

Appendix D

Other Tracing Tools

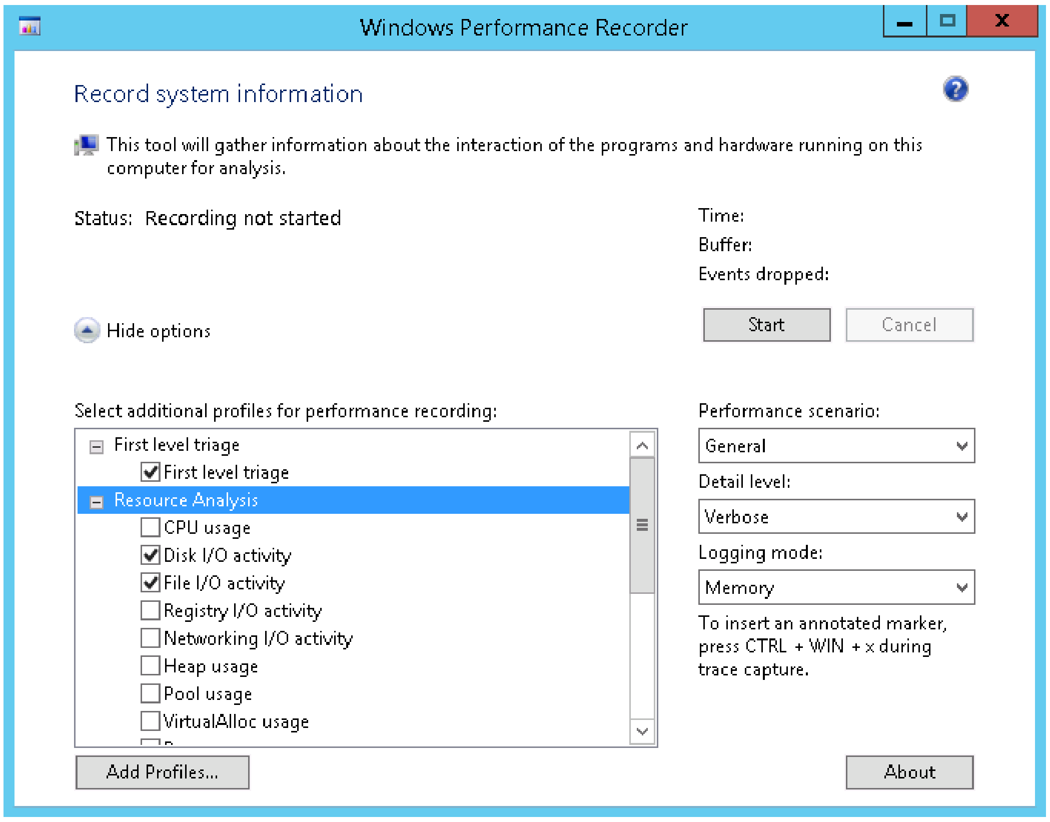
D.1 Microsoft Windows

Microsoft releases a free collection of performance analysis tools for its desktop operating systems starting from Windows 7 and for its server operating systems starting from Windows Server 2008 R2. The Windows Performance Toolkit is part of the Windows Assessment and Deployment Kit, which is available on the Microsoft webpage [152]. The toolkit contains two tools that are used for trace capturing and analysis. The Windows Performance Recorder (WPR) and the Windows Performance Analyzer (WPA). The combination of these two provides a very powerful analysis platform that goes far beyond the capabilities of the Windows Task Manager and the Performance Monitor. WPR and WPA are usually used after a problem has been identified and narrowed down by using the Performance Monitor.

D.1.1 Windows Performance Recorder

The Windows Performance Recorder is used to create the trace of the application itself (see Figure D.1). The user can select different components to be traced, such as CPU, GPU, disk, file registry and network I/O and other hardware or software components. Furthermore, it comes with a predefined set of traces for specific problems or scenarios, such as audio and video glitches, Internet Explorer or Edge browser issues.

The WPR can be used to do a general trace of a specific application, but it also can be used to record the boot, shutdown, reboot, standby and hibernation (including resume) phases. This makes the WPR a versatile tool to cover a broad range of scenarios. The recording can be done in verbose or light mode, where verbose is an in-depth recording and light records only capture the timing. WPR comes with two logging modes: memory and file based. For the memory based logging the recording is saved in a circular buffer in memory. This means, the logging length is limited by the memory


Windows Performance Recorder

Record system information

This tool will gather information about the interaction of the programs and hardware running on this computer for analysis.

Status: Recording not started

Time:
Buffer:
Events dropped:

Select additional profiles for performance recording:

- First level triage
  - [x] First level triage
- Resource Analysis
  - [ ] CPU usage
  - [x] Disk I/O activity
  - [x] File I/O activity
  - [ ] Registry I/O activity
  - [ ] Networking I/O activity
  - [ ] Heap usage
  - [ ] Pool usage
  - [ ] VirtualAlloc usage

Performance scenario: General
Detail level: Verbose
Logging mode: Memory

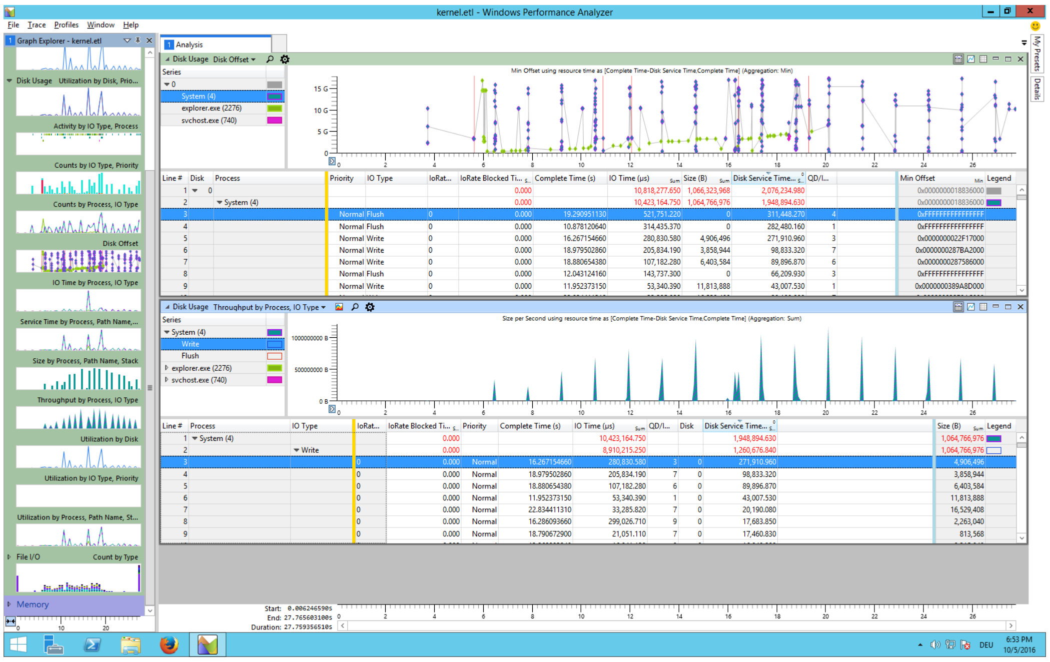
To insert an annotated marker, press CTRL + WIN + x during trace capture.

Figure D.1: Windows Performance Recorder with multiple trace options.

capacity. When the memory capacity reaches its limit, the recording will be overwritten, so logging information will be lost. The default for this mode is a buffer size of 128 KB and a buffer count of 64. This can be set manually to a fixed size or to a percentage of the main memory of the host. For a file based logging, the information is written sequentially to a file on disk. This allows, assuming a sufficient amount of storage capacity being available, for much longer recordings [153] [154] [155].

D.1.2 Windows Performance Analyzer

The Windows Performance Analyzer (WPA) is used to analyse the captured data. It can visualize the trace data based on the traced components (storage, compute, network, memory) and their individual subcategories. Figure D.2 shows a storage trace of a file transfer using the Windows File Explorer. On the left side of the WPA interface are the different components and various individual traces, such as "IO Time by Process" or "Utilization by IO Type". The main part of the window is used to present detailed trace information. The data can be displayed graphically, in text form or in a combined form (as shown in the figure). The data is sorted by process and presented in a cascaded view to improve readability. Individual processes can be hidden from the graphs, only showing those processes of interest. This fine detailed view can be used to identify an application characteristic of interest or to see the load that an application puts on individual components. Furthermore, loads can be analysed to reveal individual calls, access sizes and occurrences [153] [154] [155].


kernel.etl - Windows Performance Analyzer
File Trace Profiles Window Help
Graph Explorer - kernel.etl
Analysis
Disk Usage Disk Offset
Min Offset using resource time as [Complete Time-Disk Service Time,Complete Time] (Aggregation: Min)
Series
0
System (4)
explorer.exe (2276)
svchost.exe (740)
Disk Usage Throughput by Process, IO Type
Size per Second using resource time as [Complete Time-Disk Service Time,Complete Time] (Aggregation: Sum)
Write
Flush
Memory
File I/O

Line #	Disk	Process	Priority	IO Type	IoRat...	IoRate Blocked Ti...	Complete Time (s)	IO Time (µs) Sum	Size (B) Sum	Disk Service Time...	QD/I...	Min Offset	Legend
1	0					0.000		10,818,277.650	1,066,323,968	2,076,234.980		0x0000000018836000	▨
2		System (4)				0.000		10,423,164.750	1,064,766,976	1,948,894.630		0x0000000018836000	▨
3			Normal	Flush	0	0.000	19.290951130	521,751.220	0	311,448.270	4	0xFFFFFFFFFFFFFFFF	
4			Normal	Flush	0	0.000	10.878120640	314,435.370	0	282,480.160	1	0xFFFFFFFFFFFFFFFF	
5			Normal	Write	0	0.000	16.267154660	280,830.580	4,906,496	271,910.960	3	0x0000000022F17000	
6			Normal	Write	0	0.000	18.979502860	205,834.190	3,858,944	98,833.320	7	0x0000000287BA2000	
7			Normal	Write	0	0.000	18.880654380	107,182.280	6,403,584	89,896.870	6	0x0000000287586000	
8			Normal	Flush	0	0.000	12.043124160	143,737.300	0	66,209.930	3	0xFFFFFFFFFFFFFFFF	
9			Normal	Write	0	0.000	11.952373150	53,340.390	11,813,888	43,007.530	1	0x0000000389A8D000	

Line #	Process	IO Type	IoRat...	IoRate Blocked Ti...	Priority	Complete Time (s)	IO Time (µs) Sum	QD/I...	Disk	Disk Service Time...	Size (B) Sum	Legend
1	System (4)			0.000			10,423,164.750			1,948,894.630	1,064,766,976	▨
2		Write		0.000			8,910,215.250			1,260,676.840	1,064,766,976	☐
3			0	0.000	Normal	16.267154660	280,830.580	3	0	271,910.960	4,906,496	
4			0	0.000	Normal	18.979502860	205,834.190	7	0	98,833.320	3,858,944	
5			0	0.000	Normal	18.880654380	107,182.280	6	0	89,896.870	6,403,584	
6			0	0.000	Normal	11.952373150	53,340.390	1	0	43,007.530	11,813,888	
7			0	0.000	Normal	22.834411310	33,285.820	7	0	20,190.080	16,529,408	
8			0	0.000	Normal	16.286093660	299,026.710	9	0	17,683.850	2,263,040	
9			0	0.000	Normal	18.790672900	21,051.110	7	0	17,460.830	813,568	

```
Start:    0.006246590s
End:     27.765603100s
Duration: 27.759356510s
```

Figure D.2: Trace of a file copy (1.1 GB) from network share to local the drive using the Windows File Explorer.

D.2 IBM System Z

The IBM System/Z mainframe series has a tracing functionality built directly into the hardware and the Multiple Virtual Storage (MVS) or z/OS operating system. This tracing functionality can be used to capture six different types of traces to identify problems and performance issues [156] [157] [158]:

System Trace provides an ongoing record of hardware and software events occurring during system initialization and system operation. This trace form is activated automatically by the system during initialization, unless otherwise configured.

Master Trace is a collection of all recent system messages. While the other trace types capture internal events, the master trace logs external system activities. By default it is started at system initialization.

Component Trace provides a way for z/OS components to collect problem data about events that occur within these components. Each component is required to configure its trace to provide unique data when using the component trace. Component traces are commonly used by IBM support to diagnose problems in a component, but are also used by users to recreate a problem to gather more data.

Transaction Trace allows debugging problems of work units that run on a single system or across systems in a sysplex environment (single system image cluster for IBM z/OS). Transaction trace provides a consolidated trace of key events for the execution path of application or transaction type work units running in a multi-system application environment. It is mainly used to aggregate data to show the flow of work between components in a sysplex to serve a transaction.

Generalized Trace Facility (GTF) is similar to the system trace that traces system and hardware events, but offers the functionality of external writers, which can write user defined trace events.

GFS trace is a tool that collects information about the use of the `GETMAIN`, `FREEMAIN`, or `STORAGE` macros. It can be used to analyse the allocation of virtual storage and identify users of large amounts of storage. GFS requires GTF trace data as an input.

Due to their in-depth trace characteristics, the System Z platform allows developers to identify the problems of their code, where and when bottlenecks happen in general, or problems with specific components, such as devices or functions.

D.3 Low Level OS Tools

In some cases, I/O trace are required directly from a Linux/Unix system rather than from a virtualized host. There are a couple of tools available that can be used in such environments. In this section, `ioprof` and `strace` are introduced, which are able to capture traces at different levels and detail within an operating system. The tools are installed on the system that is to be traced. These tools have an impact on performance, as shown by Juve *et al.* [159].

D.3.1 ioprof

The Linux I/O Profiler (ioprof) [160] [161] is an open source tool developed by Intel which provides insight into I/O workloads on Linux hosts. It uses `blktrace` [162] and `blkparse` to trace I/Os and to analyse them. It presents results for easy consumption [163].

The tool is able to capture block devices as a whole (such as a specific hard drive) or on a partition (such as `/home`). The dependencies of `ioprof` are very small, it only depends on the packages `perl`, `perl-core`, `fdisk`, `blktrace` and `blkparse`. If a PDF output report is desired, `gnuplot` has to be installed. For downloading, a Git client also has to be available. When all these dependencies are met, the tool can be downloaded with the command shown in Listing D.1.

```
# git clone https://github.com/01org/ioprof.git
```

Listing D.1: Download command for ioprof from the GitHub repository.

`ioprof` offers three different operation modes that are selectable by the arguments passed to the program:

- The live mode sends the information directly to `stdout`.
- The trace mode allows capturing the data without processing.
- The post-process mode is used to analyse a previously recorded trace.

When the program is called without any arguments, it reports all available options as shown in Listing D.2.

```
# ./ioprof/ioprof.pl
./ioprof/ioprof.pl (2.0.4)
Invalid command

Usage:
./ioprof/ioprof.pl -m trace -d <dev> -r <runtime> [-v] [-f]   # run
    trace for post-processing later
./ioprof/ioprof.pl -m post  -t <dev.tar file>     [-v] [-p]   # post-
    process mode
./ioprof/ioprof.pl -m live  -d <dev> -r <runtime> [-v]        # live
    mode

Command Line Arguments:
-d <dev>            : The device to trace (e.g. /dev/sdb).  You can run
    traces to multiple devices (e.g. /dev/sda and /dev/sdb)
at the same time, but please only run 1 trace to a single device (e.g. /
    dev/sdb) at a time
-r <runtime>        : Runtime (seconds) for tracing
-t <dev.tar file>   : A .tar file is created during the 'trace' phase.
    Please use this file for the 'post' phase
You can offload this file and run the 'post' phase on another system.
-v                  : (OPTIONAL) Print verbose messages.
-f                  : (OPTIONAL) Map all files on the device specified
    by -d <dev> during 'trace' phase to their LBA ranges.
This is useful for determining the most fequently accessed files, but
    may take a while on really large filesystems
-p                  : (OPTIONAL) Generate a .pdf output file in addition
     to STDOUT.  This requires 'pdflatex', 'gnuplot' and 'terminal png'
    to be installed.
```

Listing D.2: Options offered by ioprof.

The commands for starting a live trace, a trace without processing and the post-processing of a previous trace are shown in Listing D.3.

```
# ./ioprof.pl -m live -d /dev/sdb1
# ./ioprof.pl -m trace -d /dev/sdb1 -r 660
# ./ioprof.pl -m post -t sdb1.tar
```

Listing D.3: Example commands to start different forms of traces and post-processing on a saved tracefile.

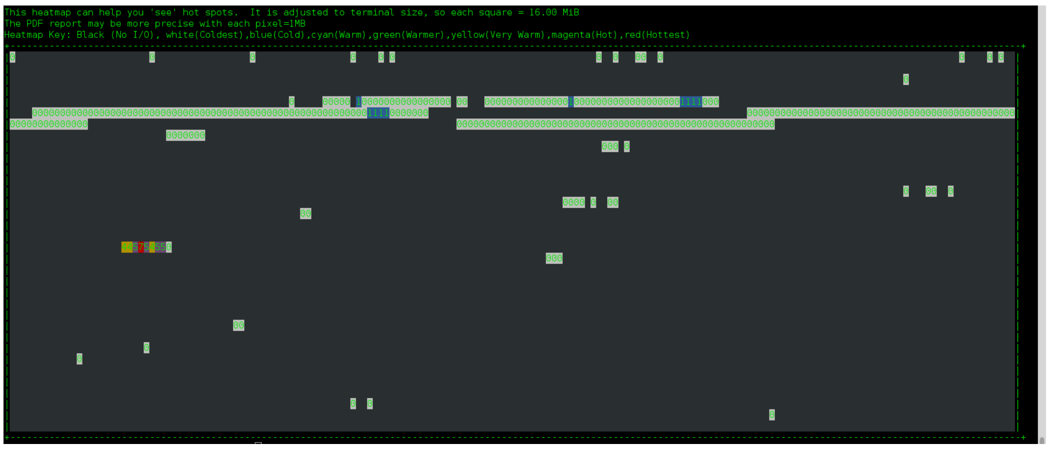
Running `ioprof` in the post-process mode reads in all `blktrace` files and creates a statistical analysis of the trace and a console ASCII heatmap. Two of these heatmap examples are shown in Figure D.3 and Figure D.4. The statistics reported on the command line are shown in Listing D.4.

```
# ./ioprof.pl -m post -t sda1.tar
./ioprof.pl (2.0.4)
Unpacking sda1.tar.  This may take a minute.
lbas: 201326592 sec_size: 512 total: 96.00 GiB
Time to parse.  Please wait...
Finished parsing files.  Now to analyze
Done correlating files to buckets.  Now time to count bucket hits.

--------------------------------------------
Histogram IOPS:
1.9 GB 17.9% (17.9% cumulative)
3.8 GB 7.4% (25.4% cumulative)
5.8 GB 6.4% (31.7% cumulative)
7.7 GB 6.3% (38.1% cumulative)
9.6 GB 5.6% (43.7% cumulative)
11.5 GB 4.7% (48.3% cumulative)
13.4 GB 4.2% (52.6% cumulative)
15.4 GB 4.0% (56.6% cumulative)
17.3 GB 3.6% (60.2% cumulative)
19.2 GB 3.5% (63.8% cumulative)
21.1 GB 3.5% (67.3% cumulative)
23.1 GB 3.5% (70.8% cumulative)
25.0 GB 3.5% (74.3% cumulative)
26.9 GB 3.3% (77.6% cumulative)
28.8 GB 2.8% (80.4% cumulative)
30.7 GB 2.5% (82.9% cumulative)
32.7 GB 2.1% (85.0% cumulative)
34.6 GB 1.6% (86.7% cumulative)
36.5 GB 1.4% (88.1% cumulative)
38.4 GB 1.4% (89.5% cumulative)
40.3 GB 1.3% (90.8% cumulative)
42.3 GB 1.0% (91.8% cumulative)
44.2 GB 0.8% (92.6% cumulative)
46.1 GB 0.7% (93.3% cumulative)
48.0 GB 0.7% (94.0% cumulative)
49.9 GB 0.7% (94.7% cumulative)
51.9 GB 0.7% (95.4% cumulative)
```

```
53.8 GB 0.7% (96.1% cumulative)
55.7 GB 0.7% (96.8% cumulative)
57.6 GB 0.7% (97.6% cumulative)
59.5 GB 0.7% (98.3% cumulative)
61.5 GB 0.7% (99.0% cumulative)
63.4 GB 0.7% (99.7% cumulative)
65.1 GB 0.3% (100.0% cumulative)
--------------------------------------------
Approximate Zipfian Theta Range: 0.0014-1.3591 (est. 0.5635).
Stats IOPS:
"4K READ" 0.92% (5164 IO's)
"16K READ" 1.12% (6273 IO's)
"32K READ" 0.84% (4696 IO's)
"64K READ" 0.84% (4721 IO's)
"128K READ" 56.72% (317020 IO's)
"4K WRITE" 9.36% (52332 IO's)
"512K WRITE" 26.42% (147650 IO's)
Stats BW:
"4K READ" 0.02% (0.02 GiB)
"16K READ" 0.08% (0.10 GiB)
"32K READ" 0.12% (0.14 GiB)
"64K READ" 0.25% (0.29 GiB)
"128K READ" 33.60% (38.70 GiB)
"4K WRITE" 0.17% (0.20 GiB)
"512K WRITE" 62.59% (72.09 GiB)
```

Listing D.4: Output of the post-processing of a recorded trace.

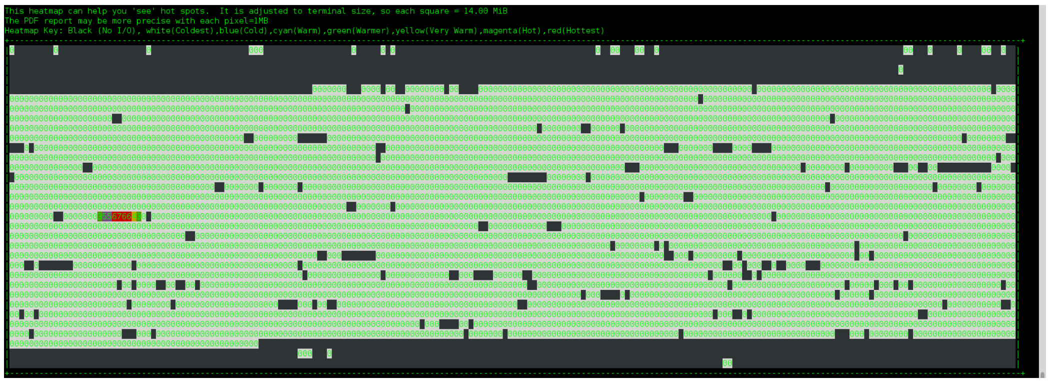


```
This heatmap can help you 'see' hot spots.  It is adjusted to terminal size, so each square = 16.00 MiB
The PDF report may be more precise with each pixel=1MB
Heatmap Key: Black (No I/O), white(Coldest),blue(Cold),cyan(Warm),green(Warmer),yellow(Very Warm),magenta(Hot),red(Hottest)
```

Figure D.3: ioprof console ASCII heatmap (Black (No I/O), white(Coldest), blue(Cold), cyan(Warm), green(Warmer), yellow(Very Warm), magenta(Hot), red(Hottest)) of blogbench read and write workload run.

When the post-process mode is called with the `-p` argument, `ioprof` creates a full PDF report. The report is structured and contains multiple sections:

- A summary of the workload containing the read/write distribution.
- If ioprof is called with the `-f` argument, it will report details of the most accessed



```
This heatmap can help you 'see' hot spots.  It is adjusted to terminal size, so each square = 14.00 MiB
The PDF report may be more precise with each pixel=1MB
Heatmap Key: Black (No I/O), white(Coldest),blue(Cold),cyan(Warm),green(Warmer),yellow(Very Warm),magenta(Hot),red(Hottest)
```

Figure D.4: ioprof console ASCII heatmap (Black (No I/O), white(Coldest), blue(Cold), cyan(Warm), green(Warmer), yellow(Very Warm), magenta(Hot), red(Hottest)) of Postmark benchmark run (min size: 1024B; max size: 16MB; 3000 files; 10000 iterations).

files, which can be used to identify heavily accessed files. Using this mode will increase tracing duration, as the file placement has to be determined. Without the `-f` argument this section will stay empty.

- An IOPS histogram (see Figure D.5) depicting the access regions.
- An IOPS heatmap, similar to the console version (see Figure D.6).
- Statistics about the I/O size of the trace. The report shows the I/O distribution according to their access size and type in a barchart (see Figure D.7).
- A section for Bandwidth statistics (not yet implemented).
- A Caveat Emptor section with description of the procedure.

D.3.2 strace

For an even deeper analysis of I/O operations, the tool `strace` [164] can be used. It traces system calls of a process and the signals received by a process. This can be a viable option to identify a system call (service request between a program and the operating system kernel) that creates a high I/O load. For getting a general understanding of the I/O characteristics of a workload, this tool is too detailed, as all calls of a process are recorded in high detail and accuracy.

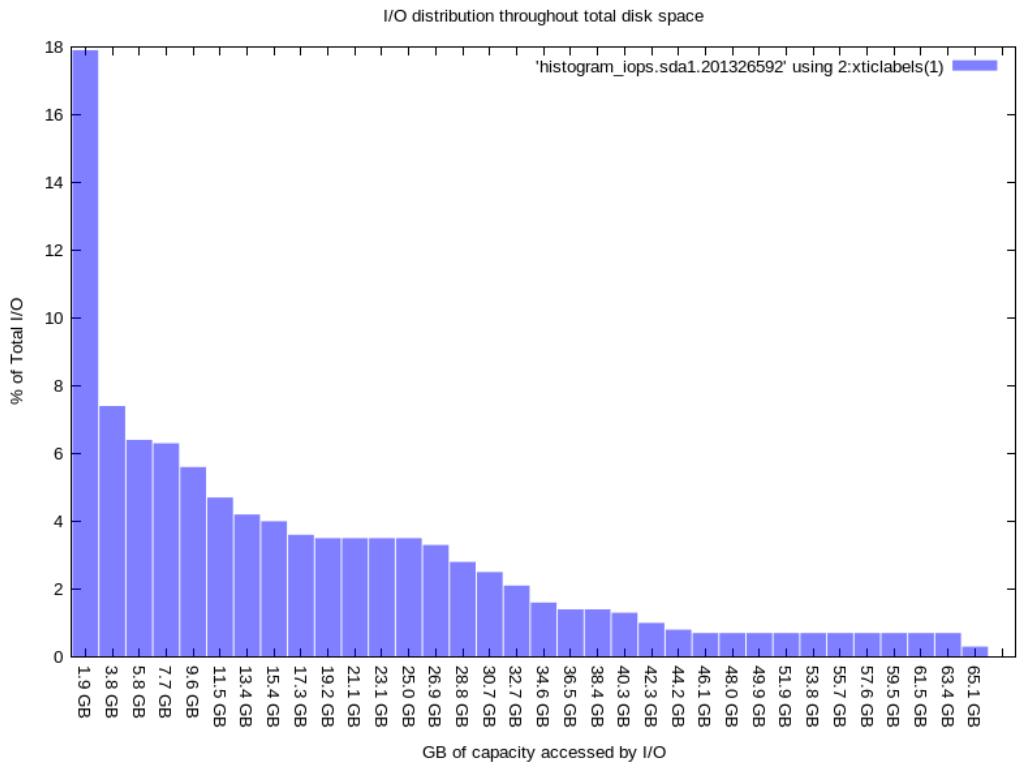


Figure D.5: ioprof IOPS histogram from pdf report.

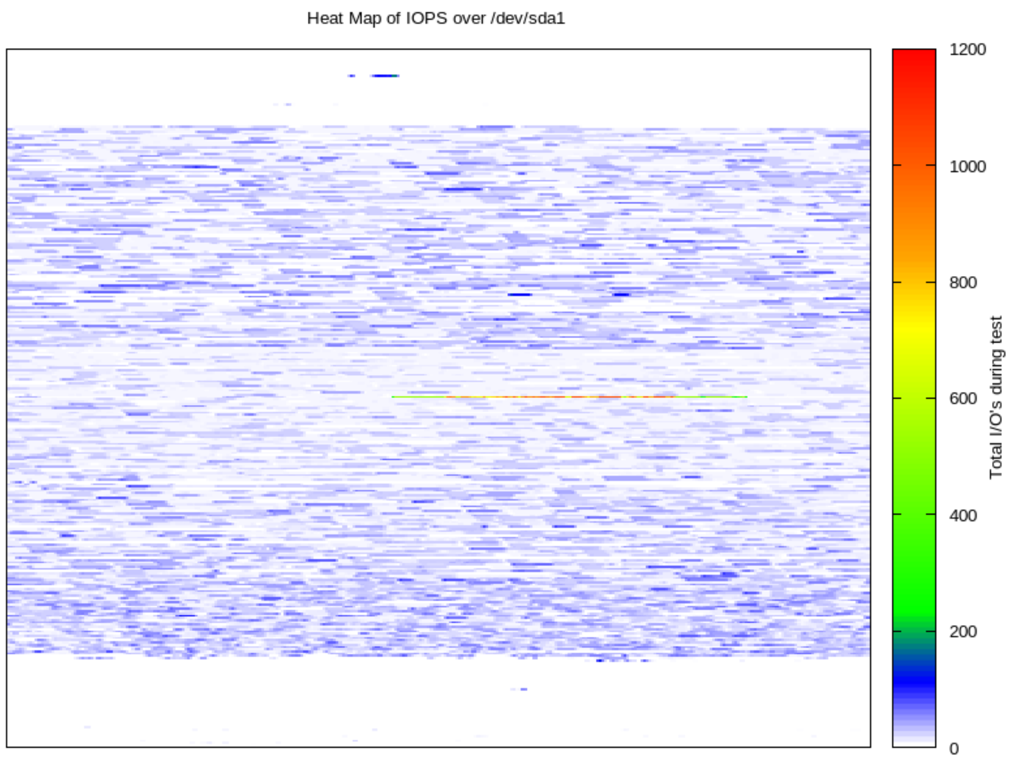


Figure D.6: ioprof IOPS heatmap from pdf report.

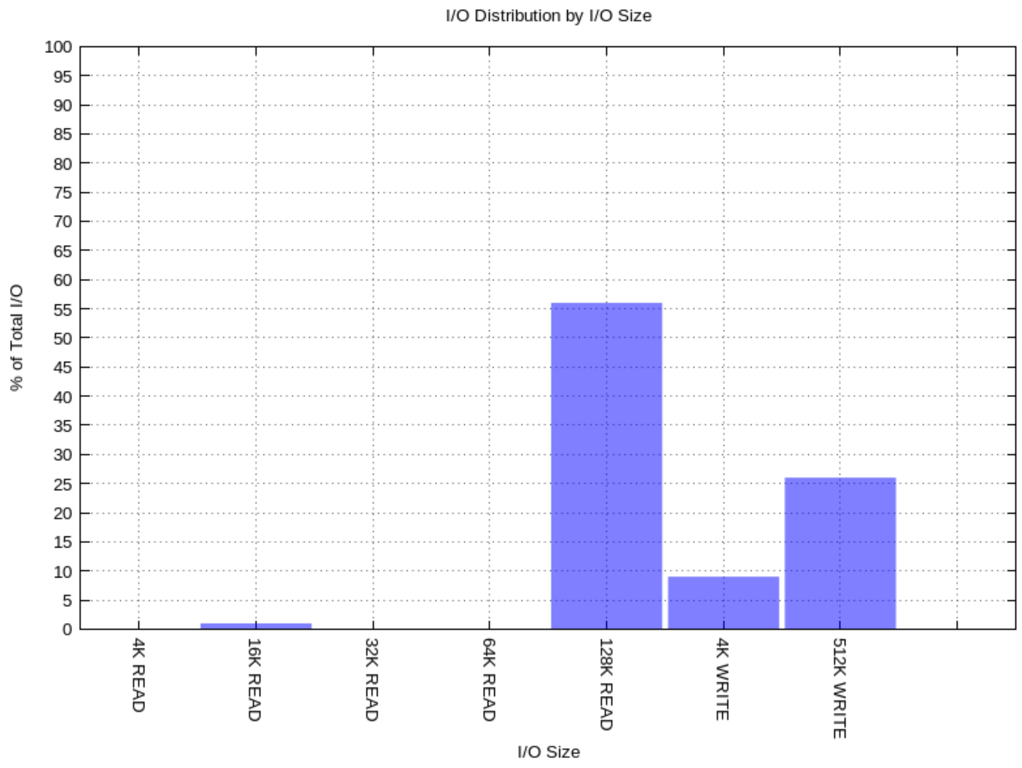


Figure D.7: ioprof IOPS statistics from pdf report.

Appendix E

Command Line and Code Snippets

E.1 Methodology

```
Maintaining 32 concurrent writes of 4194304 bytes for up to 7200 seconds
    or 0 objects
Object prefix: benchmark_data_r610-ceph-3_25506
sec Cur ops   started  finished  avg MB/s  cur MB/s  last lat   avg lat
0       0         0         0         0         0         -         0
1      32        59        27   107.705       108  0.995565  0.652107
2      32       107        75   149.295       192  0.634922  0.678721
3      32       158       126    167.42       204  0.528435  0.660387
4      31       198       167   166.534       164  0.649536  0.658034
5      32       245       213   170.012       184  0.265812  0.688478
6      32       294       262   174.324       196  0.608319  0.684573
7      31       345       314    179.02       208   1.03478  0.669909
8      32       391       359   179.113       180  0.499497  0.673347
....
7187      31    265351    265320   147.402        96   1.33455  0.868269
7188      11    265353    265342   147.394        88   1.67498  0.868311
Total time run:         7201.233967
Total writes made:      265353
Write size:             4194304
Bandwidth (MB/sec):     147.393

Stddev Bandwidth:       46.4686
Max bandwidth (MB/sec): 256
Min bandwidth (MB/sec): 0
Average Latency:        0.868349
Stddev Latency:         0.373631
Max latency:            5.50358
Min latency:            0.146244
```

Listing E.1: Rados bench output for write benchmark with 32 threads, 4MB block size and runtime of 7200 seconds.

E.2 Empirical Studies

E.2.1 Testbed

```
sudo apt-get install --install-recommends linux-generic-lts-utopic
```

Listing E.2: Installation command for installing the Ubuntu enablement stack. The Kernel installed in this example is from 3.16 from Ubuntu 14.10 (Utopic).

E.2.2 Initialization

```
#cloud-config
users:
- name: stefan
passwd: stefan
sudo: ['ALL=(ALL) NOPASSWD:ALL']
groups: sudo
shell: /bin/bash
chpasswd:
list: |
stefan:stefan
expire: False
apt_proxy: http://192.168.1.1:8080
packages:
- mc
- htop
- iotop
- build-essential
- postmark
- tsocks
runcmd:
- wget ftp://192.168.1.1/tsocks.conf --user=ftpuser --password=ftpupl0ad
     -O /etc/tsocks.conf
- echo "http_proxy=http://192.168.1.1:8080" > /etc/wgetrc
- wget http://phoronix-test-suite.com/releases/repo/pts.debian/files/
    phoronix-test-suite_6.2.1_all.deb -O /root/phoronix-test-suite_6.2.1
    _all.deb
- dpkg -i --force-depends /root/phoronix-test-suite_6.2.1_all.deb
- apt-get install -f -y
- phoronix-test-suite
- ufw disable
- /bin/sed -i 's/<DynamicRunCount>TRUE/<DynamicRunCount>FALSE/' /etc/
    phoronix-test-suite.xml
- /bin/sed -i 's/<ProxyAddress>.*/<ProxyAddress>192.168.1.1<\/
    ProxyAddress>/' /etc/phoronix-test-suite.xml
- /bin/sed -i 's/<ProxyPort>.*/<ProxyPort>8080<\/ProxyPort>/' /etc/
    phoronix-test-suite.xml
- cp /usr/share/phoronix-test-suite/deploy/phoromatic-upstart/phoromatic
    -client.conf /etc/init/phoromatic-client.conf
- /bin/sed -i 's/\[35\]/\[235\]/' /etc/init/phoromatic-client.conf
- /bin/sed -i 's/\[0126\]/\[016\]/' /etc/init/phoromatic-client.conf
- /bin/sed -i 's/phoromatic.connect/phoromatic.connect
    192.168.1.2:5800\/Z5GL1Q/' /etc/init/phoromatic-client.conf
- tsocks phoronix-test-suite batch-install pts/postmark pts/dbench
- phoronix-test-suite batch-install pts/pgbench pts/apache pts/iozone
    pts/blogbench pts/unpack-linux pts/fs-mark pts/fio
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/apache-1.6.1/test-definition.xml
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/dbench-1.0.0/test-definition.xml
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/fio-1.8.2/test-definition.xml
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/pgbench-1.5.2/test-definition.xml
- /bin/sed -i 's/$PGBENCH_MORE_ARGS -T 60/$PGBENCH_MORE_ARGS -T 3600/' /
    var/lib/phoronix-test-suite/installed-tests/pts/pgbench-1.5.2/pgbench
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>2/' /var/lib/phoronix-test-
    suite/test-profiles/pts/postmark-1.1.0/test-definition.xml
- /bin/sed -i 's/<Arguments>250000 5120 524288 500/<Arguments>500000
    4096 524288 40000/' /var/lib/phoronix-test-suite/test-profiles/pts/
    postmark-1.1.0/test-definition.xml
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/unpack-linux-1.0.0/test-definition.xml
- /bin/sed -i 's/<TimesToRun>3/<TimesToRun>9/' /var/lib/phoronix-test-
    suite/test-profiles/pts/blogbench-1.0.0/test-definition.xml
- /bin/sed -i 's/startdelay=5/startdelay=15/' /var/lib/phoronix-test-
    suite/test-profiles/pts/fio-1.8.2/install.sh
- /bin/sed -i 's/ramp_time=5/ramp_time=15/' /var/lib/phoronix-test-suite
    /test-profiles/pts/fio-1.8.2/install.sh
- /bin/sed -i 's/runtime=20/runtime=300/' /var/lib/phoronix-test-suite/
    test-profiles/pts/fio-1.8.2/install.sh
- /bin/sed -i 's/size=1g/size=10g/' /var/lib/phoronix-test-suite/test-
    profiles/pts/fio-1.8.2/install.sh
- service phoromatic-client start
```

Listing E.3: Cloud init script to install and configure environment for testing.

```
pts/fio-1.8.2 - Type: Random Read - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 4KB - Disk Target: Default Test Directory -
     Result: IOPS
pts/fio-1.8.2 - Type: Random Write - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 4KB - Disk Target: Default Test Directory -
    Result: IOPS
pts/fio-1.8.2 - Type: Sequential Read - IO Engine: Sync - Buffered: No -
     Direct: Yes - Block Size: 4KB - Disk Target: Default Test Directory
    - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Write - IO Engine: Sync - Buffered: No
    - Direct: Yes - Block Size: 4KB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Random Read - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 32KB - Disk Target: Default Test Directory
    - Result: IOPS
pts/fio-1.8.2 - Type: Random Write - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 32KB - Disk Target: Default Test Directory
    - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Read - IO Engine: Sync - Buffered: No -
     Direct: Yes - Block Size: 32KB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Write - IO Engine: Sync - Buffered: No
    - Direct: Yes - Block Size: 32KB - Disk Target: Default Test
    Directory - Result: IOPS
pts/fio-1.8.2 - Type: Random Read - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 128KB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Random Write - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 128KB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Read - IO Engine: Sync - Buffered: No -
     Direct: Yes - Block Size: 128KB - Disk Target: Default Test
    Directory - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Write - IO Engine: Sync - Buffered: No
    - Direct: Yes - Block Size: 128KB - Disk Target: Default Test
    Directory - Result: IOPS
pts/fio-1.8.2 - Type: Random Read - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 1MB - Disk Target: Default Test Directory -
     Result: IOPS
pts/fio-1.8.2 - Type: Random Write - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 1MB - Disk Target: Default Test Directory -
     Result: IOPS
pts/fio-1.8.2 - Type: Sequential Read - IO Engine: Sync - Buffered: No -
     Direct: Yes - Block Size: 1MB - Disk Target: Default Test Directory
    - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Write - IO Engine: Sync - Buffered: No
    - Direct: Yes - Block Size: 1MB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Random Read - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 32MB - Disk Target: Default Test Directory
    - Result: IOPS
pts/fio-1.8.2 - Type: Random Write - IO Engine: Sync - Buffered: No -
    Direct: Yes - Block Size: 32MB - Disk Target: Default Test Directory
    - Result: IOPS
```

```
pts/fio-1.8.2 - Type: Sequential Read - IO Engine: Sync - Buffered: No -
     Direct: Yes - Block Size: 32MB - Disk Target: Default Test Directory
     - Result: IOPS
pts/fio-1.8.2 - Type: Sequential Write - IO Engine: Sync - Buffered: No
    - Direct: Yes - Block Size: 32MB - Disk Target: Default Test
    Directory - Result: IOPS
```

Listing E.4: Baseline test cases for Phoromatic client on the virtual machines.

```
pts/blogbench-1.0.0 - Test: Write
pts/dbench-1.0.0 - Client Count: 48
pts/postmark-1.1.0
pts/pgbench-1.5.2 - Scaling: On-Disk - Test: Normal Load - Mode: Read
    Write
pts/build-linux-kernel-1.6.0
```

Listing E.5: Test cases for Phoromatic client on the virtual machines for chosen workloads. Only the appropriate test case will be excecuted.

E.2.3 Case Studies

```
# begin crush map
tunable choose_local_tries 0
tunable choose_local_fallback_tries 0
tunable choose_total_tries 50
tunable chooseleaf_descend_once 1
tunable straw_calc_version 1

# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
device 4 osd.4
device 5 osd.5
device 6 osd.6
device 7 osd.7
device 8 osd.8
device 9 osd.9

# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root

# buckets
host r610-ceph-3 {
  id -2   # do not change unnecessarily
  # weight 9.000
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 0.900
  item osd.1 weight 0.900
  item osd.2 weight 0.900
  item osd.3 weight 0.900
  item osd.4 weight 0.900
  item osd.5 weight 0.900
  item osd.6 weight 0.900
  item osd.7 weight 0.900
  item osd.8 weight 0.900
  item osd.9 weight 0.900
}
root default {
  id -1   # do not change unnecessarily
  # weight 9.000
  alg straw
  hash 0  # rjenkins1
  item r610-ceph-3 weight 9.000
}

# rules
rule replicated_ruleset {
  ruleset 0
  type replicated
  min_size 1
  max_size 10
  step take default
  step choose firstn 0 type osd
  step emit
}

# end crush map
```

Listing E.6: Decompiled original CRUSH map on a single host cluster with 10 OSDs.

```
root btrfs {
  id -3   # do not change unnecessarily
  # weight 9.000
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 1.000
  item osd.1 weight 1.000
  item osd.2 weight 1.000
  item osd.3 weight 1.000
  item osd.4 weight 1.000
}
root xfs {
  id -4   # do not change unnecessarily
  # weight 9.000
  alg straw
  hash 0  # rjenkins1
  item osd.5 weight 1.000
  item osd.6 weight 1.000
  item osd.7 weight 1.000
  item osd.8 weight 1.000
  item osd.9 weight 1.000
}

# rules
rule btrfs {
  ruleset 1
  type replicated
  min_size 1
  max_size 10
  step take btrfs
  step choose firstn 0 type osd
  step emit
}
rule xfs {
  ruleset 2
  type replicated
  min_size 1
  max_size 10
  step take xfs
  step choose firstn 0 type osd
  step emit
}
```

Listing E.7: Modified CRUSH map distinguishing between two heterogeneous pools consisting of five OSDs each.

```
rados mkpool btrfs
rados mkpool xfs
ceph osd pool set btrfs crush_ruleset 1
ceph osd pool set xfs crush_ruleset 2
```

Listing E.8: Command to create the pools and assign the newly created CRUSH rules.

E.3 Workload Characterization

```
$ lspci | grep 'Ethernet\|storage'
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X
     Fusion-MPT Dual Ultra320 SCSI (rev 01)
03:00.0 Ethernet controller: VMware VMXNET3 Ethernet Controller (rev 01)
```

Listing E.9: Storage and ethernet adapter presented to a Linux VM deployed on a VMware ESXi host.

```
$ vscsiStats -l
Virtual Machine worldGroupID: 212903, Virtual Machine Display Name:
    Ceph_profiling, Virtual Machine Config File: /vmfs/volumes/56bdd1ad-
    c0f3bbba-ee35-14187747d721/Ceph_profiling/Ceph_profiling.vmx, {
Virtual SCSI Disk handleID: 8205 (scsi0:0)
Virtual SCSI Disk handleID: 8206 (scsi0:1)
Virtual SCSI Disk handleID: 8207 (scsi0:2)
Virtual SCSI Disk handleID: 8208 (scsi0:3)
Virtual SCSI Disk handleID: 8209 (scsi0:4)
Virtual SCSI Disk handleID: 8210 (scsi0:5)
}
Virtual Machine worldGroupID: 256253, Virtual Machine Display Name:
    vmware-io-analyzer-1.6.2, Virtual Machine Config File: /vmfs/volumes
    /56bdd1ad-c0f3bbba-ee35-14187747d721/vmware-io-analyzer-1.6.2/vmware-
    io-analyzer-1.6.2.vmx, {
Virtual SCSI Disk handleID: 8203 (scsi0:0)
Virtual SCSI Disk handleID: 8204 (scsi0:1)
}
```

Listing E.10: Virtual machines and their virtual disks exposed through vscsiStats.

```
$ vscsiStats -s -w 602711 -i 8261
$ vscsiStats -p all -w 602711 -i 8261 -c > histogram.csv
$ vscsiStats -x -w 602711 -i 8261
$ vscsiStats -r
```

Listing E.11: Commands for starting, saving and stopping a recording, followed by the reset command.

```
$ vscsiStats -s -t -w 602711 -i 8266
vscsiStats: Starting Vscsi stats collection for worldGroup 602711,
    handleID 8266 (scsi0:0)
vscsiStats: Starting Vscsi cmd tracing for worldGroup 602711, handleID
    8266 (scsi0:0)
<vscsiStats-traceChannel>vscsi_cmd_trace_602711_8266</vscsiStats-
    traceChannel>
Success.
$ logchannellogger vscsi_cmd_trace_602711_8266 trace_output_file
$ vscsiStats -x -w 602711 -i 8266
$ vscsiStats -e trace_output_file
$ vscsiStats -e trace_output_file > trace_output_file.csv
```

Listing E.12: vscsiStats data collection and logging with subsequent output and conversion from binary to `csv` format.

E.4 Application Traces

```
$ less /sys/block/sda/device/queue_depth
```

Listing E.13: Command to look up the queue depth of a storage device.

```
BEGIN;

UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :
   aid;

SELECT abalance FROM pgbench_accounts WHERE aid = :aid;

UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid
   ;

UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :
   bid;

INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid,
   :bid, :aid, :delta, CURRENT_TIMESTAMP);

END;
```

Listing E.14: The five types of SQL statements used in the pgbench benchmark in the read and write configuration.

```
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

Listing E.15: The SQL statement used in the pgbench benchmark in the read configuration.

Peer reviewed references

[2] Sage A. Weil. "Ceph: Reliable, Scalable, and High-performance Distributed Storage". PhD thesis. Santa Cruz, CA, USA: University of California at Santa Cruz, 2007.

[3] Sage A. Weil et al. "Ceph: A scalable, high-performance distributed file system". In: *Proceedings of the 7th symposium on Operating systems design and implementation.* USENIX Association, 2006, pp. 307–320. URL: `http://dl.acm.org/citation.cfm?id=1298485` (visited on 26/03/2014).

[4] Sage A. Weil et al. "CRUSH: Controlled, scalable, decentralized placement of replicated data". In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing.* ACM, 2006, p. 122. URL: `http://dl.acm.org/citation.cfm?id=1188582` (visited on 26/03/2014).

[5] Sage A. Weil et al. "Rados: a scalable, reliable storage service for petabyte-scale storage clusters". In: *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07.* ACM, 2007, pp. 35–44. URL: `http://dl.acm.org/citation.cfm?id=1374606` (visited on 26/03/2014).

[10] Feiyi Wang et al. "Performance and Scalability Evaluation of the Ceph Parallel File System". In: *Proceedings of the 8th Parallel Data Storage Workshop.* PDSW '13. New York, NY, USA: ACM, 2013, pp. 14–19. ISBN: 978-1-4503-2505-9. DOI: `10.1145/2538542.2538562`. URL: `http://doi.acm.org/10.1145/2538542.2538562` (visited on 19/11/2014).

[12] DongJin Lee, Michael O'Sullivan and Cameron Walker. "Benchmarking and Modeling Disk-based Storage Tiers for Practical Storage Design". In: *SIGMETRICS Perform. Eval. Rev.* 40.2 (Oct. 2012), pp. 113–118. ISSN: 0163-5999. DOI: `10.1145/2381056.2381080`. URL: `http://doi.acm.org/10.1145/2381056.2381080` (visited on 26/11/2014).

[13] DongJin Lee Lee et al. "Robust Benchmarking for Archival Storage Tiers". In: *Proceedings of the Sixth Workshop on Parallel Data Storage.* PDSW '11. New York, NY, USA: ACM, 2011, pp. 1–6. ISBN: 978-1-4503-1103-8. DOI: `10.1145/2159352.2159354`. URL: `http://doi.acm.org/10.1145/2159352.2159354` (visited on 26/11/2014).

[14] Brian F. Cooper et al. “Benchmarking cloud serving systems with YCSB”. In: *Proceedings of the 1st ACM symposium on Cloud computing.* ACM, 2010, pp. 143–154. URL: http://dl.acm.org/citation.cfm?id=1807152 (visited on 03/02/2017).

[21] Pınar Tözün et al. “From A to E: Analyzing TPC’s OLTP Benchmarks: The Obsolete, the Ubiquitous, the Unexplored”. In: *Proceedings of the 16th International Conference on Extending Database Technology.* EDBT ’13. New York, NY, USA: ACM, 2013, pp. 17–28. ISBN: 978-1-4503-1597-5. DOI: 10.1145/2452376.2452380. URL: http://doi.acm.org/10.1145/2452376.2452380 (visited on 06/01/2014).

[22] Qing Zheng et al. “COSBench: Cloud Object Storage Benchmark”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering.* ICPE ’13. New York, NY, USA: ACM, 2013, pp. 199–210. ISBN: 978-1-4503-1636-1. DOI: 10.1145/2479871.2479900. URL: http://doi.acm.org/10.1145/2479871.2479900 (visited on 26/11/2014).

[23] Avishay Traeger et al. “A Nine Year Study of File System and Storage Benchmarking”. In: *Trans. Storage* 4.2 (May 2008), 5:1–5:56. ISSN: 1553-3077. DOI: 10.1145/1367829.1367831. URL: http://doi.acm.org/10.1145/1367829.1367831 (visited on 26/11/2014).

[24] Samuel Lang et al. “I/O Performance Challenges at Leadership Scale”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis.* SC ’09. New York, NY, USA: ACM, 2009, 40:1–40:12. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654100. URL: http://doi.acm.org/10.1145/1654059.1654100 (visited on 26/11/2014).

[27] Michael Sevilla et al. “A Framework for an In-depth Comparison of Scale-up and Scale-out”. In: *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems.* DISCS-2013. New York, NY, USA: ACM, 2013, pp. 13–18. ISBN: 978-1-4503-2506-6. DOI: 10.1145/2534645.2534654. URL: http://doi.acm.org/10.1145/2534645.2534654 (visited on 26/11/2014).

[28] Gene M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* AFIPS ’67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. DOI: 10.1145/1465482.1465560. URL: http://doi.acm.org/10.1145/1465482.1465560 (visited on 02/02/2017).

[29] L. B. Costa and M. Ripeanu. “Towards automating the configuration of a distributed storage system”. In: *2010 11th IEEE/ACM International Conference on Grid Computing.* 2010 11th IEEE/ACM International Conference on Grid Computing. Oct. 2010, pp. 201–208. DOI: 10.1109/GRID.2010.5697971.

[30] Gabriele Bonetti et al. “A Comprehensive Black-box Methodology for Testing the Forensic Characteristics of Solid-state Drives”. In: *Proceedings of the 29th*

Annual Computer Security Applications Conference. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 269–278. ISBN: 978-1-4503-2015-3. DOI: 10.1145/2523649.2523660. URL: http://doi.acm.org/10.1145/2523649.2523660 (visited on 21/02/2017).

[31] Kent Smith. "Understanding SSD over-provisioning". In: 2013. URL: http://flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120822_TE21_Smith.pdf (visited on 27/02/2017).

[33] Michael Yung Chung Wei et al. "Reliably Erasing Data from Flash-Based Solid State Drives." In: *FAST*. Vol. 11. 2011, pp. 8–8. URL: http://static.usenix.org/legacy/events/fast11/tech/full_papers/Wei.pdf (visited on 21/02/2017).

[36] Sitansu S. Mittra. *Database Performance Tuning and Optimization: Using Oracle*. 2003 edition. New York: Springer, 13th Dec. 2002. 489 pp. ISBN: 978-0-387-95393-9.

[37] William Kent. "A Simple Guide to Five Normal Forms in Relational Database Theory". In: *Commun. ACM* 26.2 (Feb. 1983), pp. 120–125. ISSN: 0001-0782. DOI: 10.1145/358024.358054. URL: http://doi.acm.org/10.1145/358024.358054 (visited on 15/08/2017).

[38] C. J. Date. *Database Design and Relational Theory: Normal Forms and All That Jazz*. 1st ed. Sebastopol, Calif: O'Reilly and Associates, 2012. 274 pp. ISBN: 978-1-4493-2801-6.

[39] Surajit Chaudhuri. "An Overview of Query Optimization in Relational Systems". In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS '98. New York, NY, USA: ACM, 1998, pp. 34–43. ISBN: 978-0-89791-996-8. DOI: 10.1145/275487.275492. URL: http://doi.acm.org/10.1145/275487.275492 (visited on 15/08/2017).

[40] Masato Oguchi et al. "Performance Improvement of iSCSI Remote Storage Access". In: *Proceedings of the 4th International Conference on Uniquitous Information Management and Communication*. ICUIMC '10. New York, NY, USA: ACM, 2010, 48:1–48:7. ISBN: 978-1-60558-893-3. DOI: 10.1145/2108616.2108675. URL: http://doi.acm.org/10.1145/2108616.2108675 (visited on 15/08/2017).

[41] M. Oh et al. "Performance Optimization for All Flash Scale-Out Storage". In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 2016 IEEE International Conference on Cluster Computing (CLUSTER). Sept. 2016, pp. 316–325. DOI: 10.1109/CLUSTER.2016.11.

[46] Vasily Tarasov et al. "Benchmarking File System Benchmarking: It *IS* Rocket Science". In: *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011, pp. 9–9. URL: http://dl.acm.org/citation.cfm?id=1991596.1991609 (visited on 02/08/2017).

[55] Windsor W. Hsu, Alan Jay Smith and Honesty C. Young. "I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks&Mdash;an Analysis at the Logical Level". In: *ACM Trans. Database Syst.* 26.1 (2001), pp. 96–143. ISSN: 0362-5915. DOI: `10.1145/383734.383737`. URL: `http://doi.acm.org/10.1145/383734.383737` (visited on 06/01/2014).

[59] Shimin Chen et al. "TPC-E vs. TPC-C: Characterizing the New TPC-E Benchmark via an I/O Comparison Study". In: *SIGMOD Rec.* 39.3 (Feb. 2011), pp. 5–10. ISSN: 0163-5808. DOI: `10.1145/1942776.1942778`. URL: `http://doi.acm.org/10.1145/1942776.1942778` (visited on 09/08/2016).

[78] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: `10.1145/362686.362692`. URL: `http://doi.acm.org/10.1145/362686.362692` (visited on 25/05/2015).

[81] S. Greaves, Y. Kanai and H. Muraoka. "Shingled Recording for 2-3 Tbit/in". In: *IEEE Transactions on Magnetics* 45.10 (Oct. 2009), pp. 3823–3829. ISSN: 0018-9464. DOI: `10.1109/TMAG.2009.2021663`.

[94] "IEEE Standard for Local and metropolitan area networks–Link Aggregation". In: *IEEE Std 802.1AX-2008* (Nov. 2008), pp. 1–163. DOI: `10.1109/IEEESTD.2008.4668665`.

[104] James Turnbull and Jeffrey McCune. *Pro Puppet.* English. Apress, 2011. ISBN: 1430230576 9781430230571 9781430230588 1430230584. (Visited on 12/06/2013).

[105] John Arundel. *Puppet 2.7 Cookbook.* English. Packt, 2011. ISBN: 9781849515399 1849515395 1849515387 9781849515382. (Visited on 12/06/2013).

[110] Stefan Meyer and John P. Morrison. "Impact of Single Parameter Changes on Ceph Cloud Storage Performance". In: *Scalable Computing: Practice and Experience* 17.4 (10th Nov. 2016), pp. 285–298. ISSN: 1895-1767. DOI: `10.12694/scpe.v17i4.1201`. URL: `http://www.scpe.org/index.php/scpe/article/view/1201` (visited on 06/04/2017).

[112] S. Meyer and J. P. Morrison. "Supporting Heterogeneous Pools in a Single Ceph Storage Cluster". In: *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC).* 2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). Sept. 2015, pp. 352–359. DOI: `10.1109/SYNASC.2015.61`.

[113] Carl Henrik Holth Lunde. "Improving Disk I/O Performance on Linux". 2009. URL: `https://www.duo.uio.no/handle/10852/10099` (visited on 10/12/2014).

[114] Robert Love. *Linux System Programming: Talking Directly to the Kernel and C Library.* 1st ed. Beijing ; Cambridge: O'Reilly & Associates, 2007. 388 pp. ISBN: 978-0-596-00958-8.

[115] David Boutcher and Abhishek Chandra. "Does Virtualization Make Disk Scheduling Passé?" In: *SIGOPS Oper. Syst. Rev.* 44.1 (Mar. 2010), pp. 20–

24. ISSN: 0163-5980. DOI: 10.1145/1740390.1740396. URL: http://doi.acm.org/10.1145/1740390.1740396 (visited on 12/12/2016).

[116] Stephen Pratt and Dominique A Heger. "Workload dependent performance evaluation of the linux 2.6 i/o schedulers". In: *2004 Linux Symposium.* 2004. URL: http://landley.net/kdocs/mirror/ols2004v2.pdf#page=139 (visited on 30/06/2015).

[117] X. Zhang, K. Davis and S. Jiang. "iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O". In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium.* 2012 IEEE 26th International Parallel and Distributed Processing Symposium. May 2012, pp. 715–726. DOI: 10.1109/IPDPS.2012.70.

[118] Jaeho Kim et al. "Disk Schedulers for Solid State Drivers". In: *Proceedings of the Seventh ACM International Conference on Embedded Software.* EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 295–304. ISBN: 978-1-60558-627-4. DOI: 10.1145/1629335.1629375. URL: http://doi.acm.org/10.1145/1629335.1629375 (visited on 05/04/2017).

[119] Dror G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation.* 1 edition. New York, NY, USA: Cambridge University Press, 23rd Mar. 2015. 564 pp. ISBN: 978-1-107-07823-9.

[121] Nitin Agrawal et al. "A five-year study of file-system metadata". In: *ACM Transactions on Storage* 3.3 (1st Oct. 2007), 9–es. ISSN: 15533077. DOI: 10.1145/1288783.1288788. URL: http://portal.acm.org/citation.cfm?doid=1288783.1288788 (visited on 17/10/2016).

[122] Andrew S. Tanenbaum, Jorrit N. Herder and Herbert Bos. "File Size Distribution on UNIX Systems: Then and Now". In: *SIGOPS Oper. Syst. Rev.* 40.1 (Jan. 2006), pp. 100–104. ISSN: 0163-5980. DOI: 10.1145/1113361.1113364. URL: http://doi.acm.org/10.1145/1113361.1113364 (visited on 17/10/2016).

[123] Peter M. Chen and Edward K. Lee. "Striping in a RAID Level 5 Disk Array". In: *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems.* SIGMETRICS '95/PERFORMANCE '95. New York, NY, USA: ACM, 1995, pp. 136–145. ISBN: 978-0-89791-695-0. DOI: 10.1145/223587.223603. URL: http://doi.acm.org/10.1145/223587.223603 (visited on 28/04/2016).

[126] I. Ahmad. "Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server". In: *2007 IEEE 10th International Symposium on Workload Characterization.* 2007 IEEE 10th International Symposium on Workload Characterization. Sept. 2007, pp. 149–158. DOI: 10.1109/IISWC.2007.4362191.

[154] Clint Huffman. *Windows Performance Analysis Field Guide.* 1st. Syngress Publishing, 2014. ISBN: 978-0-12-416701-8.

[156] Merwyn Jones and HMC & SE Development Team. *Introduction to the System z Hardware Management Console*. February 2010. IBM Redbooks. IBM Redbooks. 372 pp. ISBN: SG24-7748-00. (Visited on 03/10/2016).

[157] IBM. *MVS Diagnosis: Tools and Service Aids*. V1R13.0. z/OS. IBM. 708 pp. ISBN: GA22-7589-19. (Visited on 03/10/2016).

[158] IBM. *z/OS Problem Management*. Version 1 Release 13. IBM. ISBN: G325-2564-08. (Visited on 03/10/2016).

[159] Gideon Juve et al. “Characterizing and profiling scientific workflows”. In: *Future Generation Computer Systems*. Special Section: Recent Developments in High Performance Computing and Security 29.3 (Mar. 2013), pp. 682–692. ISSN: 0167-739X. DOI: `10.1016/j.future.2012.08.015`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X12001732` (visited on 27/10/2016).

Non-peer reviewed references

[1] OpenStack User Committee. *OpenStack users share how their deployments stack up.* Superuser. URL: `http://superuser.openstack.org/articles/openstack-users-share-how-their-deployments-stack-up` (visited on 22/07/2015).

[6] Sage Weil. “Ceph Erasure Coding amd Cache Tiering”. SCALE13X. 22nd Feb. 2015. URL: `https://www.socallinuxexpo.org/sites/default/files/presentations/20150222%20scale-%20sdc%20tiering%20and%20ec.pdf` (visited on 08/03/2017).

[7] Intel Open Source. *Performance Portal for Ceph | 01.org.* URL: `https://01.org/ceph` (visited on 19/12/2015).

[8] “Build a High-Performance and High-Durability Block Storage Service Based on Ceph”. OpenStack Kilo summit followup. URL: `http://lists.opennebula.org/pipermail/ceph-users-ceph.com/2014-November/044462.html` (visited on 16/12/2015).

[9] Sébastien Han. *Sébastien Han - Blog.* Google Docs. URL: `https://docs.google.com/presentation/d/1a1j_koT7__369_Jes5mnL-fVyA6JK1RXhTJBl2UJ7mc/embed?start=false&loop=false&delayms=3000&usp=embed_facebook` (visited on 19/12/2015).

[11] Feiyi Wang et al. *Ceph parallel file system evaluation report.* Oak Ridge National Laboratory (ORNL); Oak Ridge Leadership Computing Facility (OLCF), 2013. URL: `http://www.osti.gov/scitech/biblio/1104795` (visited on 29/03/2017).

[15] *Welcome to Apache™ Hadoop®!* URL: `http://hadoop.apache.org/` (visited on 03/02/2017).

[16] *Apache Cassandra.* URL: `http://cassandra.apache.org/` (visited on 03/02/2017).

[17] *Voldemort.* URL: `http://www.project-voldemort.com/voldemort/` (visited on 03/02/2017).

[18] *Apache HBase – Apache HBase™.* URL: `http://hbase.apache.org/` (visited on 03/02/2017).

[19] *MongoDB*. MongoDB. URL: `https://www.mongodb.com/index` (visited on 03/02/2017).

[20] *Apache CouchDB*. URL: `https://couchdb.apache.org/` (visited on 03/02/2017).

[25] Mark Nelson. *Ceph Bobtail Performance – IO Scheduler Comparison*. Ceph. 22nd Jan. 2013. URL: `http://ceph.com/geen-categorie/ceph-bobtail-performance-io-scheduler-comparison/` (visited on 31/01/2017).

[26] Mark Nelson. *Ceph Performance Part 1: Disk Controller Write Throughput*. Ceph. 9th Oct. 2013. URL: `http://ceph.com/geen-categorie/ceph-performance-part-1-disk-controller-write-throughput/` (visited on 31/01/2017).

[32] Billy Tallis. *The Samsung 750 EVO (120GB & 250GB) SSD Review: A Return To Planar NAND*. URL: `http://www.anandtech.com/show/10258/the-samsung-750-evo-120gb-250gb-ssd-review-a-return-to-planar-nand` (visited on 27/02/2017).

[34] Andrew Ku. *Second-Generation SandForce: It's All About Compression - OCZ's Vertex 3: Second-Generation SandForce For The Masses*. Tom's Hardware. 24th Feb. 2011. URL: `http://www.tomshardware.com/reviews/vertex-3-sandforce-ssd,2869-3.html` (visited on 21/02/2017).

[35] Andrew Ku. *SandForce: Performance With Incompressible Data - Upgrade Advice: Does Your Fast SSD Really Need SATA 6Gb/s?* Tom's Hardware. 1st Feb. 2012. URL: `http://www.tomshardware.co.uk/sata-6gbps-performance-sata-3gbps,review-32370-6.html` (visited on 21/02/2017).

[42] Jian Zhang and Jiangang Duan. "Best Practices for Increasing Ceph Performance with SSD". FlashMemory Summit 2015. 13th Aug. 2015. URL: `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2015/20150813_S303E_Zhang.pdf` (visited on 02/02/2017).

[43] *Ceph Bobtail JBOD Performance Tuning*. Ceph. 4th Feb. 2013. URL: `http://ceph.com/geen-categorie/ceph-bobtail-jbod-performance-tuning/` (visited on 02/02/2017).

[44] VMware. *Tuning ESX/ESXi for better storage performance by modifying the maximum I/O block size (1003469) | VMware KB*. VMware Knowledge Base. 3rd Oct. 2015. URL: `https://kb.vmware.com/kb/1003469` (visited on 02/02/2017).

[45] Scott Drummonds. *Using vscsiStats for Storage Performance Analysis | VMware Communities*. URL: `https://communities.vmware.com/docs/DOC-10095` (visited on 09/12/2016).

[47] OpenStack User Committee. *OpenStack User Survey Insights: November 2014*. Superuser. URL: `http://superuser.openstack.org/articles/openstack-user-survey-insights-november-2014` (visited on 03/03/2015).

[48] *axboe/fio · GitHub.* URL: https://github.com/axboe/fio (visited on 14/12/2015).

[49] *Web Services Glossary.* URL: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice (visited on 12/12/2016).

[50] *ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4.* URL: https://httpd.apache.org/docs/2.4/programs/ab.html (visited on 30/07/2015).

[51] *OpenBenchmarking.org - Apache Benchmark Test Profile.* URL: https://openbenchmarking.org/test/pts/apache (visited on 30/07/2015).

[52] *OpenBenchmarking.org - Apache Benchmark v1.6.1 Test [apache].* URL: https://openbenchmarking.org/innhold/6011f3aa688b6405a827fc9c57695dfc84cf7564 (visited on 30/07/2015).

[53] J. Katcher. *PostMark: A New File System Benchmark.* Technical Report TR3022. Network Applicance Inc. October 1997. 1997.

[54] *TPC-Homepage V5.* URL: http://www.tpc.org/ (visited on 09/08/2016).

[56] Transaction Processing Performance Council (TPC). *TPC Benchmark A - Standard Specification v2.0.0.* 7th June 1994. URL: http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpca_v2.0.0.pdf (visited on 09/08/2016).

[57] Transaction Processing Performance Council (TPC). *TPC Benchmark B - Standard Specification v2.0.0.* 7th June 1994. URL: http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-b_v2.0.0.pdf (visited on 09/08/2016).

[58] *HammerDB.* URL: http://www.hammerdb.com/ (visited on 12/12/2016).

[60] Transaction Processing Performance Council (TPC). *TPC Benchmark H - Standard Specification v2.17.1.* 13th Nov. 2014. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf (visited on 12/12/2016).

[61] *akopytov/sysbench.* GitHub. URL: https://github.com/akopytov/sysbench (visited on 12/12/2016).

[62] *Manpage for sysbench.* URL: http://man.cx/sysbench(1) (visited on 12/12/2016).

[63] Alexey Kopytov. *SysBench Manual.* URL: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf (visited on 12/12/2016).

[64] *PostgreSQL: Documentation: 9.5: pgbench.* URL: https://www.postgresql.org/docs/current/static/pgbench.html (visited on 09/08/2016).

[65] *PostgreSQL: The world's most advanced open source database.* URL: https://www.postgresql.org/ (visited on 09/08/2016).

[66] *Travis CI - Test and Deploy Your Code with Confidence.* URL: https://travis-ci.org/ (visited on 12/12/2016).

[67] *Jenkins.* URL: https://jenkins.io/ (visited on 12/12/2016).

[68] *Hudson Continuous Integration.* URL: `http://hudson-ci.org/` (visited on 12/12/2016).

[69] *CruiseControl.* URL: `http://cruisecontrol.sourceforge.net/` (visited on 12/12/2016).

[70] Thorsten Leemhuis. *kcbench(1): Kernel compile benchmark - Linux man page.* URL: `https://linux.die.net/man/1/kcbench` (visited on 12/12/2016).

[71] *Seafile.* URL: `https://www.seafile.com/en/home/` (visited on 12/12/2016).

[72] *Nextcloud.* URL: `https://nextcloud.com/` (visited on 12/12/2016).

[73] *ownCloud.org.* URL: `https://owncloud.org/` (visited on 12/12/2016).

[74] *DBENCH.* URL: `https://dbench.samba.org/` (visited on 17/12/2015).

[75] *rados – rados object storage utility — Ceph Documentation.* URL: `http://docs.ceph.com/docs/jewel/man/8/rados/` (visited on 12/12/2016).

[76] *rbd – manage rados block device (RBD) images — Ceph Documentation.* URL: `http://docs.ceph.com/docs/jewel/man/8/rbd/` (visited on 12/12/2016).

[77] *ceph/cbt.* GitHub. URL: `https://github.com/ceph/cbt` (visited on 12/12/2016).

[79] Intel Corporation et al. *Open NAND Flash Interface Specification - Revision 4.0.* 4th Feb. 2014. URL: `http://www.onfi.org/~/media/onfi/specs/onfi_4_0-gold.pdf?la=en` (visited on 22/12/2016).

[80] *High Endurance Technology in the Intel® Solid-State Drive 710 Series.* URL: `http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/ssd-710-series-het-brief.pdf` (visited on 27/05/2015).

[82] *Intel® SSD DC P3700 Series Specifications.* Intel. URL: `http://www.intel.com/content/www/us/en/solid-state-drives/ssd-dc-p3700-spec.html` (visited on 27/05/2015).

[83] Amber Huffman. "NVM Express Overview & Ecosystem Update". Flash Memory Summit 2013. Santa Clara, CA, 13th Aug. 2013. URL: `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130813_A11_Huffman.pdf` (visited on 27/05/2015).

[84] *Dell PowerEdge R200 Specifications.* URL: `http://www.dell.com/downloads/global/products/pedge/en/pe_r200_spec_sheet_new.pdf` (visited on 18/03/2015).

[85] Hewlett Packard Enterprise (HPE). *HP ProLiant DL360 Generation 6 (G6) Quickspecs.* URL: `https://www.hpe.com/h20195/v2/GetDocument.aspx?docname=c04284365` (visited on 13/12/2016).

[86] *Dell PowerEdge R610 Specifications.* URL: `http://www.dell.com/downloads/global/products/pedge/en/server-poweredge-r610-specs-en.pdf` (visited on 18/03/2015).

[87] *LSI PCI Express to 3.0 Gbit-s Serial Attached SCSI (SAS) Host Adapters Users Guide.* URL: `http://www.lsi.com/downloads/Public/Host%20Bus%`

20Adapters/Host%20Bus%20Adapters%20Common%20Files/PCIe_3GSAS_UG.pdf (visited on 18/03/2015).

[88] *Broadcom BCM5709C Product Brief.* URL: http://www.broadcom.com/collateral/pb/5709C-PB02-R.pdf (visited on 18/03/2015).

[89] *Intel® Gigabit ET Quad Port Server Adapter.* Intel® ARK (Product Specs). URL: http://ark.intel.com/products/50398/Intel-Gigabit-ET-Quad-Port-Server-Adapter (visited on 18/03/2015).

[90] *Dell PowerEdge R710 Specifications.* URL: http://www.dell.com/downloads/global/products/pedge/en/R710_spec_sheet.pdf (visited on 18/03/2015).

[91] *HitachiUltrastar A7K1000.* URL: http://www.hgst.com/tech/techlib.nsf/techdocs/DF2EF568E18716F5862572C20067A757/%5C$file/Ultrastar_A7K1000_final_DS.pdf (visited on 11/02/2015).

[92] *Barracuda ES.2 Serial ATA.* URL: http://www.seagate.com/staticfiles/support/disc/manuals/NL35%20Series%20&%20BC%20ES%20Series/Barracuda%20ES.2%20Series/100468393f.pdf (visited on 11/02/2015).

[93] *WD RE4 Series Disti Spec Sheet - 2879-701338.pdf.* URL: http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701338.pdf (visited on 17/12/2015).

[95] *SAS MultiLink logo.* URL: http://www.scsita.org/library/logos/MultiLink_SAS_color.zip (visited on 02/03/2015).

[96] *Dell PowerConnect 5200 Series Switches.* URL: http://www.dell.com/downloads/global/products/pwcnt/en/pwcnt_52xx_specs.pdf (visited on 18/03/2015).

[97] *Dell PowerConnect 6200 Series Switches.* URL: http://www.dell.com/downloads/emea/products/pwcn/PowerConnect_6200_spec_sheet_new.pdf (visited on 18/03/2015).

[98] UpGuard. *Declarative vs. Imperative Models for Configuration Management: Which Is Really Better?* URL: https://www.upguard.com/blog/articles/declarative-vs.-imperative-models-for-configuration-management (visited on 27/12/2016).

[99] *Kernel/LTSEnablementStack - Ubuntu Wiki.* URL: https://wiki.ubuntu.com/Kernel/LTSEnablementStack (visited on 12/02/2015).

[100] *Linux-Kernel Archive: [GIT PULL] Btrfs for 3.19-rc.* URL: http://lkml.iu.edu/hypermail/linux/kernel/1412.1/03583.html (visited on 12/02/2015).

[101] *Puppet Labs: IT Automation Software for System Administrators.* http://puppetlabs.com/. URL: http://puppetlabs.com/ (visited on 06/11/2012).

[102] *Dashboard Manual: Installing – Documentation – Puppet Labs.* http://docs.puppetlabs.com/dashboard/manual/1.2/bootstrapping.html. URL: http://docs.puppetlabs.com/dashboard/manual/1.2/bootstrapping.html (visited on 06/11/2012).

[103] *Foreman.* http://theforeman.org/. URL: `http://theforeman.org/` (visited on 08/11/2012).

[106] Michael Larabel. *Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing, Open-Source Benchmarking.* URL: `http://www.phoronix-test-suite.com/` (visited on 12/04/2016).

[107] Michael Larabel. *OpenBenchmarking.org - An Open, Collaborative Testing Platform For Benchmarking & Performance Analysis.* URL: `https://openbenchmarking.org/` (visited on 12/04/2016).

[108] Michael Larabel. *Phoronix Test Suite - Phoromatic.* URL: `http://www.phoronix-test-suite.com/?k=phoromatic` (visited on 12/04/2016).

[109] *Documentation — Cloud-Init 0.7.7 documentation.* URL: `http://cloudinit.readthedocs.org/en/latest/` (visited on 12/04/2016).

[111] *ceph/config_opts.h – Ceph GitHub.* GitHub. URL: `https://github.com/ceph/ceph` (visited on 05/04/2017).

[120] *Logging and Debugging — Ceph Documentation.* URL: `http://docs.ceph.com/docs/jewel/rados/troubleshooting/log-and-debug/` (visited on 13/12/2016).

[124] *Achieving a Million I/O Operations per Second from a Single VMware vSphere 5.0 Host.* URL: `http://www.vmware.com/resources/techresources/10211` (visited on 06/07/2016).

[125] VMware. *Choosing a network adapter for your virtual machine (1001805) | VMware KB.* VMware Knowledge Base. 22nd Feb. 2016. URL: `https://kb.vmware.com/kb/1001805` (visited on 06/07/2016).

[127] Paul Dunn. *New vscsiStats Excel Macro.* Dunnsept's Blog. 11th Mar. 2010. URL: `https://dunnsept.wordpress.com/2010/03/11/new-vscsistats-excel-macro/` (visited on 08/07/2016).

[128] *Blogbench - About.* URL: `https://www.pureftpd.org/project/blogbench` (visited on 14/07/2016).

[129] *OpenBenchmarking.org - BlogBench Test Profile.* URL: `https://openbenchmarking.org/test/pts/blogbench` (visited on 14/07/2016).

[130] *OpenBenchmarking.org - BlogBench v1.0.0 Test [blogbench].* URL: `https://openbenchmarking.org/innhold/db045f3acfeee21c2d1aeb0e185457939b92134d` (visited on 14/07/2016).

[131] *Maximum Number of UFS Subdirectories (System Administration Guide: Basic Administration).* URL: `https://docs.oracle.com/cd/E19683-01/817-3814/fsfilesysappx-5/index.html` (visited on 14/07/2016).

[132] Erik Trulsson. *UFS2 limits.* E-mail. 9th Nov. 2008. URL: `http://lists.freebsd.org/pipermail/freebsd-questions/2008-November/186081.html` (visited on 14/07/2016).

[133] Intel Corporation and Seagate Technology. *Serial ATA Native Command Queuing*. July 2003. URL: http://www.seagate.com/docs/pdf/whitepaper/D2c_tech_paper_intc-stx_sata_ncq.pdf (visited on 15/07/2016).

[134] helix84. *Native Command Queuing*. URL: https://upload.wikimedia.org/wikipedia/commons/4/4a/NCQ.svg (visited on 15/07/2016).

[135] *OpenBenchmarking.org - PostMark Test Profile*. URL: https://openbenchmarking.org/test/pts/postmark (visited on 26/07/2016).

[136] *OpenBenchmarking.org - PostMark v1.1.0 Test [postmark]*. URL: https://openbenchmarking.org/innhold/9fdd4b7a251888a045e112e930a79d82a35771cd (visited on 26/07/2016).

[137] *OpenBenchmarking.org - Dbench Test Profile*. URL: https://openbenchmarking.org/test/pts/dbench (visited on 09/08/2016).

[138] *OpenBenchmarking.org - Dbench v1.0.0 Test [dbench]*. URL: https://openbenchmarking.org/innhold/91394baa1217292d4bbba46ac8c921f3718325cc (visited on 09/08/2016).

[139] *EUR-Lex - 32003H0361 - EN - EUR-Lex*. URL: http://eur-lex.europa.eu/eli/reco/2003/361/oj (visited on 28/02/2017).

[140] *OpenBenchmarking.org - Timed Linux Kernel Compilation Test Profile*. URL: https://openbenchmarking.org/test/pts/build-linux-kernel (visited on 12/12/2016).

[141] *OpenBenchmarking.org - Timed Linux Kernel Compilation v1.6.0 Test [build-linux-kernel]*. URL: https://openbenchmarking.org/innhold/dda952e40113b276e205198ef88adb6007eee885 (visited on 12/12/2016).

[142] *Performance since PostgreSQL 7.4 / pgbench | PostgreSQL Addict*. URL: https://blog.pgaddict.com/posts/performance-since-postgresql-7-4-to-9-4-pgbench (visited on 09/08/2016).

[143] *OpenBenchmarking.org - PostgreSQL pgbench Test Profile*. URL: https://openbenchmarking.org/test/pts/pgbench-1.5.2 (visited on 08/08/2016).

[144] *OpenBenchmarking.org - PostgreSQL pgbench v1.5.2 Test [pgbench]*. URL: https://openbenchmarking.org/innhold/96ecbfe8776b3e4eff581573a63b18aceb6f50f8 (visited on 08/08/2016).

[145] *CinderSupportMatrix – OpenStack*. URL: https://wiki.openstack.org/wiki/CinderSupportMatrix (visited on 12/02/2014).

[146] Nathan Ruthman and Xyratex. "Rock-Hard Lustre: Trends to Scalability and Quality". SC11 OpenSFS/EOFS. SC11, 2011. URL: http://www.opensfs.org/wp-content/uploads/2011/11/Rock-Hard1.pdf (visited on 24/03/2014).

[147] *Building out Storage as a Service with OpenStack Cloud*. 13th Dec. 2013. URL: http://www.yet.org/2012/12/staas/ (visited on 24/03/2014).

[148] *Gluster Community Website*. URL: http://www.gluster.org/ (visited on 26/03/2014).

[149] Dinesh Subrahveti, IBM. "GPFS OpenStack Integration". Apr. 2013. URL: `http://www.gpfsug.org/wp-content/uploads/2013/05/24-04-13%7B%5C_%7DGPFS-OpenStack%7B%5C_%7DDS.pdf` (visited on 25/03/2014).

[150] *MooseFS network file system.* URL: `http://www.moosefs.org/` (visited on 26/03/2014).

[151] *OpenStack.* GitHub. URL: `https://github.com/openstack` (visited on 13/12/2016).

[152] *Windows Assessment and Deployment Kit (ADK) for Windows 8.1 Update.* Microsoft Download Center. URL: `http://www.microsoft.com/en-us/download/details.aspx?id=39982` (visited on 05/10/2016).

[153] *Windows Performance Toolkit Technical Reference.* URL: `https://msdn.microsoft.com/en-us/library/windows/hardware/hh162945.aspx` (visited on 29/09/2016).

[155] Robert Smith et al. *Analyzing Storage Performance using the Windows Performance Analysis ToolKit (WPT).* Notes from a Platforms Premier Field Engineer. URL: `https://blogs.technet.microsoft.com/robertsmith/2012/02/07/analyzing-storage-performance-using-the-windows-performance-analysis-toolkit-wpt/` (visited on 29/09/2016).

[160] Intel. *Linux* I/O Profiler (ioprof) | 01.org.* URL: `https://01.org/ioprof` (visited on 27/10/2016).

[161] *GitHub 01org/ioprof.* GitHub. URL: `https://github.com/01org/ioprof` (visited on 27/10/2016).

[162] Alan D. Brunelle. *blktrace User Guide.* URL: `http://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html` (visited on 27/10/2016).

[163] Jeff Layton. *ioprof, blktrace, and blkparse » ADMIN Magazine.* ADMIN Magazine. URL: `http://www.admin-magazine.com/HPC/Articles/I-O-Profiling-at-the-Block-Level` (visited on 27/10/2016).

[164] *strace.* SourceForge. URL: `https://sourceforge.net/projects/strace/` (visited on 27/10/2016).