Bachelor Degree Project


UNIVERSITY OF SKÖVDE

# ANALYSING PERFORMANCE EFFECTS OF DEDUPLICATION ON VIRTUAL MACHINE STORAGE

Bachelor Degree Project in Information Technology
Basic level – 22.5HP

Spring term 2017

Marcus Kauküla
University of Skövde

Supervisor: András Márki
Examiner: Birgitta Lindström

# Abstract

Virtualization is a widely used technology for running multiple operating systems on a single set of hardware. Virtual machines running the same operating system have been shown to have a large amount of identical data, in such cases deduplication have been shown to be very effective in eliminating duplicated data.

This study aimed to investigate if the storage savings are as large as shown in previous research, as well as to investigate if there are any negative performance impacts when using deduplication. The selected performance variables are resource utilisation and disk performance.

The selected deduplication implementations are SDFS and ZFS deduplication. Each file system is tested with its respective non-deduplicated file systems, ext4 and ZFS.

The results show that the storage savings are between 72,5 % and 73,65 % while the resource utilisation is generally higher when using deduplication. The results also show that deduplication using SDFS has an overall large negative disk performance impact, while ZFS deduplication has a general disk performance increase.



# Abstract – Swedish

Visualisering används i stor utsträckning för att köra flera operativsystem på en uppsättning hårdvara. Tidigare forskning visar på att mycket utav data som lagras av virtuella maskiner är identisk mellan olika virtuella maskiner. För att eliminera duplicerad data kan deduplicering användas.

Målet med denna studien är att undersöka ifall lagringsbesparingarna är så stora som påvisats i tidigare forskning, men även att undersöka den potentiellt negativa prestandainverkan som kan uppstå vid användning av deduplicering. Det utvalda prestandavariablerna är resursanvändning och diskhastighet.

Det utvalda dedupliceringsimplementationerna är SDFS och ZFS deduplicering. Varje implementation jämförs med deras respektive icke-deduplicerade filsystem, ext4 och ZFS.

Resultaten visar att lagringsbesparingarna är mellan 72,5 % och 73,65 %, medans resursanvändningen är generellt högre vid användning av deduplicering. Resultaten visar också att deduplicering med SDFS resulterar i en generellt stor prestanda minskning, medan ZFS deduplicering resulterar i en generell prestandaökning.



**Keywords:** Virtualization, Virtual machine storage, Deduplication, ZFS, SDFS

# Table of Contents

# 1 Introduction

During the past years, virtualization has become popular for everything from running legacy operating systems on home computers to running massive IaaS services that can provide thousands of virtual servers. With the increase of virtualization, there has also been an increase in the storage required to cope with storing the virtual machines. Deduplication has been a common feature in modern filesystems and is most often used to store backups, but more recent research has explored the possibility of applying deduplication on virtual machine storage. This has been proven effective because of the large similarity between virtual machines.

Previous research on deduplication of virtual machines focuses largely on storage savings, while research on general deduplication has shown multiple negative performance impacts when applying deduplication to data storage. This study therefore focuses on comparing deduplication implementations in current file systems to determine the effect of deduplication on virtual machine storage.

A literature study is performed to evaluate the status of current research within the area. Then an experiment is performed to test multiple deduplication implementations in modern file systems and measure multiple variables to evaluate the performance impact of deduplication. The goal is to provide an information basis that can be used to guide companies and individuals when implementing deduplication on virtual machine storage.

# 2 Background

This section covers subjects and concepts which are the basis for this study.

## 2.1 Virtualization

Early computers in the 1950s only had the capability to run a single specialised program at one time on one machine, the programs had to be made for a special set of hardware which was slow and costly. But with time hardware has become faster and cheaper which started to allow for more functionality. In the early 1960s, M.I.T developed a time-sharing system which allowed for multiple users to interact with a single computer through multiple consoles. IBM introduced their Virtual Machine Facility/370 (VM/370) operating system in 1972, VM/370 which allowed for multiple users to run different programs on seemingly separate computers (Corbató, Merwin-Daggett & Daley, 1962; Creasy, 1981). These are examples of early types of virtualization which have provided a basis for the current virtualization techniques.

With hardware still becoming faster and cheaper the functionality of virtualization has become greater and more widespread. Today's virtualization is mostly used to enable multiple operating systems to share a single set of hardware, this allows for a more effective utilisation of the current powerful hardware (Schlosser, Duelli & Goll, 2011).

While there are many types of virtualization such as storage-, network- and application virtualization. This study focuses on full virtualization since according to Scarfone, Souppaya & Hoffman (2011) it has had an increase in popularity because of its high efficiency and ability to run multiple operating systems (OS) on one computer. In full virtualization, the main component is the virtual machine monitor (VMM), also known as a hypervisor. The hypervisor is responsible for controlling access to the hardware and separating the different running OS. Each virtualized OS is called a virtual machine. The hypervisor can operate in two different ways, native or host. A native-hypervisor, as used by VMware ESXi and Xen, is installed directly on the hardware. While a host-hypervisor, as used by VirtualBox and KVM, is installed on an existing OS (Schlosser et al., 2011).

## 2.2 Virtual machine storage

To store the operating system and other files, a virtual machine needs persistent storage. To provide storage for a virtual machine, a virtual disk is used. The virtual disk is a file created and managed by the hypervisor. While the virtual disk is only a file on the hypervisor, from the virtual machine, it is seen as a hard drive where files can be read and written like a physical hard drive. The virtual disk is most commonly created during the configuration of a new virtual machine but can also be created separately and be assigned to a virtual machine as an extra hard drive (Joshi, Shingade & Shirole, 2014).

There are two general types of virtual disk files used by common hypervisors such as VirtualBox and VMware ESXi. Fixed-sized (VirtualBox), also known as thick (VMware) is when a specific size is set to the virtual disk. If for example a 10 GB virtual disk is created, the file will always occupy 10 GB on the hypervisor. Dynamically allocated (VirtualBox), also known as thin (VMware) is when the file expands as files are written. This means that the file

will only scale as more data is written by the virtual machine and the file will not occupy more space than necessary (Oracle Corporation, 2017; VMware Inc, 2017).

## 2.3 Deduplication

The basic concept of data deduplication is to eliminate duplications of data to reduce the storage size. To perform deduplication the data is divided into chunks. The file system can then determine if two or more chunks are identical. If two or more chunks are identical, only one copy is saved, therefore eliminating storage of duplicated data. There are two ways to divide the data into chunks, fixed-size chunking and variable-size chunking (Mandagere, Zhou, Smith & Uttamchandani, 2008; Jin & Miller, 2009).

Fixed-sized chunking uses a pre-determined size to divide the data. For example, chunking a 10 KB file with a chunk size of 1 KB would result in 10 chunks. Variable-size chunking uses a rolling hash to dynamically determine chunk size, this means that the chunks can vary in size. Variable-size chunking usually provides better deduplication since its more resistant to changes in the data, for example inserting new data in the middle of a file. It however has a negative effect on disk performance since the requests cannot be aligned by hard drive blocks. Both fixed-size and variable-size use hash functions, such as SHA1, to identify chunks. By calculating a hash value for each chunk, a unique value is given for that specific set of data. This allows for hash values to be matched with each other to find duplicates in data without having to match the entire chunk. The result is a faster matching of chunks, which provides better performance. Each hash is stored in a hash table containing the hash and the chunks location on the hard drive (Jin & Miller, 2009; Mandagere et al., 2008).

There are two ways to perform deduplication, in-line and out-of-line. In-line performs the deduplication when the data is written to the disk, this means that each time data is written to the disk a hash is calculated and matched to the hash table to find duplicates. If no duplicates are found the chunk is added to the disk and the hash is saved in the hash table. In case a duplicate is found the data does not need to be stored as a duplicate (Mandagere et al., 2008).

Out-of-band deduplication is when the deduplication process is performed on already stored data. Out-of-band is often performed at regular intervals to update the hash table and remove duplicated data. This means that it potentially does not have the performance impacts that can occur with in-line deduplication since it does not calculate or compare hashes at every write. It however does not provide the same constant storage savings as in-band deduplication and will result in a large burst of disk operations when deduplication is performed (Mandagere et al., 2008).

## 2.4 Deduplication of virtual machine storage

Deduplication of virtual machine storage has been tested to some degree in previous studies, these studies have shown great potential in storage savings due to the high similarity between virtual disks in large virtualization environments (Jin & Miller, 2009; Jayaram et al., 2011). However, these studies have not performed a deeper analysis of the potential performance impacts that deduplication can have. This study therefore compares file systems and deduplication implementations to evaluate the effect of deduplication on virtual machine storage.

Jin & Miller (2009) present in their research that deduplication can save a large amount of storage space when applied to virtual machine storage, therefore storage savings need to be considered when evaluating deduplication. Previous research by Tan & Yan (2016) and Ng et al. (2011) show that the read and write performance are negatively affected when using deduplication, therefore during this study, the disk performance will be measured. Joshi et al. (2014) measure both sequential and random disk operations when measuring the performance of virtual machine storage. Research by Mandagere et al. (2008) indicates that the CPU and memory usage increase when deduplication is used, therefore these variables need to be included when evaluating deduplication.

# 3 Problem definition

This section covers aspects related to the problem definition, such as why the study is performed and what is expected to be achieved.

## 3.1 Motivation

The current studies regarding general deduplication investigate and compares multiple deduplication solutions and techniques. These studies present how data deduplication can be implemented and some performance trade-offs that can be expected in deduplication environments (Mandagere et al., 2008; Tan & Yan, 2016).

Studies have been performed, which applies deduplication on virtual machine storage. These studies have shown great potential in the application of deduplication on virtual machine storage. Previous studies have focused on theoretical implementations or practical implementations of a single deduplication system. Previous research does not present the potential performance impacts that deduplication can have, which is presented in general data deduplication studies (Jin & Miller, 2009; Schwarzkopf, Schmidt, Rüdiger & Freisleben, 2012).

Previous research has shown both great potentials in storage savings and potential negative performance impacts. The motivation of this study is therefore to perform an experiment which compares file systems and deduplication implementations with a practical implementation on virtual machine storage.

## 3.2 Aim

The aim of this study is to evaluate a practical implementation of deduplication using current file systems on virtual machine storage and compare performance between the different file systems and deduplication implementations. The aim is also to create an information basis which can be considered by for example IT-administrators to aid in decision-making regarding the implementation of deduplication on virtual machine storage.

## 3.3 Research question

The research questions which are used to accomplish the aim of this study are:

> *RQ1.: How does deduplication in current file systems compare regarding storage savings when applied to virtual machine storage?*
>
> *RQ2.: How does deduplication in current file systems compare regarding resource utilisation when applied to virtual machine storage?*
>
> *RQ3.: How does deduplication in current file systems compare regarding disk performance when applied to virtual machine storage?*

The research questions are designed to cover all parts of the aim and motivation, as well as to compare and evaluate multiple file systems and deduplication implementations with regard to performance aspects based on previous studies.

## 3.4 Objectives

These objectives are designed and selected to fulfil the aim of the study and to answer the research question.

1. Evaluate previous research in deduplication of virtual machine storage.
2. Design a study that compare file systems and deduplication implementations to answer the research question.
3. Perform the experiment according to the previously created experiment design.
4. Analyse and present the results gathered during the experiment.

## 3.5 Expected results

1. *How does deduplication in current file systems compare regarding storage savings when applied to virtual machine storage?*
The expected results are that there is a large decrease in storage usage when applying deduplication on virtual machines due to their large similarity. The expected storage savings is between 40 % and 80 % based on previous research by Jin & Miller (2009).

*2. How does deduplication in current file systems compare regarding resource utilisation when applied to virtual machine storage?*
It is expected that the resource utilisation will increase when implementing deduplication, specifically CPU and RAM utilisation based on previous research by Mandagere et al. (2008).

3. *How does deduplication in current file systems compare regarding disk performance when applied to virtual machine storage?*
The expected results are that there is a decrease in disk performance when applying deduplication. Both read and write performance is expected to be affected according to research by Tan & Yan (2016) and Ng et al. (2011).

# 4 Method strategy

The method strategy describes the methods that are selected to successfully perform the study and to achieve the set goals.

## 4.1 Literature study

To provide a basis for the study and to find research related to technologies and concepts used in this study a literature study is performed. By performing a thorough literature study the status of previous research can be established and selection of methods and experiment design can be made.

The literature study will be performed according to the systematic literature review method presented in Wohlin et al. (2012). By using the following search strings research related to this study has been found: deduplication storage, deduplication virtual machine, deduplication file system. These search strings have been used to search in online journal databases. The three main databases used are ACM, IEEE and Springer. After finding a related study using the search strings, snowballing is performed. Snowballing allows finding related studies by following the citations of the selected paper or following references in the selected paper (Wohlin et al., 2012).

## 4.2 Experiment

The selected method for answering the research question is an experiment. The method was selected since an experiment according to Wohlin et al. (2012) "are launched when we want control over the situation and want to manipulate behaviour directly, precisely and systematically" (p. 16). An experiment therefore allows to have complete control over the selected variables and to manipulate variables to study the outcome. This will allow for the collection of data which can be reviewed to identify patterns.

No alternative methods have been found that can be applied to this study. A survey is often conducted in retrospect and the results are descriptive and explanatory, therefore does not give the execution and measurement control which an experiment gives. A survey would therefore not be suitable for this study. A case study is based on collecting data and performing statistical analysis, but case studies are only an observational study, which means it does not have the execution control of an experiment and therefore making it unsuitable for this study (Wohlin et al., 2012).

## 4.3 Experiment design

This section describes the experiment design that is used when performing the experiment.

### 4.3.1 Independent variables

Independent variables are according to Wohlin et al. (2012) variables that are being modified and controlled during an experiment. Since the goal of the study is to compare performance and utilisation of different file systems and deduplication implementations, the independent variable for this study is the type of file system or deduplication implementation. The independent variable is modified during the experiment to test multiple file systems and

deduplication implementations. The independent variable is the only variable to be modified during the experiment to provide an accurate comparison.

### 4.3.2 Dependent variables

Dependent variables are according to Wohlin et al. (2012) variables that are being studied and measured. In this study, the dependent variables are the performance variables that are relevant for this study. The first performance variable is the used storage space of the virtual machine images. The storage size is used to calculate storage savings. The second performance variable is the disk speed of the virtual machines. This is measured in read/write speeds and latency. The last performance variable is resource utilisation on the storage location, the measured variables are the usage of CPU and RAM which can be effected by deduplication.

### 4.3.3 Experiment environment

To perform the experiment an environment is created that has the ability to host multiple virtual machines and use different file systems. The selected environment represents a common small scale deployment of virtual machines with a separated storage server. A separate storage server solution is selected since it allows for easier reconfiguration of the virtual machine storage without affecting the hypervisor.

To accommodate both a hypervisor and a storage server two physical computers are used, one configured as a hypervisor and the other configured as a storage server. The storage server has the following specification: Intel Xeon W3550 @ 3.07 Ghz, 48 GB RAM and two Western Digital 1 TB hard drives (WD10EZRZ), and the hypervisor has the following specifications: Intel Xeon W3550 @ 3.07 Ghz, 24 GB RAM and one Seagate 160 GB hard drive (ST3160318AS). The two servers are connected to the same local gigabit Ethernet network through a TP-Link TL-SG108 switch. The third computer is used for management tasks. Figure 1 shows the topology to be used in the experiment.

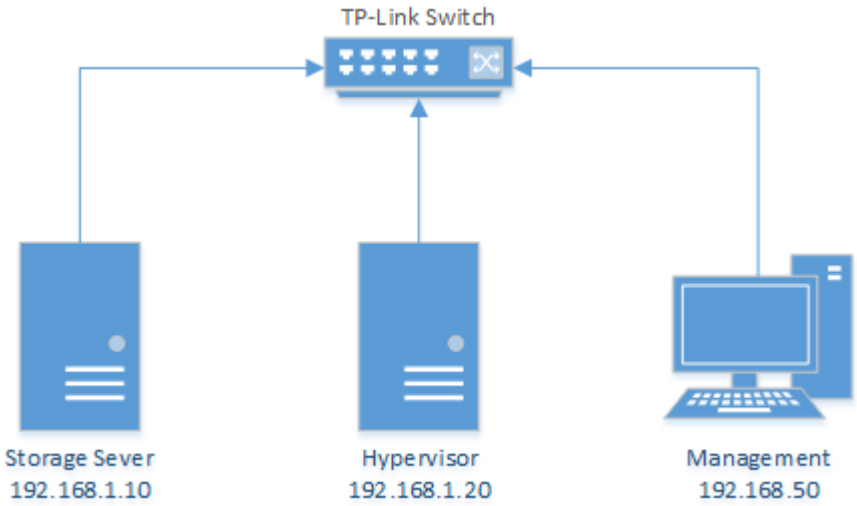



**Figure 1** Experiment topology

The hypervisor is configured with VMware ESXi 6.5, ESXi is used since it is available through the University's lab resources and it is a native hypervisor which is common in server implementations. The storage server is configured with Ubuntu Server 16.04.2 LTS

and NFS. Ubuntu is selected since it is the most popular operating system in large virtualization platforms such as Amazon EC2 and OpenStack (The Cloud Market, 2017 & OpenStack, 2016). The virtual machine storage is distributed to the hypervisor using NFSv4. NFSv4 is used since it provides network-based distribution of storage and is natively supported by ESXi (VMware, 2017). By separating the virtualization from the storage server, it becomes easier to change file systems and deduplication implementations without affecting the virtualization. All virtual machines are stored on the remote storage server where deduplication is implemented, therefore the stored virtual machines are deduplicated.

A set of 10 virtual machines are used throughout the experiment, the virtual machines are using Ubuntu Server 16.04LTS. Ubuntu Server is also used here for the same reasons it was selected for the storage server. The 10 virtual machines are configured to run different types of services to represent a more realistic environment, Table 1 shows each virtual machine and their respective installed packages. Each virtual machine is given the default recommended resource allocations for Ubuntu Server: 1 vCPU, 1 GB RAM and 16 GB hard drive. The hard drive uses thin provisioning since this is the default option for virtual machines stored on NFS storage, and using thick provisioning on NFS storage would require hardware acceleration (VMware, 2017). Networking between the virtual machines are handled by the default ESXi virtual switch and the networking to storage server is handled by the physical network adapter connected to the virtual switch. Due to hardware limitations in the network card, MTU sizes over 1500 is not possible.

| **Name** | **Packages (excluding default utilities)** |
|---|---|
| VM-1-DNS | OpenSSH, bind |
| VM-2-DNS | OpenSSH, bind |
| VM-3-LAMP | OpenSSH, Apache2, MySQL, PHP5 |
| VM-4-LAMP | OpenSSH, Apache2, MySQL, PHP5 |
| VM-5-Mail | OpenSSH, Postfix, Dovecot |
| VM-6-Mail | OpenSSH, Postfix, Dovecot |
| VM-7-Storage | OpenSSH, Samba, DEDISbench, IOPing |
| VM-8-Storage | OpenSSH, Samba, fio |
| VM-9-DHCP | OpenSSH, ISC DHCP |
| VM-10-Database | OpenSSH, PostgreSQL |

**Table 1 –** Virtual machine pool

### 4.3.4 Variable selection

The independent variable is the variable being modified and controlled. The variable has four values which are the two file systems and two deduplication implementations that are used. The first selected file system is Ext4. Ext4 is selected since according to Simon (2015) Ext4 is the updated version of Ext3 and is the recommended and default file system for new

installations. To provide deduplication to ext4 OpenDedup SDFS is used. SDFS is selected since it provides in-line deduplication and was originally created to be a deduplication implementation to be used for virtual machine storage. SDFS is installed on top of an existing file system and ext4 will be used for the base file system (OpenDedup, 2017).

The second selected file system is Oracle ZFS, ZFS is selected since it has built in support for in-line deduplication. ZFS can be used with and without deduplication.

The dependent variables are the variables that are being measured, in this experiment the dependent variables are storage savings, resource utilisation and disk performance. Disk performance is measured from the virtual machines since it is where data is being read and written. To benchmark the virtual machines disk performance DEDISbench and fio are used. DEDISbench is used for write operations since it is designed to benchmark deduplication and will perform write operations with a mix of non-duplicated and duplicated blocks according to a predefined distribution. This will give a more realistic scenario than only using random non-duplicated data as in other benchmark tools (Paulo, Reis, Pereira & Sousa, 2012). Fio is used for benchmarking read operations since DEDISbench read benchmarks only use files filled with zeros and this could cause read operations to only be read from the cache and not from the storage device. Fio uses randomised data which will prevent only reading data from the cache. The files used by fio are created before the benchmark so the files will still be deduplicated and added to the deduplication hash table.

The variables that are measured on the storage server is storage savings and resource utilisation. To calculate storage savings the size of the virtual machines before deduplication is compared to size after deduplication. The standard Unix command du is used to gather the used disk space before and after deduplication.

The second set of variables to be measured on the storage server is resource utilisation, specifically CPU and RAM usage. The variables are measured on the storage server since it is where deduplication is implemented and therefore where an increase is expected according to previous research by Mandagere et al. (2008). These two variables are selected since previous research by Mandagere et al. (2008) shows an increase in CPU and RAM utilisation. CPU utilisation and RAM usage are monitored using dstat.

### 4.3.5 Experiment flow

Figure 2 shows a flow chart over how the experiment is performed. The first steps are to configure the storage and virtualization server according to the previously described design. The next step is to create the pool of virtual machines that will be used during the experiment. After the pool is created the second hard drive on the storage server is formatted with one of the selected file systems and configured with deduplication of required. The pool of virtual machines is then transferred to the new shared storage which then can be accessed by the virtualization server. When the virtual machines are transferred the storage size and resource utilisation can be measured. The next step is the to run the disk benchmarks, after each test the cache on the storage server is cleared to avoid reading from the cache. When the benchmarks are complete the cycle will continue with another file system and/or deduplication implementation until all file systems and deduplication implementations have been tested.

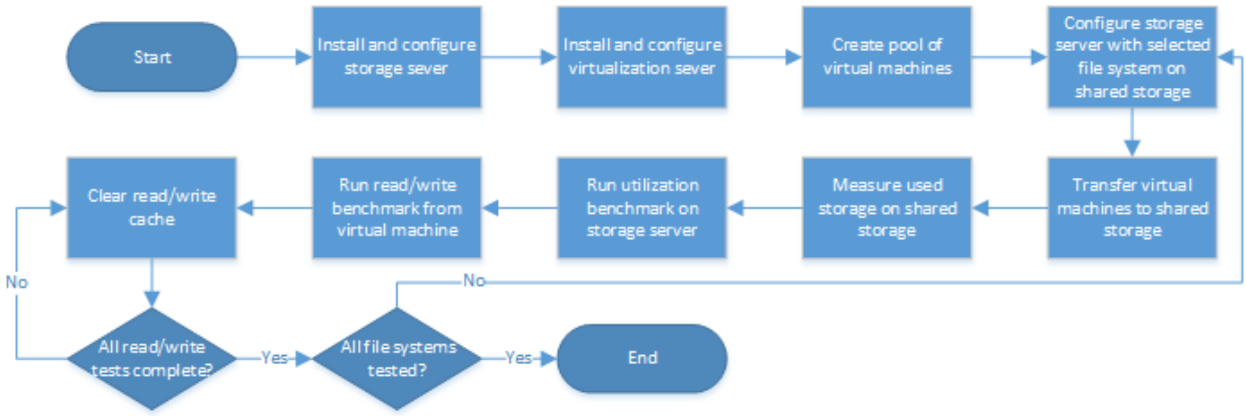



**Figure 2** Experiment flow

## 4.4 Threats to validity

Potential threats to validity have to be handled to provide an experiment that is designed and performed correctly. By handling the identified validity threats it can be assured that the measurements taken are correct and not affected by unidentified variables. This section is performed according to the structure and criteria presented by Wohlin et al. (2012). The list consists of threats that are considered to be applicable to this research.

### 4.4.1 Conclusion validity

Conclusion validity is to ensure that there is a statistical relationship between treatment and outcome. This means that a correct conclusion has been made about the relations between treatment and outcome.

*Low statistical power* is handled by running experiment multiple times therefore collecting a larger amount of data during the experiment. This allows for easier identification of outliers and also allows for a more accurate representation of the data.

*Violated assumptions of statistical tests* are handled by using statistical methods and diagrams to present the data in an accurate and representative way. The statistical method used in this experiment is average over multiple tests and standard deviation to represent the spread in the data. This data is presented in bar graphs. Data which is gathered over longer periods of time, such as CPU utilisation, will be presented in both bar graph and time plot format.

*Fishing and the error rate* is handled by discarding previously gathered data if a change in the method occurs, this prevents the results from being misrepresentative. Also by presenting the data gathering process in a transparent manner, as well as presenting all gathered results, the validity threat is further handled.

*Reliability of measures* is handled by using the same tools and parameters when measuring the dependent variables, this allows to the highest possible reliability of the measurements.

*Random irrelevancies in experimental setting* are handled by not providing internet access to the experiment computers, this eliminates possible irrelevancies from outside sources. By using seeds in disk benchmark tools where possible allows for more replicable results. Also by performing each test multiple times allows for an accurate average representation, especially for random read/write operations. However, for example power outages cannot be

prevented with the available resources. If a power outage occurs during the experiment the gathered at the time of outage will be discarded.

### 4.4.2 Internal validity

Internal validity is to ensure that when a relationship is found between treatment and outcome that it's a causal relationship. This means that the relationship is not a result of a factor which is not controlled or measured.

*History* is handled by disabling internet access to every experiment computer, therefore eliminating possible updates that could potentially affect the experiment outcome.

*Instrumentation* is handled by using well-known benchmark tools that have been used in previous research. Also by reading each tools documentation, correct usage and parameters can be established.

*Ambiguity about direction of casual influence* is handled by using a common non-deduplication file systems as a control to which deduplication file systems can be compared to. By performing disk performance tests both through the virtual machines and locally on the storage server, potential impacts of the network and virtualization can be identified.

### 4.4.3 Construct validity

Construct validity is to ensure that the relationship between theory and observation reflect the construct of the cause and that the outcome will reflect the construct of the effect.

*Inadequate preoperational explication of constructs* is handled by performing a thorough literature study, which provides a scientific basis for the experiment.

*Restricted generalizability across constructs* is handled to a certain degree by observing the most related variables, but it is possible that there may be certain other variables that may be affected in a larger scale experiment. This is however not possible to test due to time and resource restrictions.

### 4.4.4 External validity

External validity is to ensure that eventual casual relationships can be generalised outside the scope of the study.

*Interaction of setting and treatment* is handled to a certain degree by using common file systems, deduplication implementations and virtualization techniques likely to be used in a real-life scenario. But due to time and resource limitations, it is not possible to perform the experiment in a large-scale production environment.

## 4.5 Ethics

The selected methods are an experiment and a literature study, therefore many ethical problems that are applicable to surveys and case studies do not apply. The ethics for this study is that no experiment results have been modified and that no fishing for tests results has occurred, and that the results have been presented in a clear and transparent manner. The experiment process as a whole is presented in a transparent manner to guarantee that this has been followed.

# 5 Experiment Implementation

This section describes the implementation of the experiment design.

## 5.1 Environment setup

This section describes the setup of the experiment environment which is used when performing benchmark tests.

### 5.1.1 Network

During the initial setup, each computer is connected to the university's lab network to provide access to the local netboot environment from which the operating systems are installed from. Internet access is also available from the university's network which allows for installation of the required packets before migrating to the experiment network. The experiment network consists of one network switch connected to each experiment computer, this allows for minimal potential bottlenecks in network traffic and eliminates possible effects from outside hosts.

### 5.1.2 Storage server

Ubuntu 16.04.2 LTS is deployed to the storage server via the local netboot. The operating system is installed through the GUI install wizard. During the installation, the operating system is installed on one of the local hard drives, automatic updates are disabled and both OpenSSH and standard system utilities are selected for automated installation. After the operating system is installed, the benchmark utilities and file systems are installed using the aptitude packet manager. The following packets are installed: dstat, zfs, sdfs-latest, nfs-kernel-server.

### 5.1.3 Hypervisor and virtual machines

VMware ESXi 6.5 is also installed through the local netboot. The hypervisor is installed through the install wizard using default options and the installation location is the computers local hard drive. After ESXi is installed the virtual machines are created. First, a virtual machine template is created. The virtual machine template is created using the standard settings for an Ubuntu machine, 1 vCPU, 1 GB RAM and 16 GB thin storage. Netboot is used to deploy the operating system (Ubuntu Server 16.04.2 LTS) to the virtual machine template. Ubuntu is installed using the default options during the installation with automatic updates disabled and standard system utilities. The virtual machine template is then exported as an OVF and vmdk file. These files are then used to deploy each of the virtual machines according to Table 1. Each machine is deployed from the template and the desired packets are installed on each machine according to Table 1. When all virtual machines have been deployed and configured, each machine is exported to OVF and vmdk files. These files are then deployed on the networked storage for each tested file system and deduplication implementation.

## 5.2 Experiment execution

This section describes configuration of each file system and deduplication implementation, as well as the configuration parameters of each benchmark tool.

### 5.2.1 Storage and resource utilisation

Storage measurement is taken after all virtual machines have been deployed to the shared file systems, and when deduplication implementations are used, after deduplication has been performed. The following command is used when measuring storage utilisation:

```
du -hs
```

The variable “h” is used to present storage usage in human readable format (gigabyte) and the “s” variable is used to only present the total used storage size.

When measuring resource utilisation, the program dstat is used by using the following command:

```
dstat –cdnm –output file-name.csv
```

The variables “cdnm” is used to measure CPU utilisation, disk usage, network utilisation and memory usage. “Output” is used to print the results to a csv file. Dstat writes all measurements to the selected file at one-second intervals. The resource utilisation is measured for 10 minutes during random read, random write and idle usage, random read/write operations are used to prevent data from only being read/written from caches.

### 5.2.2 I/O Benchmarks

This section describes how each of the I/O performance benchmark is performed. Each benchmark is performed three times which is used to provide an average over multiple runs, this to provide a representative value for each benchmark.

For the read benchmarks, the program fio is used. The following fio job file is used to specify the parameters to be used during read benchmarks:

```
; -- Start job file --
[readtest]
ioengine=libaio
iodepth=4
rw=read/randread (sequential or random read operations)
bs=1k
direct=0
size=2048m
numjobs=1
; -- end job file --
```

The job file is based on the parameters used by Joshi et al. (2014). The parameter rw is changed to read or randread depending on which type of I/O operation is being tested. And the block size parameter is changed to 1k to minimise network fragmentation since the largest available MTU is 1,5k due to hardware limitations. The file size is enlarged to 2048MB (double the virtual machines RAM) to prevent the file from being cached by the virtual machine.

For write benchmarks, DEDISbench is used. The following DEDISbench command is used when benchmarking:

```
./DEDISbench –p –w –t1 –b1024 –c1 –vabc123 –a0/a1
```

The “p” parameter enables peak mode meaning that the program will use the full I/O bandwidth. To set the benchmark to write operations the “w” parameter is used. The block size is, as in the read benchmark, set to 1k with the “b” parameter due to the smaller MTU size used. By setting the “c” parameter to 1, only one file is being written. The “v” parameter specifies a seed to be used by the benchmark program, this is set to abc123 and allows for the write operations to be identical over multiple runs, providing more consistent and replicable results. The “a” parameter specifies which type of I/O access is being used, “a0” is used when sequential write operations are being tested and “a1” is used when random write operations are tested.

IOPing is used to measure I/O latency. The following command is used:

```
ioping /tmp –s 1k –W –c 10
```

The “/tmp” variable specifies that the program uses the temporary directory to perform the test. Variable “s” is set to “1k” to match the used MTU size. The variable “W” is used to enable usage of write operations instead of read operations, this it used to elude the read cache that is used when performing a read operation test. Variable “c” is set to 10 to perform 10 latency tests and produce an average latency number.

### 5.2.3 Ext4

The first file system being tested is ext4. The file system is created and shared using to following commands:

```
mkfs.ext4 /dev/sdb
mkdir /nfs
mount /dev/sdb /nfs
echo “/nfs  192.168.1.10(rw,sync,no_subtree_check)” >> /etc/exports
```

First, ext4 is created on the secondary hard drive. A new directory is then created and the newly created file system is mounted to the new directory. The file system is then shared over NFS by adding the mount location to the NFS exports file.

### 5.2.4 SDFS

SDFS is the first deduplication implementation to be tested, SDFS is configured on top of a new ext4 filesystem. The following commands are used to configure SDFS:

```
echo "* hard nofile 65535" >> /etc/security/limits.conf
echo "* soft nofile 65535" >> /etc/security/limits.conf
mkfs.ext4 /dev/sdb
mkdir /nfs
mkfs.sdfs --volume-name=dedup --volume-capacity=900GB –-base-path=/nfs –
io-safe-close=false
mkdir /nfs-sdfs
```

```
mount –t dedup /nfs-sdfs
echo “/nfs-sdfs   192.168.1.10(rw,sync,no_subtree_check)” >> /etc/exports
```

The file size limits are changed according to the Administration Guide (OpenDedup, 2017), then a new ext4 file system is created on the secondary hard drive according to the steps presented in the section 5.2.3. A new SDFS volume is then created with the name “dedup”, a capacity of 900 GB and with the base path set to “/nfs”. By setting the base path to “/nfs” all data is stored on the secondary hard drive. The variable “io-safe-close” is set to false according to the Administration Guide (OpenDedup, 2017) since the volume is shared using NFS. A new directory is created and the SDFS volume is mounted to the new directory. The SDFS volume is shared using NFS with the same parameters used with ext4.

### 5.2.5 ZFS

The file system ZFS is the first tested file system which has integrated deduplication support. The following commands are used when configuring ZFS:

```
zpool create nfs /dev/sdb
zfs set sharenfs=on nfs

zfs  set dedup=on nfs
```

The zpool command is used to create a new ZFS pool named “nfs” on the secondary hard drive. An NFS share is created using the command “zfs set” to enable NFS sharing on the previously created ZFS pool. The same command is used to enable deduplication on the ZFS pool.

# 6 Results

In this section, the results gathered from the experiment are presented.

## 6.1 Storage

Figure 3 shows the used disk space for each file system and deduplication implementation. The disk space is presented in gigabytes.

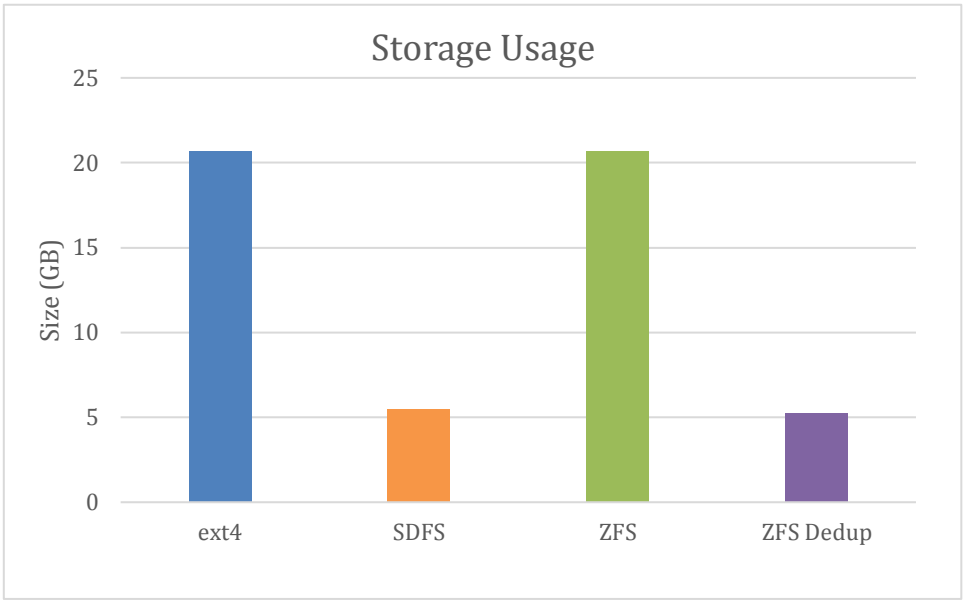



**Figure 3** Storage usage

As shown in Figure 3, both non-deduplication tests show identical used disk space of 20,7 GB. When deduplication is applied a significant decrease in used storage is shown. The results show that SDFS uses 5,5 GB, a decrease of 72,5 %. And ZFS with deduplication shows a disk usage of 5,27 GB, a decrease of 73,65 %.

## 6.2 Resource utilisation

Figure 4 shows the RAM usage when the system is in an idle state without cached data. RAM usage is presented in megabytes.

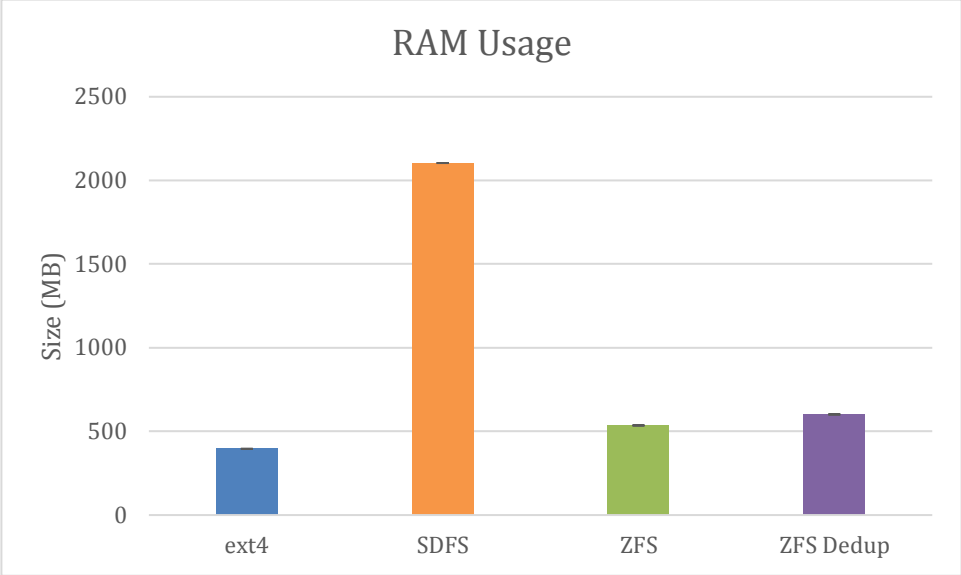



**Figure 4** RAM Usage

The figure shows that when deduplication is not present, ext4 and ZFS uses 395,90 MB and 535,72 MB respectively. The ext4 based deduplication implementation SDFS uses 2104,06 MB, which is an increase of 431,46 % over ext4. ZFS deduplication uses 601,59 MB of RAM, which is an increase of 12,30 % over ZFS without deduplication. The standard deviation shows a very small spread in test results, indicating that the RAM usage is highly consistent.

Figure 5 shows a time plot of CPU usage during constant read operations over 10 minutes.

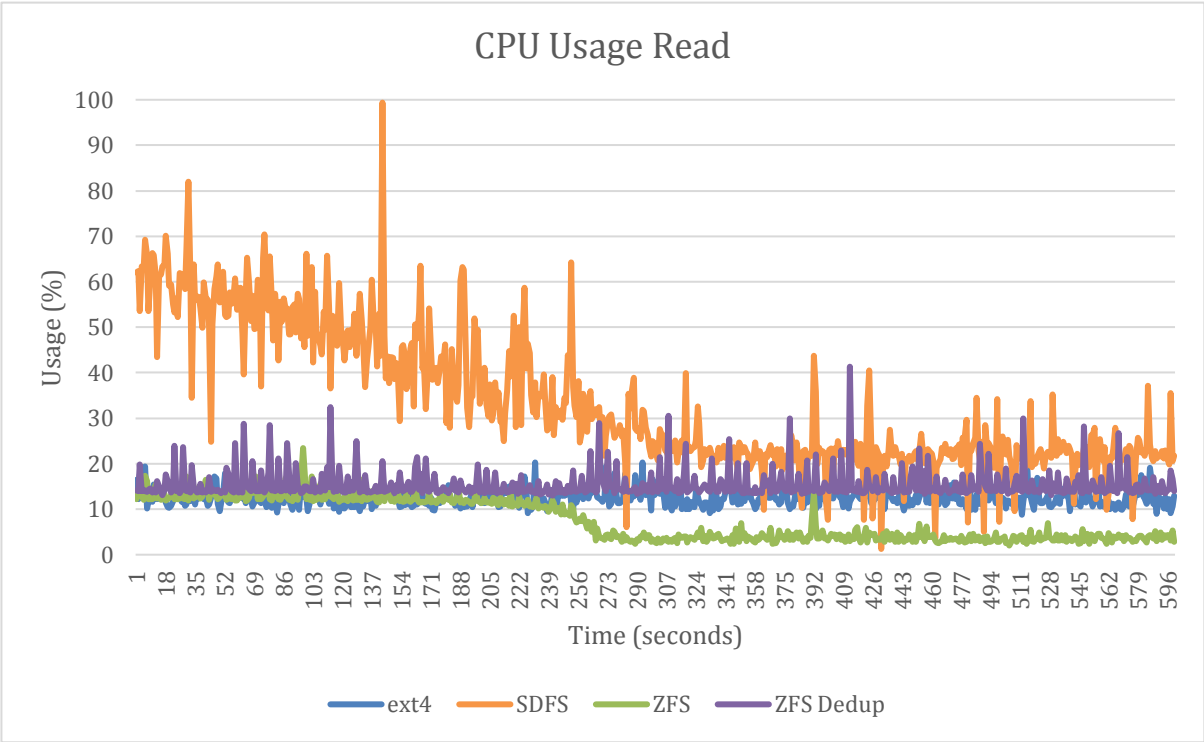



**Figure 5** CPU Usage Read Time plot

Figure 5 shows that ext4 has a consistent CPU usage over 10 minutes, while ZFS initially has a similar CPU usage as ext4, but after about 256 seconds drop to a lower CPU usage. SDFS initially has a significantly higher CPU usage compared to the other file systems and deduplication implementations, SDFS has a steady decline CPU usage rate which levels off at about 300 seconds. ZFS deduplication has an overall higher CPU utilisation than both ZFS without deduplication and ext4.

Figure 6 shows the same data as Figure 5 but in bar graph format.

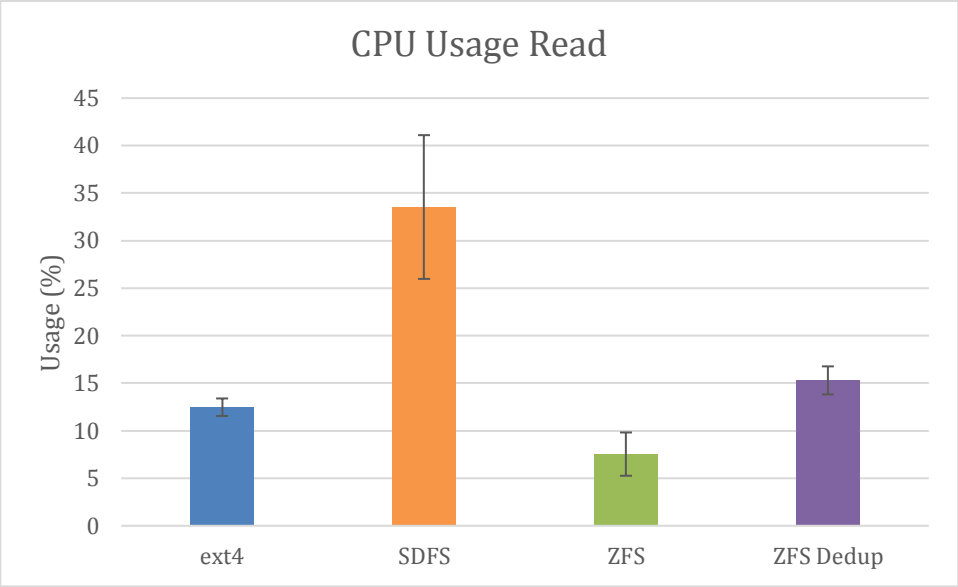



**Figure 6** CPU Usage Read Bar graph

Figure 6 shows that ext4 has an average CPU utilisation of 12,48 % and that the CPU usage is very consistent with a standard deviation of 1,82. SDFS shows a much higher CPU utilisation of 33,54 % on average, SDFS therefore has an increase of 168.75 % compared to ext4. SDFS also has a much larger variance in the test results due to its initially high CPU usage with a standard deviation of 15,12.

ZFS has an average CPU utilisation 7,54 % and a large variation with a standard deviation of 4,92. As shown in Figure 5, the reason for the large variation is the decrease CPU usage after about 256 seconds. ZFS deduplication show an average CPU utilisation of 15,3 % which is an increase of 102,92 % over ZFS without deduplication. ZFS deduplication has a lower variation with a standard deviation of 2,95.

Figure 7 shows CPU usage after 300 seconds during read operations.

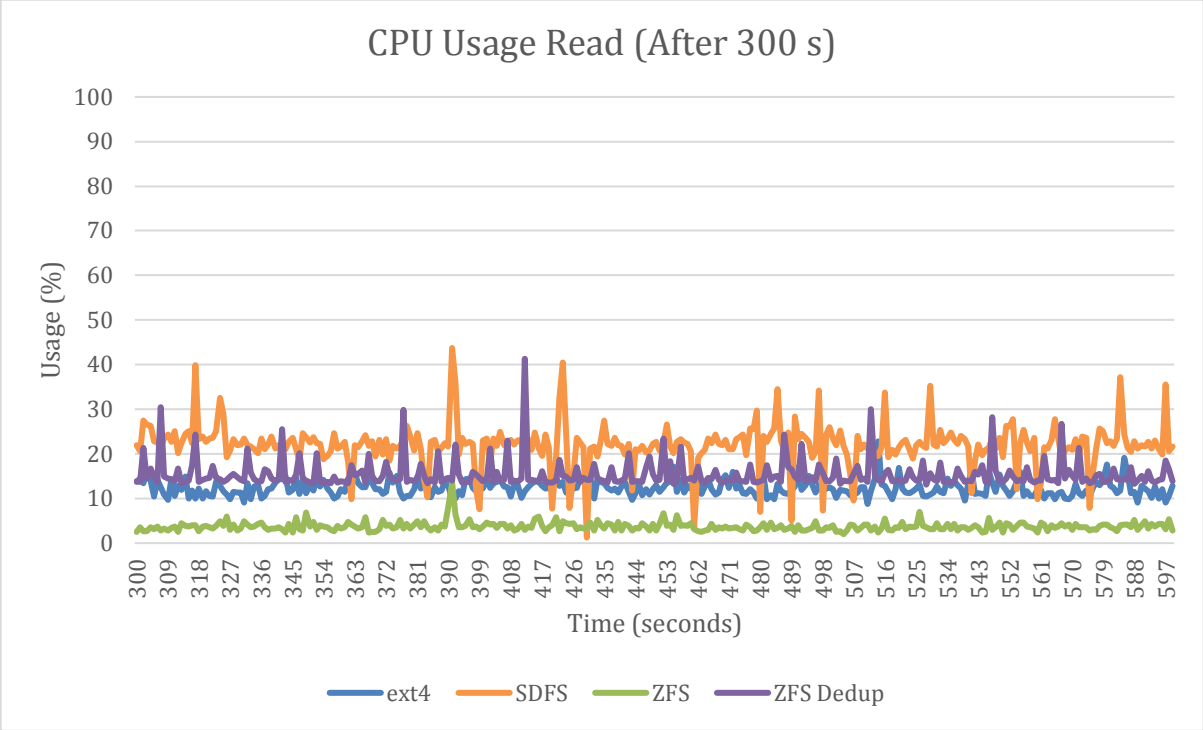



**Figure 7** CPU Usage Read (After 300s)

When comparing test results of CPU usage during read operations (Figure 5) to CPU usage during read operations after 300 seconds (Figure 7) it is clear that the results are more consistent. The results show that the variation is low on all file systems and deduplication implementations, with SDFS and ZFS having the largest difference compared to test results from Figure 5.

Figure 8 shows the same data as Figure 7 but in bar graph format.

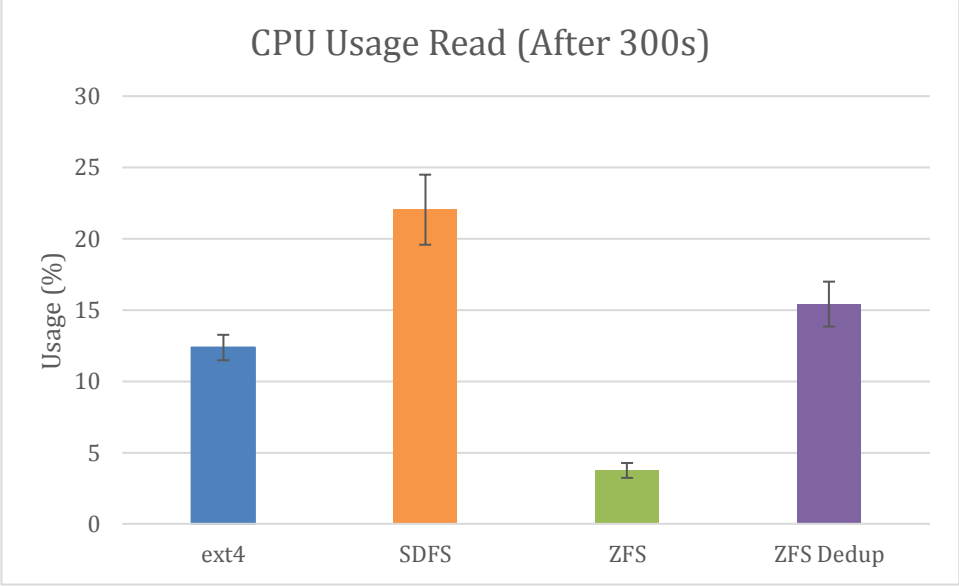



**Figure 8** CPU Usage

Figure 8 shows an overall smaller variation in test results compared to the results presented in Figure 6. This indicates that the CPU usage after 300 seconds is more consistent in some cases compared to CPU usage from the full test duration. Ext4 and ZFS deduplication shows no statistical difference between the full duration compared to after 300 seconds. ZFS shows a decrease in CPU usage of 50,26 %. The standard deviation for ZFS is also lower with a value of 1,04 compared to 4,55. SDFS also shows a large decrease in CPU usage when comparing test results from the full duration to results after 300 seconds, the decrease is 34,29 %. The standard deviation for SDFS is also significantly lower with a value of 4,92 compared to 15,12.

Figure 9 shows CPU usage during constant write operations over a time period of 10 minutes.

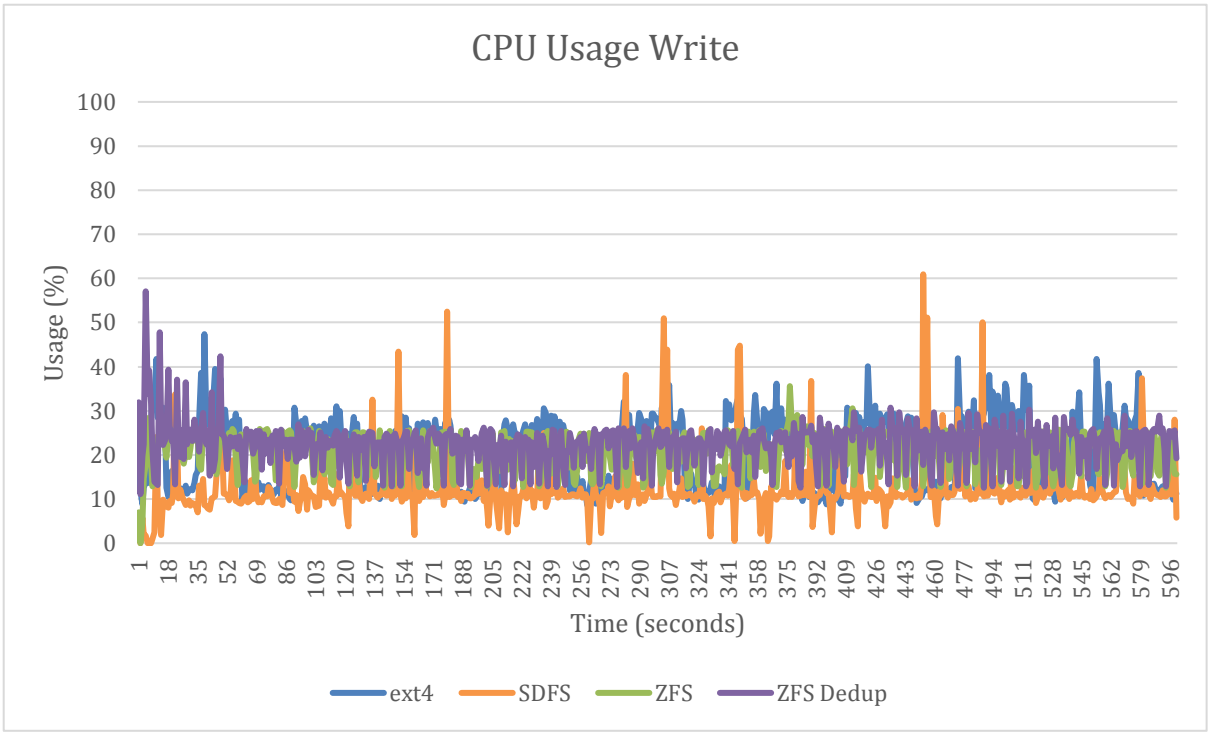



**Figure 9** CPU Usage Write Time plot

Figure 9 shows a more concentrated CPU utilisation for all file systems and deduplication implementations. Ext4 has a larger variation compared to both ZFS implementation during write operations. SDFS has a smaller CPU utilisation but with high peaks of utilisation at different intervals. ZFS has a similar CPU utilisation compared to ext4 but with smaller peaks. ZFS deduplication has a similar CPU utilisation to ZFS.

Figure 10 shows CPU usage during write operations but in a bar graph format with standard deviation.

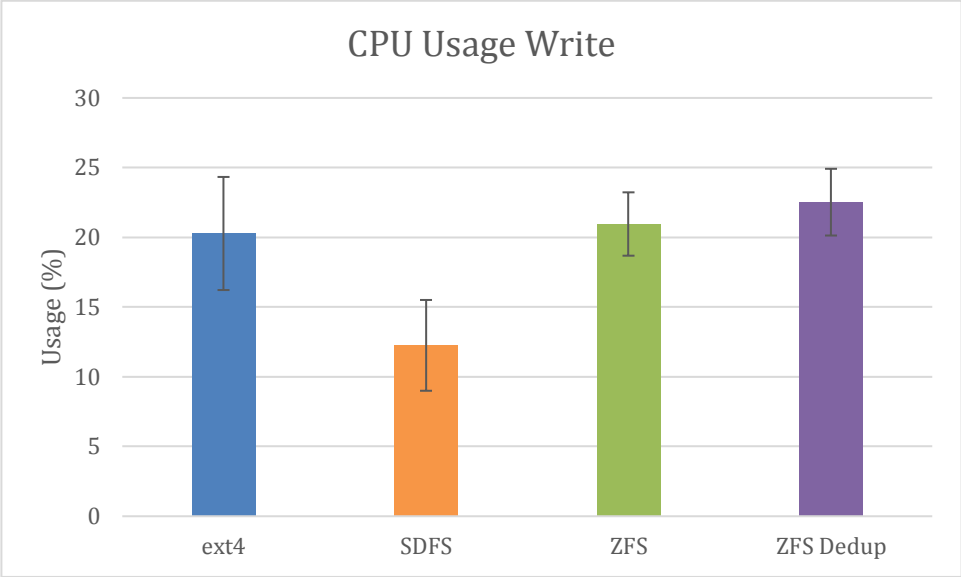



**Figure 10** CPU Usage Write Bar graph

Figure 10 shows that ext4 has an average CPU utilisation of 20,28 % and a large variation with a standard deviation of 8,11. SDFS has an average CPU utilisation of 12,25 % which is a decrease of 39,60 % compared to ext4. SDFS also has a slightly lower variation than ext4, with a standard deviation of 6,51. ZFS average CPU utilisation is very similar to ext4 with a usage of 20,96 %, ZFS also has a smaller variation with a standard deviation of 4,54. ZFS deduplication has an increased average CPU utilisation compared to ZFS, the increase is 8.05 % with an average CPU utilisation of 22,52 %. The variance is slightly higher compared to ZFS without deduplication, the standard deviation is 4,78.

The standard deviation of all file systems and deduplication implementations is high in this test. The result is that no statistical difference can be proven between the two ZFS tests. However, between ext4 and SDFS, the difference is more distinct.

## 6.3 Disk performance

The disk performance results show the used bandwidth for each file system and deduplication implementation during each performance test.

Figure 11 shows disk performance during sequential read operations.

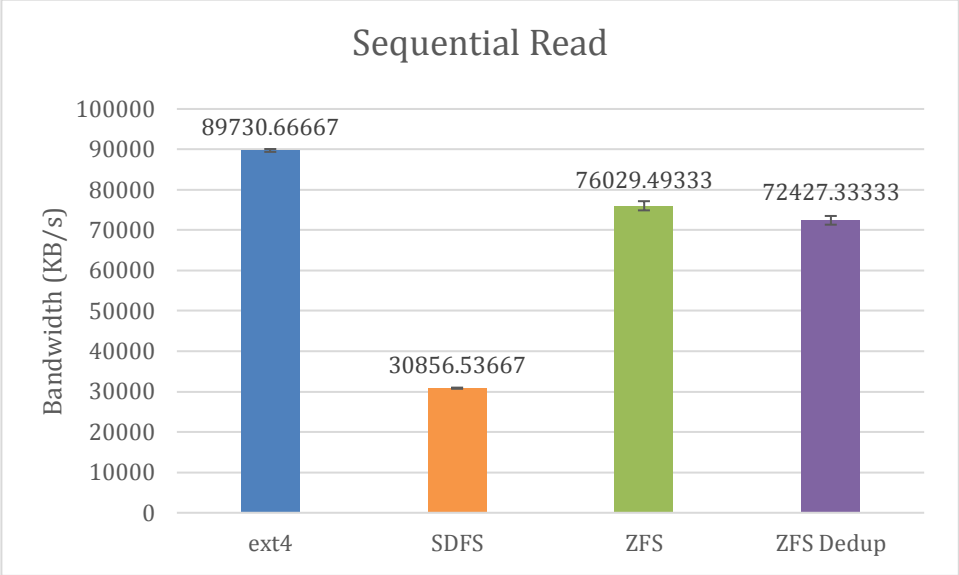



**Figure 11** Sequential Read

Figure 11 shows that ext4 has a higher average sequential read bandwidth than SDFS with a small standard deviation of 173,47. SDFS has a much lower bandwidth with a decrease of 65,61 % compared to ext4. SDFS also has a larger variation in the test compared to ext4 with a standard deviation of 282,57. ZFS deduplication has a lower bandwidth compared to ZFS, the decrease in bandwidth is 4,74 %. The standard deviation for ZFS deduplication is 2157,62 which is lower compared to ZFS, which has a standard deviation of 2270,16. Comparing the two deduplication implementations show that SDFS has a 57,40 % decrease in bandwidth compared to ZFS deduplication.

Figure 12 shows bandwidth usage during random read operations.

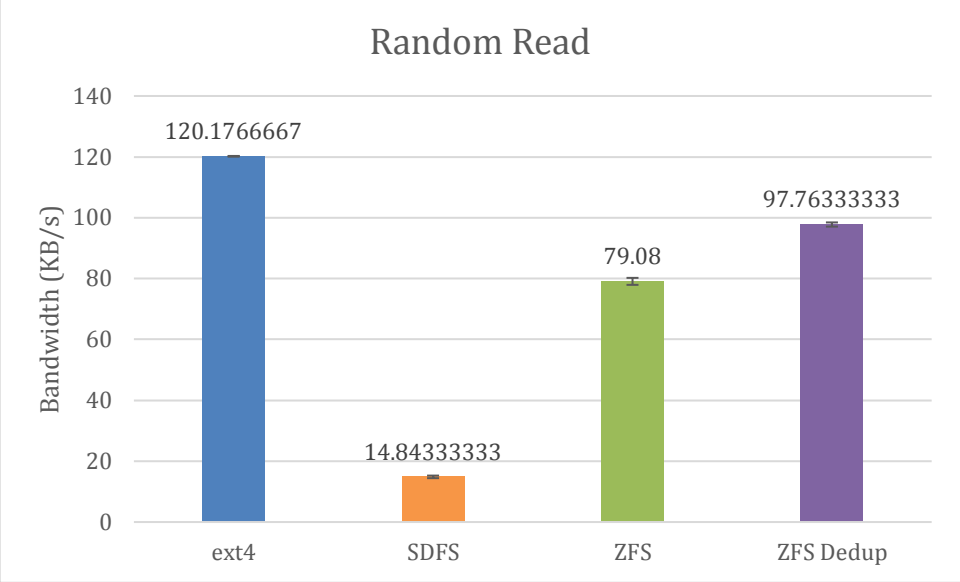



**Figure 12** Random Read

Figure 12 shows that SDFS has a large decrease in performance compared to ext4. The decrease between ext4 and SDFS is 84,64 %. Ext4 has a lower standard deviation of 0,24, compared to the standard deviation of SDFS which is 0,82. Unlike ext4 and SDFS, ZFS shows an increase in performance when using deduplication. ZFS deduplication shows a 23,62 % increase in bandwidth compared to ZFS. ZFS deduplication also has a lower standard deviation compared to ZFS. ZFS deduplication has a standard deviation of 1,38, while ZFS has a standard deviation of 2,30. SDFS has a decrease of 84,82 & compared to ZFS deduplication.

Figure 13 shows the bandwidth of each tested file system and deduplication implementation when testing sequential write operations.

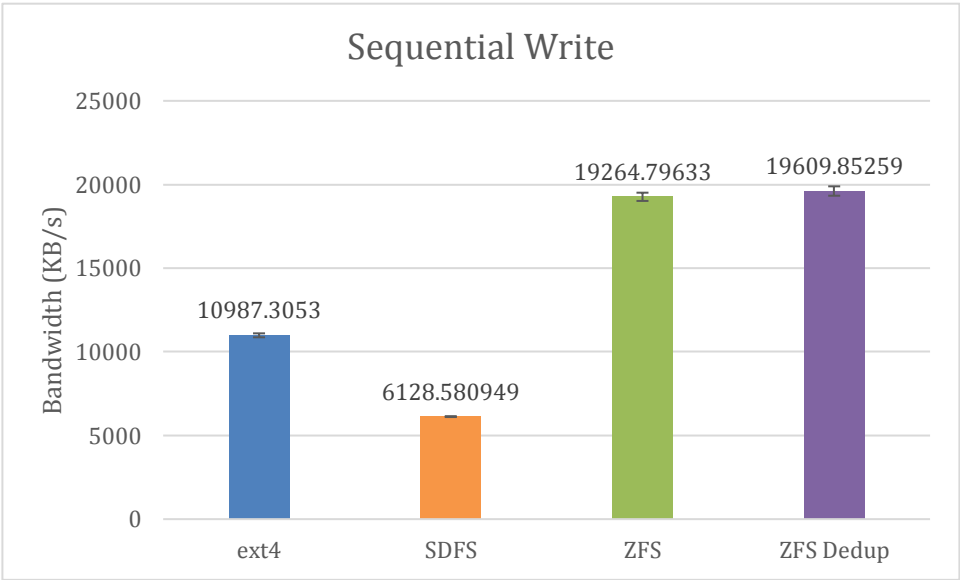



**Figure 13** Sequential Write

Figure 13 shows that ext4 has a higher bandwidth compared to SDFS. The decrease in bandwidth from ext4 to SDFS is 44,22 %. The variation in results is higher for ext4 with a standard deviation of 236,68 compared to SDFS standard deviation of 51,93. ZFS deduplication shows an increase in bandwidth of 1,79 % compared to ZFS. The standard deviation for ZFS deduplication is higher compared to ZFS. ZFS deduplication has a standard deviation of 560,87, compared to ZFS standard deviation of 492,82. Due to the high standard deviation and similarity of bandwidth for both ZFS implementations, no statistical difference can be proven. Comparing the two deduplication implementations shows that SDFS has a 68,75 % decrease in bandwidth compared to ZFS deduplication.

Figure 14 shows bandwidth usage during random write operations.

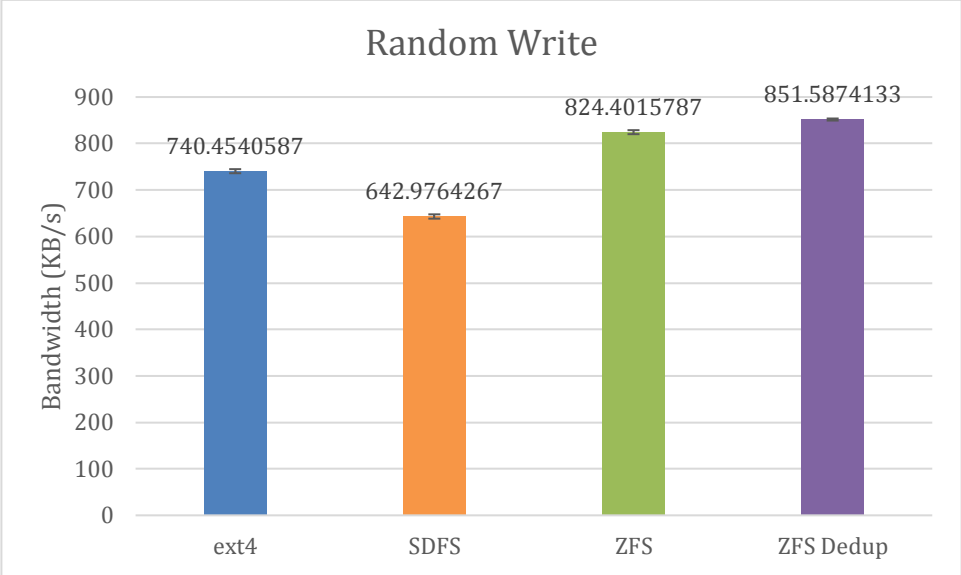



**Figure 14** Random Write

Figure 14 shows that SDFS has a lower bandwidth compared to ext4. The decrease in bandwidth between ext4 and SDFS is 13,16 %. Both ext4 and SDFS have similar standard deviations, ext4 has a standard deviation 8,67 while SDFS has a slightly larger standard deviation of 9,25. ZFS deduplication has an increase in bandwidth of 3,30 % compared to ZFS. ZFS deduplication also has a lower variation of test results with a standard deviation of 4,41, compared to ZFS standard deviation of 8,59. Comparing SDFS to ZFS deduplication show that SDFS has a decrease of 24,50 % over ZFS deduplication.

Figure 15 shows the measured disk latency of each file systems and deduplication implementation.

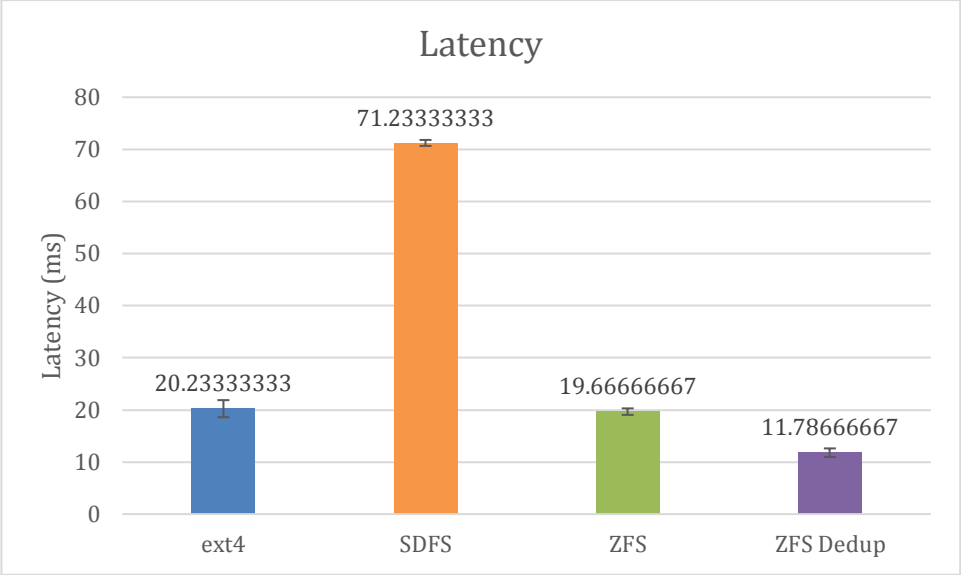



**Figure 15** Latency

Figure 15 shows that ext4 has a much lower average latency when compared to SDFS. The increase in latency for SDFS is 252,10 %. SDFS however shows a lower variance in test results with a standard deviation of 1,15 compared to ext4 which have a standard deviation of 3,28. ZFS deduplication shows a decrease in latency of 40,11 % compared to ZFS. ZFS has a standard deviation of 1,56 and ZFS deduplication has a slightly higher standard deviation of 1,65. Comparing the two deduplication implementations show that SDFS has a 504,160 % increase in latency compared to ZFS deduplication.

## 6.4 Disk performance validation

The disk performance validation results compare test results performed directly on the storage server to the tests performed from the virtual machines. The full test results are included in Appendix A.

When comparing test results from sequential read operations, ext4 and SDFS show a similar decrease in performance when comparing tests from the storage server to tests from the virtual machines. The decrease is 30,41 % for ext4 and 38,66 % for SDFS. ZFS shows a much larger decrease in performance, both with and without deduplication. ZFS shows a decrease of 47,01 % while ZFS deduplication shows a decrease of 50,69 %. The variance in test results is small for all file systems and deduplication implementations, which show that the difference between the tests is statistically significant.

Results from random read operations show that the decrease in performance is generally lower between the storage server and the virtual machines. Ext4 shows an 11,50 % decrease in bandwidth which is lower than results from sequential read operations. SDFS has a larger decrease in bandwidth with a difference of 58,79 %. Both ZFS implementations show a smaller variance with a decrease of 34,64 % and 22,19 %, for ZFS and ZFS deduplication respectively. The variance in test results is small in all tests, therefore showing that the difference between the tests is statistically significant.

Comparing results from sequential write operations show that the decrease in bandwidth for ext4 is 64,61 %. The decrease for ext4 is therefore larger than the decrease in bandwidth for SDFS, which is 49,71 %. The two ZFS implementations show a smaller decrease in bandwidth than ext4 and SDFS. ZFS has a decrease in bandwidth of 30,42 % while ZFS deduplication has a decrease of 25,36 %. The standard variance for all tests are relativity low, therefore showing a statistical difference between all tests.

Results gathered during random write operations show that ext4 has a decrease in bandwidth of 26,76 %, SDFS has a lower decrease of 16,89 %. Both ZFS implementations show a similar decrease in bandwidth to ext4 and SDFS. ZFS has a decrease in bandwidth of 26,97 % while ZFS deduplication shows a decrease of 18,95 %. The standard deviation for all tests show that the variance is small and that statistical difference between the tests can be shown.

Latency tests show a large different between the different file systems and deduplication implementations. The increase in latency when comparing ext4 tests is 56,44 % while SDFS shows an increase of 2,05 %. The two ZFS implementations show a similar latency increase pattern with the non-deduplicated ZFS having an increase of 48,99 %. While ZFS deduplication has a smaller increase in bandwidth of 23,94 %. The variance for the latency tests is overall low, meaning that statistical difference between all tests can be shown.

# 7 Conclusions

This section contains the conclusions that can be drawn from the data gathered during the experiment.

## 7.1 Storage utilisation

From the gathered data, it is clear that deduplication using both SDFS and ZFS deduplication can decrease the used storage space significantly. When using SDFS the decrease in storage utilisation is 72,5 % and ZFS deduplication has a decrease of 73,65 %. The results show that ZFS deduplication is slightly more effective in minimising the storage space needed to store the pool of virtual machines. The conclusion is that both deduplication implementation provides a large decrease in used storage space with ZFS deduplication being slightly more effective. The results answer RQ1 and match the expected results for storage savings.

## 7.2 Resource utilisation

The data shows that SDFS has a much larger increase in RAM utilisation compared to both ZFS implementations and ext4. SDFS has a 431,46 % increase over its base file system ext4. ZFS deduplication in this experiment is much more effective in RAM utilisation with an increase of only 12,13 % over its base file systems ZFS. The result is that ZFS deduplication uses 71,41 % less RAM than SDFS and that deduplication on both tested implementations have an impact on RAM utilisation.

Data gathered on CPU utilisation shows that in the case of read operations, that SDFS uses significantly more CPU than both ZFS implementations and ext4. SDFS shows an increase of 168.75 % compared to ext4, while ZFS deduplication shows an increase of 102,92 % over its base file system ZFS. The result is that ZFS is more effective in CPU utilisation during read operations than SDFS, but that both deduplication implementations have shown an increase in CPU utilisation during read operations compared to their respective non-deduplication implementation.

Comparing CPU utilisation during read operations for the full duration to the same results after 300 seconds shows that there is no statistical difference between ext4 and ZFS deduplication. Both ZFS and SDFS show significantly lower average CPU utilisation, as well as lower standard deviations.

CPU utilisation during write operations show different utilisation patterns compared to read operations. SDFS has a decrease in CPU utilisation of 39,60 %, which indicated that SDFS is more effective in terms of CPU utilisation than the non-deduplicated file system ext4. ZFS deduplication shows an increase of 8.05 % over ZFS, which is a much smaller increase than during read operations. Due to the large standard deviation of the test results for both ZFS and ZFS deduplication, no statistical difference can be shown.

The conclusion, which answers RQ2, is that both deduplication implementations resulted in higher RAM usage. SDFS has a significantly higher increase in RAM usage than ZFS deduplication. The results also show that deduplication resulted in higher CPU utilisation during read operations, with SDFS having a higher CPU utilisation impact than ZFS deduplication. Results from write operations show that SDFS has a lower CPU usage

compared to non-deduplicated ext4, while ZFS deduplication show no statistical difference in CPU usage. The expected results of higher the resource utilisation when using deduplication are matched by results from ZFS deduplication. But only partially matched by results for SDFS, since write operations use less CPU.

## 7.3 Disk performance

The data on disk performance shows that in all tested cases that SDFS has a significant performance deficit compared to its non-deduplicated file system ext4. SDFS has during read operations a performance decrease of 65,61 % and 84,64 %, for sequential and random operations respectively. The performance deficit is smaller but still significant for sequential write operations where the decrease of SDFS is 44,22 % compared to ext4. The smallest performance impact for SDFS is on random write operations where the decrease is 13,16 %. SDFS also has a large increase in latency compared to ext4, with an increase of 252,10 %. The results are that applying deduplication with SDFS on ext4 results in large performance impacts, especially during read and sequential write operations, but also in latency.

ZFS deduplication unlike SDFS shows few negative performance impacts and is generally faster than ZFS. During sequential read operations the performance decrease is only 4,74 % compared to ZFS and for random read operations, ZFS deduplication has an increase in performance of 23.62 %. For write operations, ZFS deduplication is faster in both sequential and random operations, with sequential operations having an increase of 1,79 % and random operations having an increase of 3,30 %. ZFS deduplication also shows a decrease in latency of 40,11 % compared to ZFS.

The disk performance results show that using deduplication with SDFS result in a decrease of performance in all tested cases with the largest performance impacts being in read operations. While ZFS deduplication shows a small increase in performance in all cases except for sequential read operations. These results answer RQ3 and partially match the expected results. SDFS matches the expected results since the performance is slower when using deduplication, but ZFS deduplication generally performs better which was not expected.

## 7.4 Disk performance validation

Comparing disk performance between tests run on the storage server and on the virtual machines show a decrease in performance on all tests. During both sequential and random read operations the performance decrease is larger on SDFS than on ext4. The decrease in sequential read operations is 30,41 % on ext4 and 38,66 % on SDFS, while on random read operations the decrease is 11,50 % on ext4 and 58,79 % on SDFS. Showing that the difference is much larger on random read operations than on sequential read operations. For sequential write operations, the decrease in performance is larger on ext4 than on SDFS. The decrease in performance on sequential write operations is on ext4 64,61 % and SDFS has a decrease of 49,61 %. For random write operations, the decrease on ext4 is 26,76 % while SDFS has a decrease of 16,89 %. The data from the latency test shows that ext4 has a larger decrease in performance compared to SDFS. Ext4 has an increase of 56,44 % in latency compared to SDFS increase of 2,05 %.

In both read tests, ZFS and ZFS deduplication show a larger decrease in performance compared to ext4 and SDFS. In the sequential read tests, ZFS has a decrease in performance

of 47,01 % and ZFS deduplication has a decrease of 50,69 %. Both ZFS and ZFS deduplication therefore have a larger performance decrease in sequential read tests compared to ext4 and SDFS. When comparing random read tests ZFS has a performance decrease of 34,64 % and ZFS deduplication a decrease of 22,19 %. Both ZFS implementations therefore have a larger decrease than ext4 but not as large as SDFS. When comparing sequential write operations, ZFS has a decrease of 30,42 % and ZFS deduplication a decrease of 25,36 %. Both of which are smaller than ext4 and SDFS, showing that ZFS and ZFS deduplication have a smaller performance impact than ext4 and SDFS. In random write operations, ZFS has a performance decrease of 26,97 %, which is larger than ZFS deduplication, that has a decrease of 18,95 %. The result is that both ZFS implementations have a larger performance decrease on random write operations compared to both ext4 and SDFS. The latency tests show that ZFS has a performance decrease of 48,99 % and ZFS deduplication a decrease of 23,94 %. This shows that ext4 has larger a decrease than ZFS, but that SDFS has a smaller decrease than ZFS deduplication.

The conclusion is that the performance for all tests on all file systems and deduplication implementations have decreased when comparing local tests performed on the storage server to tests on remote storage trough the virtual machines. On sequential read and random write operations, ext4 and SDFS performed better than ZFS and ZFS deduplication. But for sequential write operations, both ext4 and SDFS have a higher decrease in performances compared to the two ZFS implementations. For random read operations, ext4 performed better than the two ZFS implementations, but SDFS has the largest performance decrease then all the other three implementations.

# 8 Discussion

This section focuses on discussing and reflecting on the results, the performed experiment and comparing the gathered results with related work.

## 8.1 Results discussion

The test results from the storage savings test show that ZFS deduplication is marginally more effective than SDFS, one possible reason could be that the block size used by ZFS deduplication is smaller than the block size used by SDFS. A smaller block size could result in a more effective deduplication, therefore the storage savings could be larger (Jin & Miller, 2009).

RAM usage for SDFS is significantly larger than the RAM usage of ZFS deduplication. The reason is unknown but it is possible that this is due to how SDFS is implemented. SDFS is implemented on top of an existing file system, which may result in that more data about each chunk and chunk storage location needs to be stored compared to ZFS deduplication. ZFS deduplication is implemented in an existing file system and may therefore have a more effective data storage.

The CPU usage during read operations shows that there is a difference in CPU usage for the first 300 seconds and the last 300 seconds of the tests. SDFS shows an initial high CPU usage which has a decline to about 300 seconds where it levels out, while ZFS has a sudden decrease after about 256 seconds. A possible cause of these differences is that the file used for read operations is being cached. The lower CPU usage would therefore be the result of the lower amount of disk operations being used to read the test file. Ext4 and ZFS deduplication show a lower variance through the full test, which might be an indication that no caching is performed by these implementations. Future work is needed to further explore the differences in caching in order to draw certain conclusions.

SDFS shows a decrease in CPU usage compared to ext4 during write operations, a possible cause is that the random write bandwidth is very low for SDFS and therefore requiring a lower amount of disk operations. The result of fewer disk operations is a possible cause for the low CPU utilisation. The two ZFS implementations have similar CPU usage and also similar random write bandwidth.

The test results from disk performance tests show that SDFS has an overall lower bandwidth in all tested cases and a higher latency compared to ext4. One possible cause of the results could be because of how SDFS is implemented. SDFS is implemented on top of an existing file system, therefore separating deduplication from the storage of data. It is possible that this type of implementation causes a larger overhead since the data first needs to be processed and deduplicated by SDFS, to then be stored separately by another file system. The additional overhead is likely to cause slower performance and higher latency. ZFS deduplication is implemented in the existing file system ZFS and is therefore less likely to have the same overhead as SDFS. Unlike SDFS, ZFS deduplication shows generally higher performance compared to its non-deduplicated file system. One possible cause is that using ZFS deduplication result in fewer disk seek operations since the location of disk chunks can be read from the deduplication hash table, since the deduplication hash table resides in RAM

the operation should be faster than a disk operation. The less time used to locate chunks could result in higher performance and lower latency.

Results from the disk performance validation test show an overall decrease in performance when comparing tests from the local storage server to tests from the virtual machines. Some performance decrease is expected due to the network overhead. Another possible cause to the larger performance decrease could be the low MTU size forced by hardware limitations. The low MTU size limits the amount of data that can be sent per network packet, resulting in lower network throughput and thereby lower bandwidth.

## 8.2 Experiment discussion

The goal of this study is to evaluate the potential performance impacts that can be caused by implementing deduplication on virtual machine storage. When designing the experiment that is used to perform the study, previous research is studied to identify performance variables that can be affected by deduplication. While these performance variables are tested in this study, more could be affected than measured. An example is other types of disk operations that could be affected by deduplication, such as re-write and re-read. There are also other resource utilisation variables that could potentially be affected, such as higher RAM usage during disk operations.

The handling of the validity threat of low statistical power could be handled further by increasing the number of runs per disk test and using longer time periods for utilisation tests. By using extended tests, the results would have higher statistical power.

The test results show low performance values during disk performance tests. The tests also show uncommon performance patterns, for example, random read test being slower than random write. If the experiment is to be performed again and if more time is available, the performance test could be extended by possibly testing other benchmark tools and testing different parameters for each tool. Further performance optimisations of the file systems and deduplication implementations could potentially also be possible.

Comparing test performed locally on the storage server and from the virtual machines, the performance in many cases decreases significantly. The results therefore indicate that the used default settings might not be optimal to achieve the highest possible performance, therefore if more time would have been available different variables could be modified to achieve higher performance of shared storage. Furthermore, by using different hardware with larger MTU size support the performance could possibly be explored further.

External validity is handled to a certain degree by using common file systems and virtualization techniques. But to provide a higher degree of external validity the experiment could be performed with more than two deduplication implementations, the virtual machine pool could also be larger and contain a realistic amount of user data as well as a larger variation in installed packets. It would also be interesting to test deduplication effectivity using virtual machines with different operating systems and operating system versions.

## 8.3 Related work

According to research by Jin & Miller (2009) the storage savings when using virtual machines with the same operating system can be as high as 80 %. The results of this study

show storage savings of 72,5 % and 73,65 %, indicating that the storage savings shown in this study are comparable to previous research.

Mandagere et al. (2008) shows that the CPU utilisation when performing deduplication can be as high as 75 % depending on the block size used. The smallest block size tested was 16k and showed a CPU utilisation of about 30 %. The CPU utilisation measured during deduplication (write operations) in this study show CPU utilisations of between 12,25 % and 22,52 %. This indicates that deduplication implementations used in this study are more effective regarding CPU utilisation.

Previous research by Ng et al. (2011) shows that their presented deduplication implementation LiveDFS has a sequential read bandwidth of about 130 MB/s, which can be compared to the results of this study that show sequential read bandwidth of 30,86 MB/s for SDFS and 72,43 MB/s for ZFS deduplication. Ng et al. (2011) also present sequential write bandwidth of between 60 MB/s and 90 MB/s, depending on enabled features in their file system. The results of this study show that SDFS has a sequential write bandwidth of 6,13 MB/s and ZFS deduplication 19,61 MB/s. Comparing the results of this study to related works show that the deduplication implementations used in this study are not as effective regarding disk performance.

# 9 Future work

Future work to perform regarding usage of deduplication of virtual machine storage could include performing tests on more file systems and deduplication implementations. This would provide a more accurate picture of the impacts of deduplication and would provide more useful data which could help systems administrators further in decision-making processes.

Future work could also include performing the tests made in this study in a production virtualization environment that contains a large amount of dynamic user data. This would provide disk operations from multiple clients at one time, which could affect the results compared to this study. The external validity would increase in the case of this work.

During this study, observations were made that indicates that optimal performance of the deduplication implementations was not achieved. Future work could be to try and achieve higher performance by identifying and optimising different configurations parameters that affect performance.

# References


Ab Karim, M. B., Luke, J. Y., Wong, M. T., Sian, P. Y., & Hong, O. (2016). Ext4, XFS, BtrFS and ZFS Linux file systems on RADOS Block Devices (RBD): *I/O performance, flexibility and ease of use comparisons. In Open Systems (ICOS), 2016 IEEE Conference on* (pp. 18-23). IEEE.

Corbató, F. J., Merwin-Daggett, M. & Daley, R. C. (1962). An experimental time-sharing system. In *Proceedings of the May 1-3, 1962, spring joint computer conference* (pp. 335-344). ACM. DOI: 10.1145/1460833.1460871

Creasy, R. J. (1981). The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, *25*(5), 483-490.

Paulo, J., Reis, P., Pereira, J., & Sousa, A. (2012). Dedisbench: A benchmark for deduplicated storage systems. *In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 584-601). Springer Berlin Heidelberg.

Jayaram, K. R., Peng, C., Zhang, Z., Kim, M., Chen, H., & Lei, H. (2011). An empirical analysis of similarity in virtual machine images. *In Proceedings of the Middleware 2011 Industry Track Workshop* (p. 6). ACM. DOI: 10.1145/2090181.2090187

Jin, K. & Miller, E. L. (2009). The effectiveness of deduplication on virtual machine disk images. *In Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (p. 7). ACM. DOI: 10.1145/1534530.1534540

Joshi, G., Shingade, S. T. & Shirole, M. R. (2014). Empirical study of virtual disks performance with KVM on DAS. *In Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on* (pp. 1-8). IEEE. DOI: 10.1109/ICAETR.2014.7012890

Mandagere, N., Zhou, P., Smith, M. A. & Uttamchandani, S. (2008). Demystifying data deduplication. *In Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion* (pp. 12-17). ACM. DOI: 10.1145/1462735.1462739

Ng, C. H., Ma, M., Wong, T. Y., Lee, P. P., & Lui, J. (2011). Live deduplication storage of virtual machine images in an open-source cloud. *In Proceedings of the 12th International Middleware Conference* (pp. 80-99). International Federation for Information Processing. ACM.

Scarfone, K., Souppaya, M. & Hoffman, P. (2011). Guide to Security for Full Virtualization Technologies. Gaithersburg: National Institute of Standards and Technology.

Schlosser, D., Duelli, M. & Goll, S. (2011). Performance comparison of hardware virtualization platforms. *In International Conference on Research in Networking* (pp. 393-405). Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-20757-0_31

Schwarzkopf, R., Schmidt, M., Rüdiger, M. & Freisleben, B. (2012). Efficient storage of virtual machine images. *In Proceedings of the 3rd workshop on Scientific Cloud Computing Date* (pp. 51-60). ACM. DOI: 10.1145/2287036.2287046

Tan, Y. & Yan, Z. (2016). Multi-objective Based Performance Evaluation of Deduplication Approaches. *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE. DOI: 10.1109/TrustCom.2016.0186

The Cloud Market. (2017). EC2 Statistics. Retrieved 27 Mars 2017, from http://thecloudmarket.com/stats#/by_platform_definition

Simon, A, (2015). LinuxFilesystemsExplained. Retrieved 14 Mars 2017, from https://help.ubuntu.com/community/LinuxFilesystemsExplained

OpenDedup. (2017). Administration Guide. Retrieved 14 Mars 2017, from http://opendedup.org/odd/administration-guide

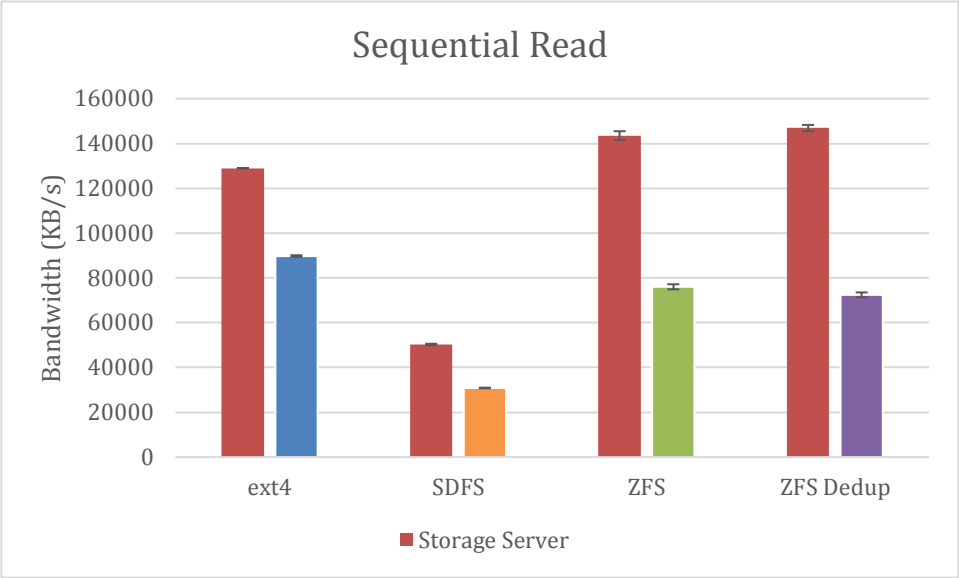
OpenStack. (2016) OpenStack User Survey: A snapshot of OpenStack users' attitudes and deployments. Retrieved 27 Mars 2017, from https://www.openstack.org/assets/survey/April-2016-User-Survey-Report.pdf

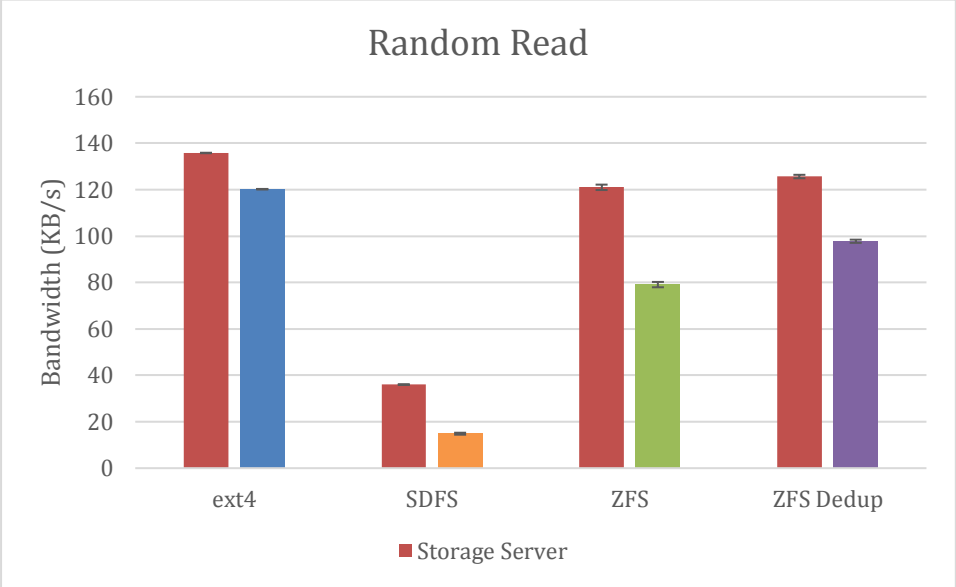
Oracle Corporation. (2017). Oracle VM VirtualBox: User Manual. Retrieved 15 February 2017, from https://www.virtualbox.org/manual/

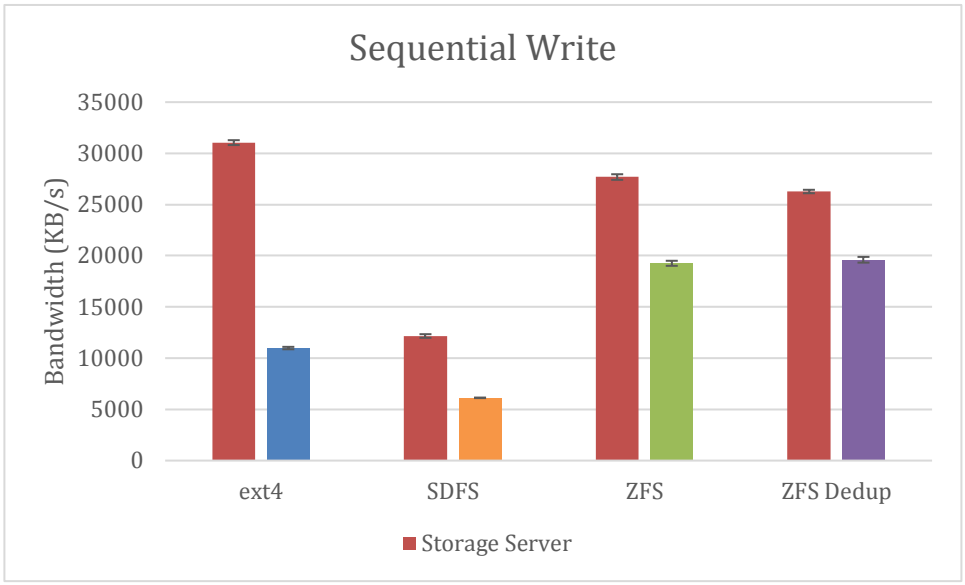
VMware Inc. (2017). vSphere Storage. Retrieved 15 February, 2017, from https://pubs.vmware.com/vsphere-65/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-65-storage-guide.pdf

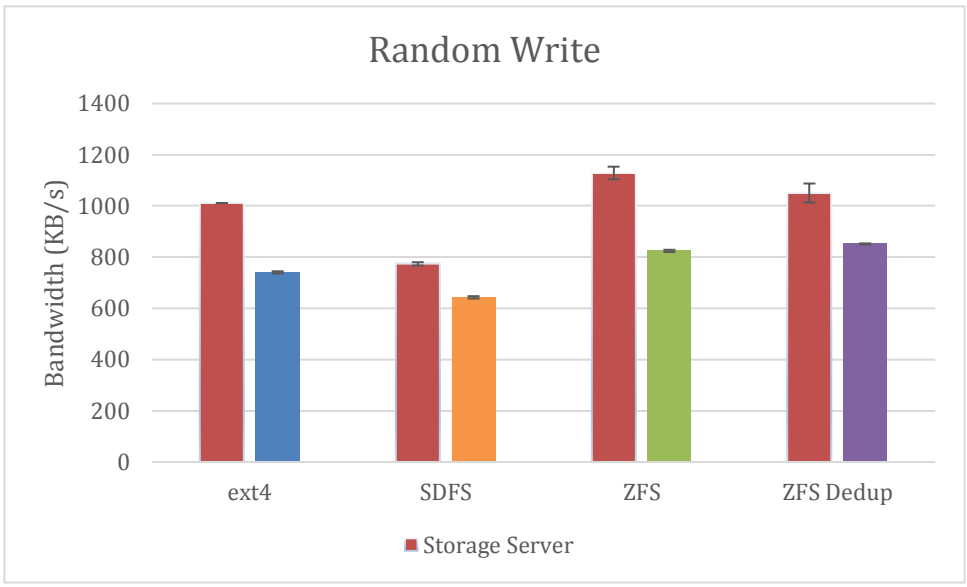
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. (2012). Experimentation in software engineering. Berlin: Springer Science & Business Media.

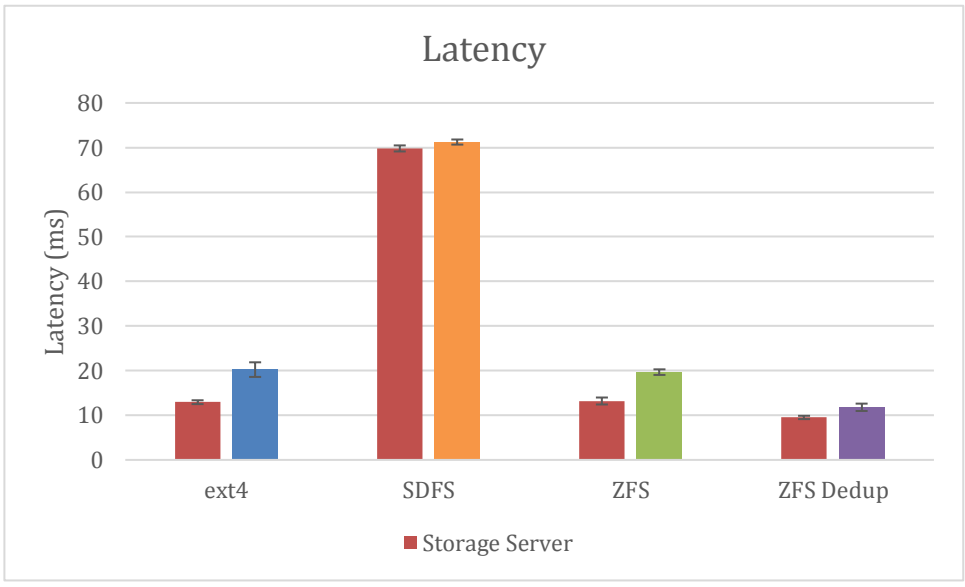
# Appendix A - Disk performance validation




Sequential Write
Bandwidth (KB/s)
35000
30000
25000
20000
15000
10000
5000
0
ext4
SDFS
ZFS
ZFS Dedup
Storage Server


Random Write
Bandwidth (KB/s)
1400
1200
1000
800
600
400
200
0
ext4
SDFS
ZFS
ZFS Dedup
Storage Server


Latency
Latency (ms)
80
70
60
50
40
30
20
10
0
ext4
SDFS
ZFS
ZFS Dedup
Storage Server