TAMPERE UNIVERSITY OF TECHNOLOGY

Mikko Tuominen

ENERGY EFFICIENCY OF DATA STORAGE SYSTEMS IN CLUSTER COMPUTING

Master of Science Thesis

Examiner: Professor Hannu-Matti Järvinen
Examiner and topic approved in the Faculty of Computing and Electrical Engineering Council meeting on 4th of May 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Tuominen, Mikko: Tallennusjärjestelmien energiatehokkuus klusterilaskennassa
Diplomityö, 42 sivua, 5 liitesivua
Kesäkuu 2011
Pääaine: Hajautetut ohjelmistot
Tarkastaja: professori Hannu-Matti Järvinen
Avainsanat: Klusterilaskenta, energiatehokkuus, SSD, levypalvelin, GlusterFS, tiedostojärjestelmä, levyskedulointi

Energiatehokkuus on tärkeä osa-alue minkä tahansa teknologian kehityksessä, eikä klusterilaskenta tee tähän poikkeusta. Energian hinnan noustessa klusterin ylläpidon kustannukset ylittävät helposti sen hankkimiseen tarvittavat kustannukset. Jokainen säästetty euro on samanarvoinen kuin ansaittu euro.

Tämä työ tarkastelee ja vertailee erilaisia laite- ja ohjelmistotason ratkaisuja, joita käytetään klusterilaskennassa datan tallentamiseen. SSD-levyjä ei yleisesti käytetä klustereissa ja yksi tämän työn päämääristä onkin selvittää soveltuuko tämä suhteellisen uusi tekniikka käytettäväksi klustereissa. Tärkein päämäärä on ymmärtää mitkä seikat vaikuttavat klusterin energiatehokkuuteen datan tallennuksen näkökulmasta. Näiden päämäärien saavuttamiseksi klusterin tehokkuutta ja energian kulutusta mitataan ja arvioidaan eri kokoonpanoilla. Tästä saatuja tuloksia voidaan käyttää energiatehokkuuden optimointiin muissa klustereissa.

Työ on jaettu kahteen osaan. Taustatietoja tutkivassa kirjallisuusosassa paneudutaan asioihin, jotka liittyvät energiatehokkuuteen, datan tallennusmalleihin, levyihin, tiedostojärjestelmiin ja levyskedulereihin. Kokeellisessa osassa esitetään testiympäristö sekä raportoidaan ja analysoidaan työn tulokset. Testien suorittamisessa käytetään apuna CERNin CMS-ohjelmistoa ja LHC:n tuottamaa dataa mallintamaan raskasta fysiikkalaskentaa. Testeissä käytetään sekä SSD-levyjä että perinteisiä kiintolevyjä yhdessä kolmen erilaisen datan tallennusmallin kanssa. Tähän kuuluvat hajautettuun tiedostojärjestelmään, levypalvelimeen ja paikalliseen levyyn pohjautuvat ratkaisut.

Tulokset paljastavat, että SSD-levyjen käytöllä ei saavuteta merkittävää etua. Toinen tärkeä tulos on, että huomattava osa klusterin kapasiteetista voi jäädä käyttämättä, mikäli tiedostojärjestelmä ja levyskeduleri eivät ole huolella valittuja. Työn johtopäätös on, että vaikka mitään estettä SSD-levyjen käytölle ei ole, kun otetaan huomioon sekä levyjen hinta että kapasiteetti, ei niiden käyttö ole perusteltua. Kun SSD-levyjen kehitys etenee, on syytä arvioida tilanne uudelleen. Mikäli hinnat laskevat ja tallennuskapasiteetti kasvaa, voi mekaaninen kiintolevy siirtyä historiaan.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Tuominen, Mikko: Energy Efficiency of Data Storage Systems in Cluster Computing
Master of Science Thesis, 42 pages, 5 Appendix pages
June 2011
Major: Distributed software
Examiner: Professor Hannu-Matti Järvinen
Keywords: Cluster computing, energy efficiency, SSD, file server, GlusterFS, file system, I/O scheduling

Energy efficiency is an important part of the development of any technology. Cluster computing is no exception. As the energy prices rise, the costs of running a cluster can easily overcome the costs of buying one. A euro saved is a euro earned.

This thesis examines and compares different hardware level approaches and software level configurations used in clusters to storage data. Solid state drives are not commonly used in clusters and one of the goals of this thesis is to study whether or not this relatively new technology is suitable to be used in clusters. The main goal is to understand what affects to the energy efficiency of a cluster from a data storage point of view. To reach these goals, the performance and energy consumption of a cluster, with different system configurations, is measured and analysed. These results can further be used to optimise existing clusters.

The thesis is divided into two parts. In the literature study part, issues related to energy efficiency, data storage models, block devices, file systems and I/O schedulers are studied. In the experimental part, the test environment is introduced in detail and the results are reported and analysed. The tests are conducted using the CMS software with real LHC data to simulate heavy physics computing. During these tests, both hard disk and solid state drives are used with three different data storage schemes; a distributed approach with GlusterFS (a distributed file system) on compute nodes, a centralised approach with dedicated file server and a local approach with drives in the compute nodes of the cluster.

The test results reveal that no significant gain is achieved by using solid state drives. Another key result is that a cluster can suffer from a major performance loss if the file system and I/O scheduler is not properly selected. The conclusion of this thesis is, that although there is no fundamental reason why solid state drives should not be used in clusters, considering the multifold price and low capacity compared to hard disk drives, it is not justifiable. As the development of solid state drives progress, a new study is in order. If the prices decline and storage capacity increases, solid state drives could abolish mechanical drives.

ACKNOWLEDGEMENTS

I was fortunate to have an opportunity to be a member of one of the greatest research community in Europe while writing this thesis. The eight months I lived in France was one of the best time of my life. I was able to learn a lot from my colleagues, both off and on duty.

First of all, I want to thank my instructor at CERN, the GreenIT Project Leader Tapio Niemi who guided me through this process, offered his expertise and help, and was always ready for discussions about my work.

My warmest thanks go to Professor Hannu-Matti Järvinen from the Department of Information Technology who helped me to finish this thesis.

I want to thank my co-workers Kalle Happonen and Jukka Kommeri for their expertise and technical support on Linux and cluster administration. I also want to thank Lauri Wendland who was kind enough to share his time and knowledge on physics computing at CERN.

A special mention goes to my family and friends for their love and support, and of course, to my dear Riitta who has been there for me all these years.

Tampere, May 19th 2011

Mikko Tuominen

mikko.r.tuominen@gmail.com
+358 400 121 332

CONTENTS

ABBREVIATIONS

AS	Anticipatory Scheduling
CFQ	Completely Fair Queuing
CMS	Compact Myon Solenoid. Large general-purpose particle physics detector and the name of one of the LHC experiment.
CMSSW	CMS software. A physics software toolkit used to compute data from the CMS detector of the LHC.
DFS	Distributed File System
FIFO	First In First Out
GlusterFS	Distributed File System software. Developed by Gluster inc. GlusterFS is free software, licensed under GNU AGPL v3 license.
HDD	Hard Disk Drive
LAN	Local Area Network
LHC	Large Hadron Collider. A particle accelerator in CERN.
NFS	Network File System. Name of a network file system implementation.
RAID	Redundant Array of Independent Disks
SSD	Solid State Drive
SSTF	Shortest Seek Time First
WAN	Wide Area Network

1. INTRODUCTION

An energy efficient computing cluster can help you to save two types of green; money green and the nature type of green. Being energy efficient on any field is good for the environment, yet alone in the line of business where thousands and thousands of machines run for 24/7. Also the electricity bill plays a major role in the financial side of running a computing facility. Environment aware image is to be considered also good publicity to any company or corporation. Whatever is your take on the subject, the fact is that one way to look at green IT, is to make more with less. More data, more bandwidth, more calculations and more utilisation with less energy, less heat generated, less wasted resources and with less money spent. Several studies, such as Tsirogiannis et al. [36] and Niemi et al. [27], have concluded that the best performing system is very likely also the most energy efficient. Energy efficiency equals savings in operational costs and improved performance equals savings in hardware costs.

Cluster computing was designed to carry out calculations, which otherwise would be too time consuming and practically impossible to perform with a single computer. Typical I/O intensive computing job performs a relatively small set of operations to a very large set of data. It is also common for such applications to have very large file sizes, from tens of megabytes of data up to a terabyte scale. I/O easily becomes the bottleneck of such a system.

Cluster computing is a tool. It is a tool for thousands of scientists around the world. Like any other tool, it needs to be efficient and reliable. Although any computing cluster can have have seemingly vast resources, these resources are constantly at use as computing clusters are usually very highly utilised. It is common for these clusters to always have a computing job in the queue waiting to get some runtime. It is obvious that any performance gain is definitely considered as a positive thing. However, the components used in computing clusters needs also be reliable and new technologies are not adopted in a hurry. Lots of software used in clusters have already had many updates and newer versions, but cluster admins may tend to stick with software that is already proven to be working. This calls for careful understanding of new technologies and thorough testing.

Solid state drives are alleged to be superior to hard disk drives. They consume less electricity, have greater bandwidth, can serve more requests per second and

do not suffer from any mechanical delays. They are even resistant to vibrations. Solid state drives are relatively new technology, which has quickly led to giant leaps forward in some of the areas of block device developement. This quick development has even raised some problems as some of the software is not keeping up with them. For example, hard disk drives can only serve a few hundred I/O operations per second, which is limited by the mechanical nature of the drive. Solid state drives can serve tens of thousands as they are based on a transistor technology and have no moving parts to slow them down. Unfortunately, solid state drives are also more expensive and have notably less storage capacity than existing hard disk drives. On the other hand, the development of solid state drives in these areas has also been quite rapid and they are catching up. There have been studies both for and against, whether solid state drives ever overtake hard disk drives in every aspect.

However, solid state drives are nothing like hard disk drives. The whole technology behind them is fundamentally different. The problem is that the current computer systems are designed with hard disk drives in mind. These two drive types cannot be compared without fully understanding their differencies and how it might affect to the system as a whole. In fact, some of the currently used software components can even degrade the performance of the solid state drives. One such example is I/O scheduling, which is designed and implemented to fix some of the weaknesses of the hard disk drives. I/O schedulers are important piece of software and they really improve the performance of hard disk drives, but they can also really hurt solid state drives.

The purpose of this study is to find out whether or not solid state drives are suitable to be used in cluster computing. Assuming they are, it is interesting to find out if solid state drives are better than hard disk drives. One big question is also, how to measure this. One metric is clearly not sufficient and excessive optimisation for one type of workload may not be any use for different kind of workload.

The goal of this study is to understand the meaning of solid state drives to cluster computing. To understand what affects to the energy efficiency of a computing cluster from a data storage point of view. What is important and what is not. What is the right target for optimisation. This is evaluated from the perspect of energy efficiency without degrading performance.

The method of exploring these challenges is quite pragmatic: building and running a test cluster with different kind of hardware setups and software configurations. The test cluster exploits the CMSSW, a physics software toolkit used in CERN to compute the data generated by the LHC. A suitable test job is constructed for the test cluster. This test job is ran against different set of system configurations and each test run is measured. These results are analysed in an attempt to find the best possible configuration.

This study does not elaborate or evaluate the total life span of either drive type. The ecological impact of manufacturing either a solid state drive or a hard disk drive is left outside of the scope of this study. Another big issue that is not included, is the economical aspect. Solid state drives are at the moment a lot more expensive than hard disk drives. Some basic numbers are provided, such as prices of these devices, but no comprehensive assessment is provided.

This thesis is constructed as follows. Chapter 2 discusses of the background of energy efficiency as a concept. Chapter 3 discusses of both drive types (block devices) and different data storaging schemes. Also software components attached to data storaging are covered in this chapter. Chapter 4 summarises the conclusions from previous studies. The information in these Chapters (ch. 2 – ch. 4) is fully derived from different sources and produced by other people. Our role has been to gather these together to provide a sound base to evaluate and analyse the following results.

Chapter 5 discusses of the test methods. The test cluster is described in detail, for both hardware and software. The actual practical side of conducting the tests is also represented thoroughly. In the next chapter, Chapter 6, the test results are illustraded and analysed. The chapter is divided into multiple sections and every section discusses a distinct configuration in detail. As the drive type is so essential in this study, both drive types are treated separately within these sections. Finally, in Chapter 7, the most important results of this study are summarised.

2. ENERGY EFFICIENCY OPTIMIZATION OF CLUSTER COMPUTING

Grid computing was designed to carry out calculations, which otherwise would be too time consuming to perform with a single computer. Before further discussing how to improve the energy efficiency of a cluster, and especially data storage, first a little introduction of what affects to the power consumption.

First, there is the sheer size of the data storage. Data storage capacity and usage efficiency have a direct impact on power consumption. Usage efficiency can be understood as a ratio between the total data storage capacity and the utilised portion of it. The more data is stored or the more disk space is unefficiently allocated, the more disks are needed and the more energy is consumed. Second, data transfer rate (I/O bandwidth) and access time also have an effect. The easiest way to improve both is to use disks with higher rotational speed. Higher I/O bandwidth or lower access times requires thus more power than the less time critical counterparts. Third and last, there is data availability and system reliability. Replicating or backing up a system requires additional components and appliances, which of course requires additional power. Improving usage efficiency, minimising the energy consumption of current components or applying new technologies, such as solid state drives, are all potential approaches to a more energy efficient data storage solutions. [30]

According to Tsirogiannis et al. [36] the most energy efficient system configuration is also the highest performing one. This is quite intuitive in a cluster environment, where high utilisation rates are expected and average job throughput is what matters. Niemi et al. [27] conducted a study, which had complementary results indicating that optimising system throughput also improves energy efficiency. They found that it is more efficient to run more than one simultaneous job per processor core on a compute node.

Measurement of energy efficiency is not a simple task. A single metric is not enough to create the full picture. For example, just looking at achieved storage space per unit of energy, i.e. GB/Wh, is not sufficient. Workload characteristics needs also to be taken into equation. Storage devices differ remarkably by performance metrics such as throughput (MB/s), access time or IOPS (I/O operations per second). Also, all distinct sources of energy consumption may not be easily quantifiable or measurable. [30]

Anderson and Tucek [10] also acknowledge the difficulty of measuring and comparing the energy efficiency. They propose a scheme to calculate energy efficiency in proportion to alternative implementations. Conveniently, this approach also diminish the effect that many components consume constant power regardless of utilisation, which better helps to evaluate the gains. They also remind that micro optimisation is feeble if orders of magnitude increases can be obtained with alternative solutions.

One common metric to measure the energy efficiency on a data center level in a spirit of green IT is the *Power Usage Effectiveness (PUE)*. PUE is the ratio between the power delivered to the data center and the power actually used by IT equipment. Difference can be explained by noticing that some power is always needed for cooling, lighting, etc. and also some is lost due power distribution process, e.g. with UPS appliances. Historically PUE has been as high as 2.25 to 3.0, which translates into 33-44% of utilisation rate. Today, PUE of 1.25 can be achieved by using modern best practices, where 80% of total facility power is delivered to IT equipment. This cascade effect of power consumption is illustraded in Figure 2.1. Facebook engineers have reported PUE as low as 1.07 at full load on their state-of-the-art data center, where energy efficiency was an important design goal [5]. [38]

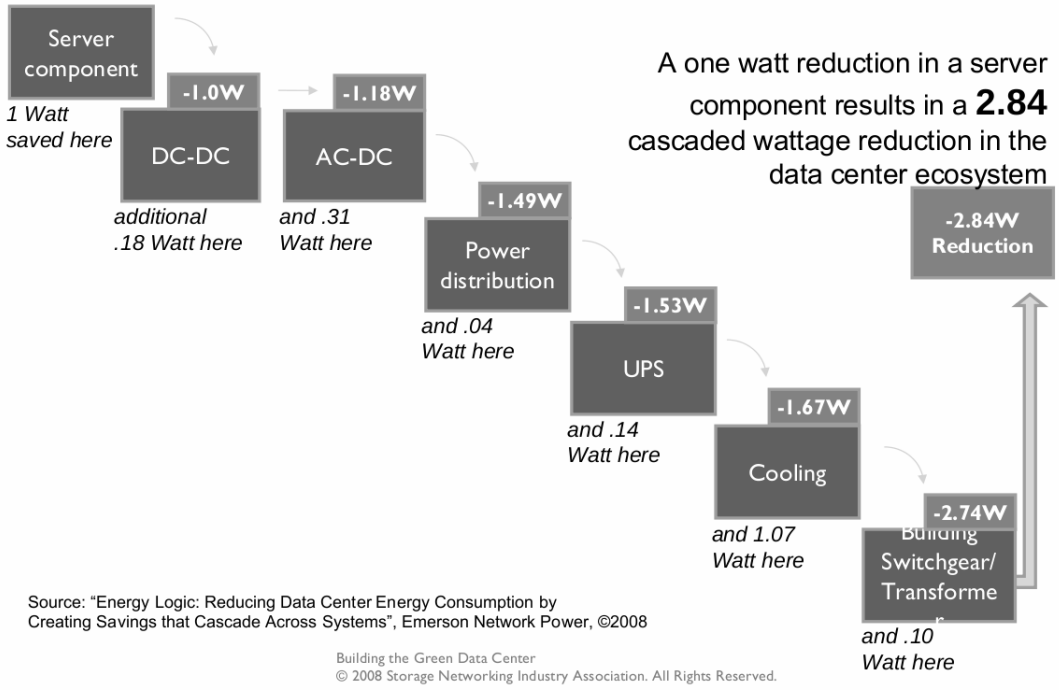

Figure 2.1: The Power Cascade Model. Source: SNIA [13]

Solid state drives (SSD) are known to consume less energy than hard disk drives (HDD) due their non-mechanical design. What makes SSDs even more appealing is that they exhibit perfect energy proportionality, which means the energy consump-

tion is dependent on the load in a linear fashion [36]. Narayanan et al. [26] criticises recent studies on SSDs being only interested on performance but not providing any cost based analysis. They are confident that SSDs will not achieve the capacity per dollar of HDDs. Totally opposite estimation is presented by Schmidt et al. [32], who argued that annual growth rates in performance of SSD development and declining of prices indicate SSDs outperforming HDDs in all aspects in the near future. They also point out that rising energy prices favor this development in a situation where operational costs dominate hardware costs.

3. DATA STORAGING

This chapter is divided into three section. The first section, Disk types, introduces the physical devices, where the bits are stored and the characteristics of these devices. The second section, Data Storage Architectures, discusses several different concepts and models needed to store data in a cluster environment. The third and last section, Hard Disk and Solid State Drives in Linux, discusses the software needed to make all this happen from an operating system point of view. All these can have an effect on the overall performance and energy efficiency of a cluster.

3.1 Disk types

Both hard disk and solid state drives are used as block devices. A block device is a storage component that offers an interface for a block level operations. A block is an abstraction between block number and the physical representation of data on the device. Operating system uses a *Logical Block Address (LBA)* as a parameter for targeting data in I/O operations. The block size of the device needs to be a multiple of the sector size of the HDD. This is discussed more thoroughly in Sections 3.1.1 and 3.3.

3.1.1 Hard Disk Drive

A *Hard Disk Drive (HDD)* is composed of multiple magnetic platters, which can be either be read or written by using a disk head. It is common to refer these magnetic platters as *heads* as there is usually only one disk head per platter. The actual disk head is attached to a disk arm, which is used to move the disk head on top of the right track. A *track* is a collection of bits sharing the same radius from the center of the disk, thus forming a circle on the platter. Tracks that share the same radius on different platters are referred to form a *cylinder*. When the disk spins the read head, while positioned stationary, can access the bits on the track in a sequential manner. A track is divided into sectors. A *sector* is the smallest unit of data that can be written to an HDD. Typically, the size of the sector is set by the manufacturer and cannot be changed. A very common sector size in the industry is 512 bytes, which has become the de facto standard. Although recently manufacturers have also introduced HDDs with 4kb sector sizes, but there are some severe compatibility issues with the existing operating systems and low level software. [19], [34]

When HDD receives an I/O request, it transforms the logical block address into a physical address, e.g. to a tuple of cylinder, head and sector numbers. Common consumer grade HDDs and their capacities are represented in Table 3.1. The Velociraptor from Western Digital is a high performance HDD and listed as a point of reference for SSDs.

Table 3.1: HDD capacities. All drives are 3.5" and SATA II. Prices: www.newegg.com (cited 1-Feb-2011)

Manufacturer	Model	Size	Price	GB/$
Hitachi	Deskstar	1 TB	$54.99	18
Samsung	EcoGreen	1 TB	$38.99	26
Seagate	Barracuda	2 TB	$69.99	29
Western Digital	Caviar Green	1 TB	$44.99	22
Western Digital	Caviar Green	2 TB	$99.99	20
Western Digital	Caviar Green	3 TB	$209.99	14
Western Digital	Velociraptor	300 GB	$169.99	1.8

Table 3.2: SSD capacities. All drives are 2.5", MLC and SATA II. Prices: www.newegg.com (cited 1-Feb-2011)

Manufacturer	Model	Size	Price	GB/$
Corsair	Force F40	40 GB	$104.99	0.38
Corsair	Force F120	240 GB	$439.99	0.55
Intel	X25-M	120 GB	$229.99	0.52
Kingston	SSDNow V Series	128 GB	$224.99	0.57
OCZ	Agility 2	160 GB	$299.99	0.53

3.1.2 Solid State Drive

A *Solid State Drive (SSD)* is a mass storage based on NAND flash memory technology. A flash memory consists of readable and reprogrammable transistors, i.e. memory cells. The memory cells preserve their state during a power outage. Data is stored in these cells as voltage levels. If the cell has only two voltage levels and thus represent only one bit, then it is called a *Single Level Cell (SLC)*. If the cell can distinguish four voltage levels (or more) and thus represent two bits (or more), then it is called a *Multi-Level Cell (MLC)*. Flash memory is discussed more thoroughly later, but first a little insight into how an SSD operates.

An SSD is composed of many flash memory chips. Each chip is composed of blocks and each block is composed of pages. These blocks must not be confused

with the block layer blocks discussed earlier. An SSD has three basic operations; read, reprogram (write) and erase. The smallest unit of data for a read or write operation is the *page*, which is typically 512 - 4096 bytes. Only fresh pages can be reprogrammed, so every dirty page must be properly erased before it is reusable. The smallest erasable unit of data is the *block*, which can hold up to 128 pages or 512kb of data. SSDs do not actually have physical sectors, but sometimes a page can be thought as been divided into logical sectors. The reason is that for historical reasons applications are assuming that a block device has 512-byte sectors.

Reading a 4kb page generally takes around tens of microseconds and writing hundreds of microseconds. SLC based devices are generally faster than MLC based. The real penalty comes from erasing a block, which takes 1.5 - 2ms. Thus reading is an order of magnitude faster than writing and two orders of magnitude faster if an erase operation is needed. SSDs (and other flash memories) use a technique called *Flash Translation Layer (FTL)* to overcome this problem.

FTL reduces the effect of time consuming write operations by reserving redundant blocks or pages and hence avoiding the costly erase operation when data is being updated. Downsides are increased overhead for address translation information and increased amount of flash memory operations. Of course this does not solve the problem completely as it only delays the erasing process [22]. This is why a *trim* operation was introduced on SSDs. Its purpose is to erase unused pages on the background. As mentioned earlier, a single page cannot be erased as the smallest erasable unit is the block. So it has to read the data from a block into a cache, erase the whole block and then rewrite the data back into the block. [2]

Where SSDs really excel over HDDs is the random access. Intel X25-M SSD can reach up to 35,000 IOPS (I/O operations per second) for random read and 8,600 IOPS for random write [20]. For comparison, a high-performance HDD "WD Velociraptor" can only perform less than 250 IOPS for both random read and random write. The relatively low IOPS count for HDDs derives from mechanical delays and cannot be significantly improved. SSDs can interleave read and write operations and hence the overall performance of the device can be better than the one of a single flash memory chip. [9].

SSDs have one clear techical weakness compared to HDDs. The write-erase cycle of a memory cell is limited. An SLC can be reprogrammed around 100,000 times and more complicated MLC only 10,000 times before it wears out [31]. This is why modern SSDs comes with something called *Wear leveling*. Wear leveling allows erase counts of blocks to be evenly distributed over the storage media in an attempt to increase the endurance of an SSD. Dynamic wear leveling is an algorithm by which the controller in the SSD recycles blocks with small erase counts in the flash memory [3].

The biggest obstacle SSDs are facing on their way to become widely adopted and respectable alternative to replace HDDs is their price. If comparing SSDs and HDDs just by looking how many gigabytes a dollar can buy, an SSD is approximately 50 times as expensive as HDD as seen from Tables 3.1 (25 GB/$ for HDDs) and 3.2 (0.5 GB/$ for SSDs). However fully electronical SSDs are known to consume less power than partly mechanical HDDs [22].

3.2 Data Storage Schemes

Before discussing more about different options for data storaging schemes, one terminological distinction needs to be pointed out. When using a term *distributed* in the context of data storage, it deliberately refers to a data storage scheme, where the actual data is distributed over multiple machine instances in contrast to client/server model type of distribution. The difference is vague as in a distributed environment the backend implementation is not necessarily transparent to the client. For example, a simple file server can internally exploit other services, which reside on other physical machines. Also many schemes providing distributed data model can have a frontend machine to work as a single entry point and appear to be a single system. In fact, in some cases it can even be technically possible to run such a system on a single machine instance. So basically the definition is based on how the system is meant to be used.

3.2.1 Network-attached Storage (NAS)

A *Network-attached Storage (NAS)* is by definition a data storage accessible over the network. NAS is based on client/server model and provides a file level data access. A NAS appliance is equipped with high-speed network interface and hardware capable of storing vast amounts of data. Terminologically, subtle differencies between a NAS appliance and a conventional file server can be distinguished. NAS is designed for high performance and usually offers customized and pre-configured software and vendor support, which make it easy to deploy and administer. These terms "NAS appliance" and "file server" are used interchangeably as there is little pragmatic difference from the end user point of view. NAS exploits network file system techniques on providing data access for client machines.

A network file system is a file system that is hosted on a remote machine and is accessible over the network. More precisely, it is a protocol to access the remote file system. Network file systems are based on client/server model and are usually stateless, although also stateful network file systems exists. Stateless means that the server provides the file system as is and keeps no record of the state of individual files. This introduces a couple of pros and cons. Stateless design simplifies the

system architecture, but also brings out some synchronisation problems and degrades consistence. This can be a serious problem if a high level of reliability and data integrity is required. Statelessness must be acknowledged and handled at application level. An approach with a centralised server makes it easier to control and backup your files as they all resides in a single system. The clients can mount the network file system like any other conventional file system. The presence of network is hidden and files are transferred to local machine only when needed. Alas, it also makes the server a single point of failure, thus eliminating it as an option for applications of low fault tolerance for accessibility.

The best-known and most common network file system in Linux environment is the *Network File System (NFS)*. The basic idea of NFS is to, from a clients point of view, emulate the behaviour of a local, mounted file system even though the disks are not physically present. NFS is said to be inadequate to scale for systems over 100-1000 nodes, i.e. NFS clients. However, this heavily depends on the use profile of the system and applications characteristics. Read intensive applications have better success than write intensive. After all, NFS is not meant to serve applications, which require high availability. There is also some concerns about the security of the NFS. In a cluster environment this is rarely an issue as clusters tend to reside in a private network, excluding the frontend machine. As a whole, NFS is a popular, widespread, easy to install and widely supported, which makes it the best choice of a data access implementation technique for the NAS subsystem. [14], [33]

3.2.2 Redundant Array of Independent Disks (RAID)

It is common for NAS appliances to exploit the RAID technology. *Redundant Array of Independent Disks (RAID)* is a scheme designed to improve both the reliability and performance of disk access. RAID can be implemented by using either a hardware or software based solution. In a hardware RAID, the server machine is equipped with a specific RAID controller, which receives the I/O requests from the OS and redirects the requests to physical disks. For the OS, only one large block device is visible. With the software-based RAID, the independent disks are visible to the OS and a virtual disk is created upon them. RAID is perceived to be reliable when it comes to storing data, but not necessarily in terms of accessibility. This is especially true when using a NFS protocol, but inaccessibility can also stem from network or power failures [14].

One important technique used by RAID is striping. Striping means that data is sliced into fixed-length chunks of data, which are dispersed over multiple disks. When data is now accessed, the I/O request can be handled parallel by multiple disks and thus improve performance significantly. There are many levels of RAID, each with different characteristics and purposes. RAID-0 level provides only striping, but

no data redundancy. RAID-1 is similar to RAID-0, but it also provides mirroring. Mirroring means that all data is sent to several (usually only two) disks as a safety precaution. This setup provides excellent data reliability and performance at the cost of disk space. RAID-5 provides data parity, which means that for every block striped a parity block is calculated and stored on different disk. If one disk fails, the data in the failed disk can be reconstructed by using the parity information. RAID-6 is similar, but it doubles the amount of parity and hence can tolerate two failed disks. [19]

3.2.3 Distributed File System

As mentioned earlier in Section 3.2, the definition used in this study for distributed data storage refers to **truly** distributed data. In contrast to network file systems, where all data is stored on a single machine, a *distributed file system (DFS)* is running on multiple machine instances. A DFS can have a centralized or decentralized architecture. In a centralized architecture, the client connects to a master server. The master server is responsible for keeping the file system metadata information and redirects the I/O requests to other servers, i.e. data servers. The data server then provides the actual data for the client. This architecture of course makes the master server a single point of failure and easily becomes the bottleneck of such a system. Hence the decentralized DFSs are available. Decentralized architecture can be implemented for example in a peer-to-peer manner, where also the file system metadata is distributed.

The distributed nature of DFSs varies as DFS can reside in a single server rack connected via high-speed LAN or it can be geographically distributed over WAN.

DFS is said to be a *parallel file system* if the data of a single file is distributed to many different servers. This approach have its pros and cons. The performance of reading or writing, especially big files, can be improved significantly as more servers can handle the I/O. On the other hand, as seemingly simple operation as a directory listing can be extremely slow as each server needs to be consulted. DFSs can also be configured to provide data replication to improve accessibility and reliability or data striping (like in RAID systems discussed in Section 3.2.2) to improve performance.

3.3 Hard Disk and Solid State Drives in Linux

To permanently store data, more is needed than just the physical devices. Presence of an operating system is required. Typical data storage scheme can be divided into 4 layers; device layer, kernel layer, file system layer and application layer. Also a block layer can be distinguished between the kernel and file system [28]. The goal is to provide abstraction between the layers, to hide the implementation and technical

details from the user and to provide interfaces to better support interoperability of variety hardware and software components. Optimization of such system can take place on any of these layers. Hard disk and solid state drives can be seen as part of the device layer. Between the physical device and kernel are device drivers, which are part of the kernel. The purpose of the device drivers is to hide the differencies between the vast variety of devices from the kernel. Kernel can now treat any device in the same way through a device driver interface. [19]

The kernel and file system layers are the most interesting ones as they provide the most of easily configurable parameters. In the Linux kernel there is a component called an I/O Scheduler. An operating system does not really require any I/O scheduler to operate as I/O requests can be serviced in a FIFO-like queue manner. This, however, is not the optimal solution in most cases and use of an I/O scheduler can improve the performance of the I/O dramatically. Linux I/O scheduler adds an interface between block layer and the device layer. [28]

When discussing about disk performance, two terms needs to be distinguished; *the response time* and *the access time* of a disk. The response time is the time an I/O requests needs to wait before it is served after it was submitted. The access time of an HDD is a sum of seek time and the actual transmission time. The seek time consists of disk arm transfer and spin delay or rotational latency. Before reading or writing can happen, the disk head needs to be positioned on the beginning of the right sector on disk. The seek time derives from moving the disk arm onto the right track and then waiting the disk to spin so that the correct sector is under the disk head. Transmission time is usually considerably less than seek time. Seek time can be minimised by intelligent positioning of the data onto the disk and also by doing disk read or write request in a best possible order. The former is done by the file system and the latter is called I/O scheduling. [19]

3.3.1 Linux File systems

A file system is an abstraction to map data blocks on a block device, such a HDD or SSD, to meaningful files for the operating system. A file system uses data structures called *inodes* to save information about the files (metadata). An inode contains information about the owner of the file, an access control vector, timestamps for file creation and modification, file size, type of the file (e.g. directory, regular file, link, etc.) and pointers to the actual data on the device. [14]

Usability of a file system can be measured by two common metrics. The first is how efficiently a file system stores files, i.e. how much space is wasted. The second is how efficiently data can be transfered. Using bigger disk blocks can improve the transfer rate as more data is handled at once, but also more disk space is wasted as the last block is left only half-full by average. [19]

Most file systems today are journaling file systems. A journaling file system means that the file system keeps a journal over its writes in case of failures in the writing process. When data is written to a drive, also the metadata information needs to be updated. If the data on the drive and the metadata is out of sync, the file system is said to be corrupted. This can occur for example in case of sudden power outage if only either the data or metadata was updated, but not both. To increase throughput performance drives usually exploit heavily drive caches, which can delay the writes and cause the drive to be out of sync. When the drive gets back online, file system can now go through its journal and replay every step to fix a possible corrupted file system. Without journaling, the whole file system would need a consistency check, which would be drastically slower operation. One of the primary concerns with all filesystems is the speed at which a filesystem can be validated and recovered after corruption.

The most popular file system in Linux during the first decade of 21st century was the Ext3 file system, which is still widely used. Ext3 is the default file system for Rocks cluster software. Ext3 is a journaling file system with maximum volume size of 16 terabytes.

Ext4 is the successor of Ext3 file system. The main motivation developing new version was the 16 TB volume size of Ext3, which stems from 32-bit block numbers. Ext4 assigns 48-bit block numbers and can have volumes up to 1 exabyte for 4kB block size. Ext4 also incorporates scalability and performance enhancements. Ext4 developers provided benchmark results, which shows improvement especially on write I/O requests. The dominating role of Ext3 is acknowledged and upgrade to Ext4 is easy and can be made without losing the data. Ext3 is however perceived as reliable and stable and thus still the file system of choice in many systems, which do not need the support for larger volume sizes. [25]

XFS is a file system created in mid-1990s by Silicon Graphics inc. for their own IRIX OS, but it is later ported to Linux. XFS is also a journaling file system. XFS was designed to be scalable and support large file and directory sizes. The maximum volume size of XFS is 16 exabytes. [35]

3.3.2 GlusterFS

GlusterFS is a distributed file system, developed by Gluster inc. and provided under GNU AGPL v3 licence. GlusterFS architecture is based on peer-to-peer model. Server machines share part of their disk space, called a *brick*, into a collective data pool. These bricks are then used to create virtual data volumes. Data mirroring and data striping are both supported. On the servers, data is stored on local file systems. Actually, what a server shares is a directory and it becomes the root directory for GlusterFS on that server. GlusterFS can allocate all the space left

on that partition. Notice, that any free space can therefore be used either by the GlusterFS or the local file system and therefore the size of GlusterFS volume changes dynamically. Bricks can be added and removed on the fly without disturbing the system. In case of resource removal the data hosted by that node is migrated to another location automatically.

To access these data volumes a client software is needed. The data volume is mounted as part of a local file system with FUSE, *Filesystem in Userspace*. FUSE is an API emulating the behavior of conventional filesystem. Each client has a dummy version of the directory tree of the volume (a filesystem). It contains the metadata (inode) information, but the file size is zero. The actual data is distributed by using the hash calculated from the name and directory path of the file. Each file is now mapped with particular virtual subvolume. These virtual subvolumes are mapped to spesific bricks, i.e. hosts. Using a hash algorithm a file name can now be connected with the host storing the actual file data. When a file is renamed, a pointer is created on new host to redirect to the old location while migrating the data to a new location in the background. When the data transfer is complete, the pointer can now be removed.

Any particular machine can act both as a server and as a client at the same time, i.e. run a server and client software. Other features of Gluster is load balancing, failover recovery, I/O scheduling, caching and quotas. Gluster supports Infiniband and Ethernet (TCP/IP) for networking. [4]

3.3.3 Linux I/O scheduling

An I/O scheduler is a kernel component, which controls the I/O queue and uses a scheduler-specific algorithm to arrange incoming I/O request. When an I/O request is received from a file system through the block layer interface, an I/O scheduler inserts it into the queue and eventually passes it to the disk controller through the device driver interface. [28]

Linux can be said to be optimised for magnetic disks [21]. This section discusses primarily on scheduling HDDs in a Linux environment. Scheduling with SSDs is discussed in Section 4.2.

The current Linux kernel 2.6 has four built-in schedulers. They are called noop, anticipatory, deadline and cfq. The cfq is the current default scheduler. These schedulers are discussed later in detail, but first a little insight on how the disk controller operates.

Disk usage can be optimised by trying to minimise the disk arm transfer, i.e. the seek time. Common algorithms are called FIFO, SSTF, SCAN and C-SCAN. FIFO (First In First Out) does no optimization. SSTF (Shortest Seek Time First) always selects the request which needs the least movement of the disk arm. This can lead in

a situation, especially on a device under heavy loads, where disk head keeps servicing request on a near-by disk blocks and other requests on the outer edges of the disk are faced with long waiting periods or even a starvation. Starvation is a state where a process is waiting for an event that never happens. SCAN just scans the disk from one edge to another, turns back whenever reaches the inner or outer edge of a disk and starts to scan to disk to another direction. SCAN is sometimes referred as the elevator algorithm due its similar operation logic to elevators. C-SCAN is like SCAN, but with a difference that it always scans the disk the same direction. When the arm reaches the edge of a disk, the arm is moved to the opposite edge by one long disk arm transfer. SCAN and C-SCAN are not affected by starvation. Of Linux 2.6 basic schedulers, the noop is based on FIFO and others on SCAN type disk arm transfer algorithm [21]. [19]

The purpose of an I/O scheduler is to improve the performance either by increasing the total bandwidth of the disk or by reducing the access time of individual I/O requests. I/O schedulers use operations called *sorting* and *merging* of I/O request as a tool to minimize the disk seek times. The sorting operation orders requests based on their sector number and inserts incoming requests on their right place on the queue. This way, if the disk is used either with SCAN or C-SCAN based scheduler, no unnecessary disk arm movement has to be made. Merging merely means that requests from different processes to the same data block are recognised and served together. Also it has to be noted that usually read operations are synchronous as a process is waiting them to finish. On the other hand, write operations are usually asynchronic, which means they do not need to be served immediately and can be stored temporarily in a cache. [28]

The most simple I/O scheduler in the default Linux kernel 2.6 is the *noop* I/O scheduler. Noop has minimal overhead and it does only basic merging and sorting of I/O requests. Noop can be a good choice when not using a HDD directly. Either the scheduling is done somewhere else than inside the Linux kernel or a non-mechanical drive, such as an SSD, is used. RAID controllers do their own scheduling and Linux kernel does not have any knowledge of the actual disk states in a RAID array. Therefore Linux kernel can only interfere by doing additional I/O request sorting. Merging of requests is of course desirable. SSDs on the other hand have no moving parts and therefore do not suffer from seek time delays. [28]

The *Deadline* scheduler implements sorting of requests, but also implements an expiry time for each request. The basic idea is aggressive reorder of requests and at the same time to make sure no request has to wait too long to be served. If a request is about to expire before it is served, then deadline starts to serve that request immediately. Read requests are given higher priority than write requests, but nonetheless the deadline mixes write requests with read requests even though

there are more pending read requests. Deadline makes a compromise between high throughput and low I/O request response time. [12, 28]

The *Anticipatory (AS)* I/O scheduler behaves like deadline, but also adds a feature called anticipation. Anticipation derives from a situation called *deceptive idleness*. Deceptive idleness happens when a read operation finishes and the process, which requested it, continues execution only to make a consecutive read request. Normally the disk arm would have already moved into an another position, but now the disk waits for a small period of time if the process wants to make an another I/O request. Naturally this behavior has a negative effect on performance if the process does not make another sequential read request. On some work loads however the overall performance can be improved. There actually are mechanisms, such as cost-benefit analysis or statistic analysis of a probability of such request arriving, which reduces the negative effect of this behavior. AS tries to reduce the read response time for each thread. [28]

Finally, the currently default Linux I/O scheduler, the *Completely Fair Queuing (CFQ)* I/O scheduler. The basic idea of CFQ is to provide fair treatment among different processes and share the I/O bandwidth evenly with the I/O requests. Internally CFQ has many I/O queues, which are operated strict FIFO manner. Each process is given its own queue derived from the process' PID with a hash algorithm. CFQ selects I/O requests from these queues in a round robin manner and moves them into a dispatch queue, which is then sorted and sent out to the device driver. [28]

It is important to note that both AS and CFQ are implemented as Linux kernel components as anticipatory and completely fair queuing are mere scheduling algorithms. Anticipation can be built on any scheduling scheme, not just on deadline. Also compeletely fair queuing does not need to work with a hash algorithm to operate. Any other desired technique can also be used to allocate the I/O queues for processes.

Changing the scheduler in Linux can be done from a command prompt. For example, setting the noop scheduler for the drive in /dev/sdb:

```
echo noop > /sys/block/sdb/queue/scheduler
```

3.3.4 Read-ahead

The *read-ahead* is a mechanism to improve the performance of a block device. The function of the read-ahead is that for every read request served, also an additional amount of data is read from the block device into a cache. It is likely that this data is now requested soon after. When such a request is received, the data can be provided directly from cache and avoid the costly seek operation. The size of additional data block can be configured and is usually expressed in kilobytes.

Changing the read-ahead value in Linux can be done from a command prompt. For example, setting the read ahead to 4kb for the drive in /dev/sdb:

```
echo 4 > /sys/block/sdb/queue/read_ahead_kb
```

4. PREVIOUS WORK

This chapter discusses the previous research work done related to the topic of this thesis. The first section discusses about how SSDs are used in server environment. The second section discusses about research on I/O scheduling.

4.1 SSDs on servers

Lee et al. [23] conducted a study which objective was to identify the areas where SSDs can best be utilized in enterprise database applications. They concluded that using SSDs for transaction log, rollback and temporary data storage is superior over HDDs. They argued that the performance of transactional database applications is more limited by disk latency than disk bandwidth and writing log records is a significant performance bottleneck. They pointed out that the I/O pattern of a workload trace collected from a commercial database server is favorable to SSDs. By implementing these changes on their test server, they managed to transform it from I/O bound to CPU bound. Their tests showed an order of magnitude improvement in transaction throughput and response time. Also, time of processing complicated database operations that required the use of temporary data area dropped to one third.

Schmidt et al. [32] conducted a study on using SSDs in a database environment as an attempt to increase efficiency and reduce costs. They concluded that SSDs outperformed HDDs both in performance and energy efficiency, but the overall cost analysis still favored HDDs. They argued that only suitable usage for SSDs is in high IOPS demand applications, where IOPS/$ or capacity/$ are of minor importance. On their benchmark tests, they used the rate of transactions per second to measure performance. The tests showed that with small database sizes (10 MB), HDDs and SSDs were equal for read-only workloads and HDDs having a slight edge for mixed workload. However, the performace of the HDDs quickly decreased as much as 50% when the size of the database tenfolded (100 MB), while this had little effect on SSDs. Growing the size of the database another ten times bigger (1000 MB); the performance of the HDDs dropped another 25%, while still not affecting the SSDs. All this applied both read-only and mixed workloads.

Narayanan et al. [26] reported similar results in their study, where they performed a cost-benefit analysis for a range of workloads. They used 49 different workload

traces collected from 15 different server machine (storage size ranging from 22 GB to 6.7 TB) to compare SSDs and HDDs. They found out that in all cases, the dominating factor was either the storage capacity or the random-read IOPS requirement. However, due to the low capacity/$ of SSDs, the HDDs always provided the cheapest solution. They presented calculations, that depending on the workload, the capacity per dollar of SSDs needs to improve by a factor of 3-3000. They also argued that energy efficiency is not as important reason to make the transition to using SSDs as low-speed SATA disks are competitive in terms of performance and capacity per watt.

According to Leventhal [24], SSDs should be used as complementary to existing storage system, not as a replacement. He argued that SSDs "falls in a sweet spot" between HDD and RAM and the characteristics of flash make it ideal for certain applications, e.g. logging and caching for databases. He pointed out that by replacing part of the RAM with SSDs for caching, where applicable, can turn out to be economically better alternative. He even implied that having SSDs as an intermediate also justify for a system with slower spinning disks. He believed that the right balance of cost and performance could be found for any workload.

4.2 Scheduling

Pratt and Heger [28] conducted a study on performance evaluation of Linux 2.6 I/O schedulers. On their tests, they simulated I/O patterns on different hardware setups, including both single-disk and RAID configurations. They used Ext3 and XFS filesystems and various workload scenarios. They concluded that selecting an I/O scheduler has to be based on the workload pattern, the hardware setup and the filesystem used, or as they put it, "there is no silver bullet". Carroll [15] conducted a similar study on I/O schedulers in a RAID environment. He also found the selection of the I/O scheduler to be workload dependent and that I/O scheduling improves performance only on small to medium size RAID arrays (six disks or less).

Kim et al. [21] conducted a study to analyse I/O schedulers on SSDs. They argued that scheduling itself does not improve the read performance of an SSD, but preferring read requests over write requests does. They presented and implemented a scheduling scheme that exploits the characteristics of the SSD. The scheme is quite simple, it just bundles write requests together to match the logical block size and schedules read requests independently in a FIFO manner. Their benchmark tests showed up to 17% improvements over existing Linux schedulers (presented in Section 3.3.3). Test results also showed that the schedulers did not make a notable difference under read-oriented workloads on SSDs. On a side note, the anticipatory scheduler seemed to outperform other existing schedulers. This is quite strange because, as discussed earlier, the anticipatory scheduler tries to exploit the locality

of data on HDDs and thus the device is kept idle for short periods of time. This should not improve the performance of an SSD, but on the contrary, degrade it. This phenomenon can be explained by noting that an individual process can benefit for getting an exclusive service for bursty I/O requests and thus improving the overall performance. However, this is more a matter of process optimisation than I/O optimisation.

5. TESTING ENERGY EFFICIENCY

This chapter discusses of the test environment and the actual tests conducted. The first section describes the test cluster in detail. The second section represents the used test tools. The physics software and the software and hardware instruments used to gather data are discussed in this section. The third and last section discusses the practical side of running the tests and describes how the tests were conducted.

5.1 Test Cluster

5.1.1 Operating System: Rocks 5.3

The choice for the operating system of the test cluster is Rocks 5.3, an open-source Linux cluster distribution. Rocks is developed by the Rocks Cluster Group at the San Diego Supercomputer Center at the University of California, San Diego and its contributors. Rocks is a fully stand-alone system and cannot be installed on top of existing system. Rocks is basically a Red Hat Linux bundled together with a whole set of cluster software. The driving motivation behind Rocks is to make clusters easy to deploy, manage, upgrade and scale. This does not mean that Rocks would be inadequate or inefficient to do high performance cluster computing. On the contrary, Rocks is used in many universities and institutions around the world.

Installing and maintaining Rocks is easy. First you have to install the frontend machine. This does not differ much from a normal linux installation. Rocks contains many optional packages, called *rolls*, which you can pick to go with you basic installation. These rolls contain additional software you may want to install. For example, the *Sun Grid Engine (SGE) roll* was included and used as the choice of the batch-queuing system for the test cluster. After installing the frontend, a cluster also needs compute nodes. Installation of a compute node is easy. All that is needed, is to configure the compute node to boot from the network. A compute node registers itself to the frontend database, downloads a system image from the frontend (or from other compute nodes) and performs a quick installation. In fact, Rocks even deals with errors just by re-installing the compute node rather than trying to fix it. If the default configuration setup and system image is not sufficient enough for your needs or you want later to modify your compute nodes, all you need to do is to configure some text files on the frontend, maybe add some additional packages

to be installed on compute nodes, assemble a new system image and re-install the nodes.

Rocks also comes with many software tools that makes the administration and management of a cluster easy. Most notably the *Ganglia*, which is a web-based cluster monitoring software. [33]

With SGE it is possible to configure the slot size for each compute node. A slot size defines how many simultaneous jobs can be submitted to a single computer node. The name actually derives from number of CPU slots a machine has and it suggests that the number of CPU cores should be equal to the number of simultaneous compute jobs. However, this study wanted to try what kind of effect this has on the performance. This study uses a term *relative slot size* to refer the ratio of the slot size and the number of actual CPU cores. For example, in the test cluster, with quadcore machines, a slot size of eight would equal a relative slot size of two.

5.1.2 Hardware

The test environment consists of a computing cluster and a dedicated file server. Cluster is composed of four machines, frontend and three compute nodes. Detailed specifications are presented in Table 5.1. Detailed specifications of the drives used are presented in Table 5.2.

Table 5.1: Test Cluster

	Frontend	**Nodes**	**File Server**
Model	Dell server	Dell R210	Dell R710
Processor	Intel Xeon 2,8 GHz	Intel Xeon 2,4 GHz	Intel Xeon 2 GHz
CPU cores	2	4	4
RAM	2 GB	8 GB	2 GB
Disk (OS)	160GB SATA (7.2k)	250GB SATA (7.2k)	146GB SAS (10k)
Ethernet	2x 1Gb	2x 1Gb	4x 1Gb

SSDs are Corsair CSSD-F40GB-2 with a SATA II 3.0Gb/s interface. Corsair F40 utilises MLC NAND flash technology. According to manufacturer's own specifications, Corsair F40 can reach read and write speed of 270 MB/s and perform 50k IOPS. [17]

HDDs are Scorpio Black WD3200BEKT from Western Digital, with a 7200 RPM spindle speed and a SATA II 3.0Gb/s interface. According to a review made by Tom's Hardware web site, just to give a rough estimate of the performance of the HDD, the WD3200BEKT was benchmarked with access time of 15.4 ms (including spin delay), maximum read speed of 84.3 MB/s and maximum write speed of 83 MB/s. Also energy efficiency was measured, which resulted idle power of 1.12 W

and peak power of 3.26 W [1]. Western Digital [37] announces the WD3200BEKT to have an average latency of 4.2 ms and an average seek time of 12 ms, which converge quite well with numbers from Tom's Hardware review. However, power consumption does not converge, as Western Digital announces WD3200BEKT to have an idle power of 0.85W and an average power consumption of 2.1W. Also the manufacturer's numbers for HDD bandwidth differ considerably, as Western Digital claims the disk can put up to a 108 MB/s for both read and write.

Table 5.2: Manufacturer specification of the drive. Prices: www.newegg.com (cited 1-Feb-2011)

	HDD	**SSD**
Model	WD Scorpio Black	Corsair F40
Size	320 GB	40 GB
Price	$59.99	$104.99
GB/$	5.3	0.38
Random access time	16 ms	0.02 ms
Read speed	108 MB/s	280 MB/s
Write speed	108 MB/s	270 MB/s
IOPS	-	50 000
Idle power	0.8 W	0.5 W
Active power	1.75 W	2.0 W

5.2 Test Tools

5.2.1 Computing at CERN

The *Large Hadron Collider (LHC)* is a particle accelerator at CERN. The four main detectors of the LHC can produce 15 petabytes of data a year [6]. The distributed computing and data storage infrastructure built to process this vast amount of data is called the *Worldwide LHC Computing Grid (WLCG)*. As of February 2011, the WLCG had 246,000 processing cores and 142 petabytes of disk space [8].

The CERN computing infrastructure is divided into three level of tier centres. Tier-0 centre is located at CERN and is responsible for storing the first copy of RAW experiment data from LHC. It is also responsible for producing the first reconstruction pass and distribution of data to Tier-1 centres. Tier-1 centres together are responsible for storing the second copies of the data stored in Tier-0. Tier-1 centres also further reprocess the data and distribute it to Tier-2 centres. Tier-2 centres are responsible for serving the analysis requirements of the physicists and also producing and reprocessing of the simulated data. The simulated data is also

distributed to Tier-1 centres. As of February 2011, besides the Tier-0 centre, there are 11 Tier-1 centers and 164 Tier-2 centres in the world [7]. [18]

5.2.2 CMSSW

The *Compact Muon Solenoid (CMS)* is one of the four big research projects attached to LHC. CMS can also refer to the actual particle detector. The *Compact Muon Solenoid Software (CMSSW)* is a physics software toolkit for analysing the data from the CMS detector.

A central concept within the CMSSW is an event. An *Event* is a C++ object container. An Event contains many data tiers for all RAW and reconstructed data related to a particular collision. The *RAW* data is the full event information and collected directly from the LHC. The RAW data is unmanipulated and is not used for analysis. The reconstructed or *RECO* data is reconstructed to physics objects and still contains most of the event information. This RECO data can be used for analysis, but it is not convenient on any substantial data sample. *Analysis Object Data (AOD)* is a subset of RECO data. AOD is expected to be used in analysis as AODs are basically beforehand screened events. All objects in the Event may be individually or collectively stored in *ROOT* files. An event data can also be stored in different files to limit the size of the file and to prevent transferring unnecessary data. This data tier model of an Event is illustraded in Figure 5.1.

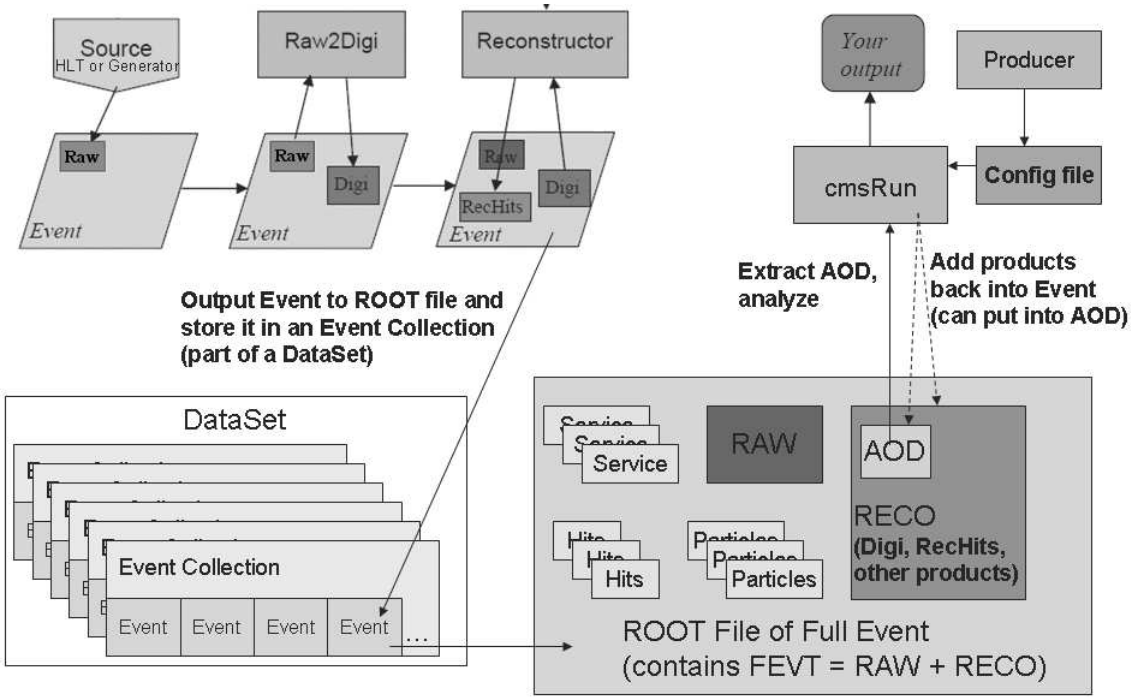

Figure 5.1: Data model used in CMSSW. Source: CMS WorkBook [16]

Before LHC was operational, raw event data was created using Monte Carlo -

simulation. Data samples generated by Monte Carlo are used to simulate the physics signal under investigation. It can also be used for creating a sample data for personal use.

CMSSW consists of many modules, which contains general purpose code for analysing the events. The goal is to minimise the code a physicist have to write himself. A configuration file is needed to tell the CMSSW which modules to load and where the data can be found. The executable is called cmsRun. [16]

5.2.3 ROOT framework and ROOT files

ROOT is a C++ framework designed for large scale data analysis and data mining. ROOT was first created at CERN, the project starting in 1995, and is still used in CERN for analysing the particle physics data created by LHC. One of the fundamental design principles was that although the programs analysing the data may change as time passes, the actual data does not. It was also designed to scale to handle petabytes of data. ROOT relies on a "write once, read many" -model due the nature of the data and makes it possible to compress the data efficiently.

A ROOT file is a compessed binary file, which can store any instance of a C++ class. Data is stored in a ROOT file with a data description so that it can be read even if the original program used to store the data is lost. Data can be stored in a ROOT file both row- and column-wise manner. If the data is stored by columns, reading the same data member from multiple instances speed up considerable as unwanted pieces of data can be skipped. For example in one instance, when a 280MB ROOT file was analysed, only 6.6MB of data was transferred over the network. ROOT even implements an auto-adaptive pre-fetch mechanism reading the next entry while previous entry is still being processed.

ROOT supports XML representation of data, but does not actually save data in XML form due the verbose nature of XML. Also a database abstraction layer is provided making it possible to store data in a ROOT file in a database-like manner. [29], [11]

5.2.4 Measuring Tools

During the tests, performance data was collected from the cluster by using both hardware and software tools. The actual power consumption was measured with a *WattsUp?* electricity meter, which was attached to the frontend machine via USB. A shell script was used to read the meter information once every second and to write the information into a log file. The electricity meter also provided a cumulative reading for the watt hours (Wh) consumed. The power consumption was measured separately for the file server and for all of the compute nodes. The power

consumption of the frontend machine was not measured. A grid monitoring software called *Ganglia* was also used. Ganglia operates by receiving constantly status reports from other machines in the cluster. Ganglia has a browser user interface to display cluster performance metrics, such as network traffic, CPU utilisation of individual machines, job queue, etc. The server logs were collected and stored together with the other output data.

5.3 Conducting Tests

5.3.1 About the performance and energy efficiency

We distinguish the performance and the energy efficiency as a two different optimisation goals. The performance is measured by the average processing times of the CMS jobs. The energy efficiency is measured by the energy in watt hours needed by an individual CMS job on average. These two can be highly dependant of each other. After all, by definition, energy equals time $\times$ power. However, the power does not need to be constant. It is possible, that increasing the performance it also has some kind of effect on the power usage. Thus, these two need to be studied separately.

5.3.2 Running tests

We created some Linux shell scripts both to automatise and standardise the testing process. Shell scripts were responsible for submitting the jobs, changing configurations where applicable (for example scheduling algorithm), clearing caches, starting and stopping wattage measurement and writing log entries. The shell scripts are attached as appendices. Appendix A shows the main script, Appendix B shows the script used for an individual test run and appendix C shows the script responsible for initialising and running the actual CMS job. Installing the drives and changing the file system needed to be done manually. A shell script was also used for creating the test input data on the target storage for the CMS jobs. To ensure homogeneous of the test data between different test configurations and between individual jobs, the test data was copied from the frontend for each time a file system was created. The drive caches both on compute host and data host was cleared between the test runs with shell command:

```
sync; echo 3 > /proc/sys/vm/drop_caches
```

Every test run was identical. The shell script first cleared caches and then set the scheduling algorithm. Then the slot size of the SGE was configured. Each compute node had 4 CPU cores as shown in Table 5.1. Slot sizes of 2, 4, 8 and 12 (relative slot

sizes of 0.5, 1, 2 and 3) was used to assign loads of 50-300% to each compute node. After the cluster was configured, the script submitted CMS jobs via SGE to each compute node equal to the current slot size of the node. Just before the jobs were submitted, an another script was started to log the wattage as mentioned in Section 5.2.4. When all the jobs were finished, also the logging script was terminated. Using the log file, starting and finishing time of a CMS job can be determined and also how much energy (watt hours) was consumed. After the first set of CMS jobs was finished, the script increased the slot size and ran a new set of jobs. When finished with a slot size of 12, scheduler was changed and slot size was set back to 2. This was repeated until all combinations of four different slot sizes and four different schedulers were used. All in all, one such test run submitted 312 CMS jobs and took about 8-10 hours to finish.

First, the test was conducted with NAS. A RAID-5 configuration of 6 HDDs (320 GB) and 4 SSDs (40 GB) was set up, creating volumes of 1.6 TB and 120 GB, respectively. The ROOT file used was 656 MB in size and it was copied to NAS total of 72 times each time and thus allocating 47 GB of the total volume. The files were renamed to "data-01-01.root"..."data-06-12.root", where the first number represented the node number and second number represented the job number. This ensured that no two CMS jobs was using the same data file. Also, the value of the read-ahead was altered to test the effect it had on the performance. Read-ahead values of 4kb, 8kb, 16kb and 32kb were used. After a test run of 312 CMS jobs finished, a new test run was started after changing the read-ahead value, the file system or RAID "disks" from HDDs to SSDs. All in all, the test run was conducted total of 24 times. 3 file systems $\times$ 4 read-ahead values $\times$ 2 different RAID "disks" equals 24.

At this point taking a quick look over the results, a pattern was perceived that indicated that increasing the read-ahead value had a negative impact on the performance. The reason most likely was that the ROOT file is a binary file and the AOD within the file is scattered. It was decided not to use the read-ahead value anymore as a configuration parameter. Also at this point, one test run was performed by using only 4 HDDs for easier comparison against the 4 SSDs. Again, based on the preliminary results, the best performing HDD configuration of 6 HDDs was picked and one more test run for 4 HDDs was performed with that configuration. Also, the energy consumption of idle compute nodes and NAS appliance was measured, both with and without the RAID pack. The idle tests logged an idle machine for one hour from startup. These results are represented in Appendix D.

Next, the SSDs were installed on the compute nodes and configured as a one big GlusterFS volume. With three nodes and without any striping or mirroring, the 40 GB SSDs created a volume of 120 GB. The test run was also conducted with this

configuration before dismounting the Gluster configuration and running the tests directly from the local drives. Because the test data was total of 47 GB, all of it could not be fitted into the 40 GB drives, so only half of it was used. Copying 24 GB of test data to each drive. This way, plenty of free space was left on the devices as had been the case also on earlier test runs.

Finally, the SSDs were changed to HDDs inside the compute nodes. As with SSDs, a GlusterFS volume was created first. With 320 GB in each node, a volume of 960 GB could be hosted by the nodes. After running the tests on Gluster, the same tests were conducted again with local drives. This time though, the whole 47 GB of test data was copied to each HDD.

6. RESULTS

The results chapter discusses the findings of the study individually. The performance and the energy efficiency are distinguished as a two different optimization goals as discussed in Section 5.3.1. However, this study also tries to evaluate the result as a whole. The performance gain is measured by comparing the average processing times of the CMS jobs. The energy efficiency gain is measured by comparing the energy in watt hours needed to run an individual CMS job. The results are presented as such or in relation to some default value. In the latter case, the performance or energy efficiency gain/loss is represented by percents. The results chapter is organized as follows.

Section 6.1 discusses what kind of an effect changing the slot size on performance. This study revealed that increasing the relative slot size had a positive effect and because of this, a two set of result data with relative slot sizes of one and three is represented later. Section 6.2 discusses the effects of changing the slot size on the energy efficiency.

Sections 6.3 and 6.4 discusses the effects of changing the read-ahead value on the performance and energy efficiency. This study found that increasing the read-ahead can have a positive effect on the power usage of the NAS appliance, but this effect is negated and out-weighted by the loss of performance. Thus increasing the read-ahead value had a negative impact to the energy efficiency as a whole.

Sections 6.5 and 6.6 discusses the importance of selecting the right file system and I/O scheduler. These sections reveal what kind of performance loss can happen if improper file system is selected and the same is done for schedulers. Finally, some estimation is represented for the combined effect for the system if both file system and I/O scheduler are not adequate for the workload at hand. Neglecting this aspect can lead to a performance loss of 6% on SSDs and more than a whopping 20% on HDDs.

Finally, in the Section 6.7, the best case results are represented for each of the three data storage scheme and for both drive types. This section is the most important in this chapter, because these configurations are screened thoroughly and most of the differencies perceived comes from the nature of the drive or scheme itself, not from the sub-optimal configurations. In this section, the differencies between a SSD and a HDD are most clearly visible. Also, the different fundamental approaches for

selecting the layout for the data storage scheme are as comparable between each other as it is possible in this study.

6.1 Slot size and performance

The test results showed that increasing the slot size had a positive effect on performance. Increasing the relative slot size from one to three had a performance gain of 5.4 – 9.6% with SSDs and 13 – 21% with HDDs. The results were filtered so that only the best performing configuration, i.e. file system, scheduler and read-ahead combination from each data storage scheme was selected. The energy consumption of an individual CMS job was used as a criteria. The results are illustraded in Figure 6.1, grouped by data storage scheme. In a group of four for each scheme, the left-most represents the relative slot size of 0.5 and right-most represents the relative slot size of 3. Remember, that the relative slot size of 0.5 equals only half of the potential CPU cores utilized.

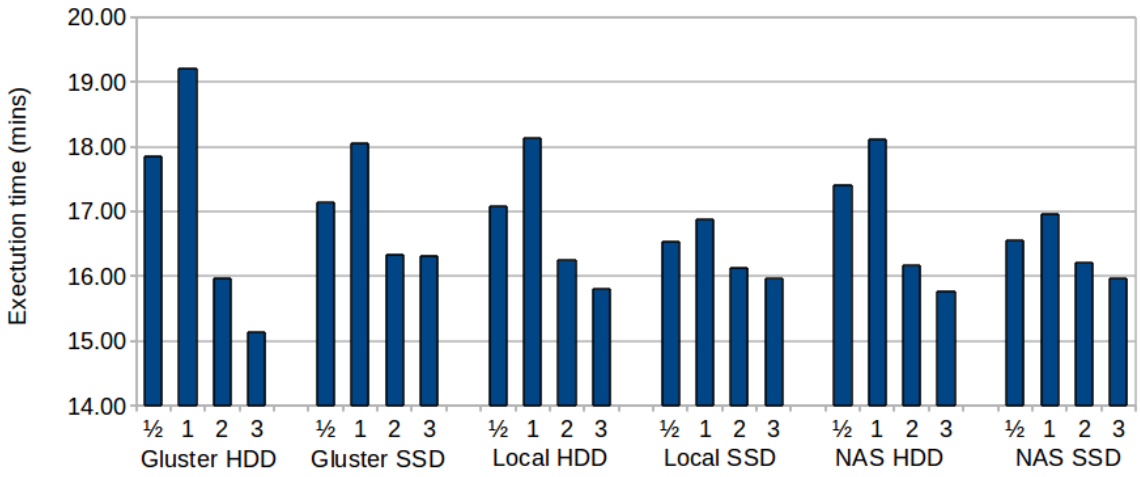

Figure 6.1: Comparing different slot sizes. Results are grouped by data storage scheme and drive used. Relative slot sizes of 0.5 – 3 was used.

This study propose that the positive correlation of increased slot size and performance stem from abolishing the effect of I/O wait. As one process (CMS job) waits data to arrive, the CPU can be given to another process and thus the CPU cycles can be utilized more efficiently while waiting for I/O.

The performance gain seemed to be relatively smaller for SSDs than HDDs. This can be explained by SSDs having a better read performance and SSDs can thus service data requests sooner than HDDs, even when using the relative slot size of one. This could also explain why both HDDs and SSDs perform almost identically with the same data storage scheme and with the relative slot size of two. The compute node is now more likely to have a process being ready for execution, regardless of used data storage scheme as over-provisioning of the node is introduced.

Increasing the relative slot size from two to three is shown to improve the performance of the HDDs even more, but to have no effect on SSDs. It is not clear to us why this is happening.

6.2 Slot size and energy efficiency

If studying the power usage of the compute node alone, the results show that the compute nodes consume less power on average with relative slot size of one than with two or three as illustraded in Figure 6.2. When including also the time factor and now studying the over-all energy consumption of the test cluster (scaled to represent energy per job), it is discovered that changing the slot size has very little effect on the energy efficiency with SSDs and with local HDD. This is illustraded in Figure 6.3, which also includes the test case where the compute nodes are only half-utilized. This clearly shows, that a very large portion of the energy used by a compute node is consumed by the processors and that the energy consumption is proportional to the load of the machine.

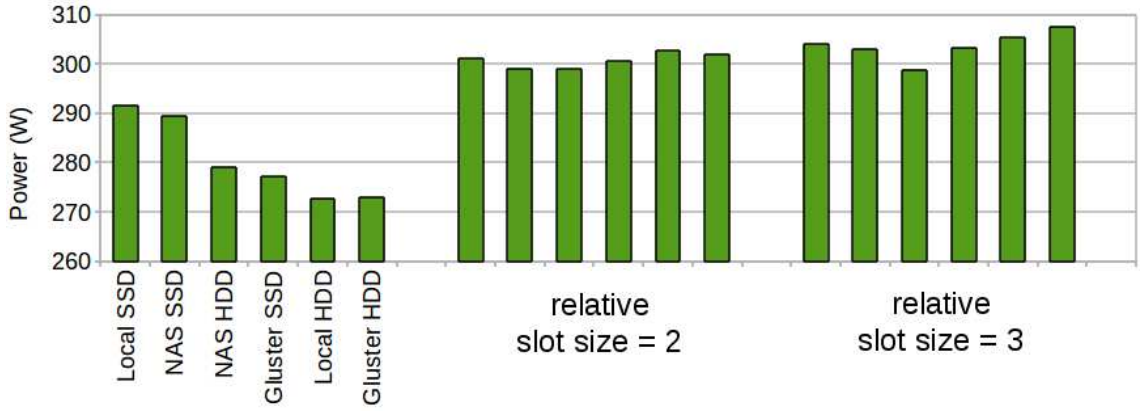

Figure 6.2: Total power consumption of the three compute nodes on average. Relative slot sizes of one, two and three were used. Data schemes are in the same order in each set.

Although the average job processing time decreases when increasing the slot size, the power usage of the node is increased. This is quite natural, because what really is improved is the utilization of the processor of the node. The increased performance and decreased power usage counter each other and lead to almost similar energy efficiency in terms of Wh/job (see Figure 6.3). In other words, the energy consumption increases linearly in relation to performance.

When using HDDs with NAS or with Gluster, the linearly proportional energy consumption is no longer valid. This is because relatively better performance increase gain discussed earlier in Section 6.1.

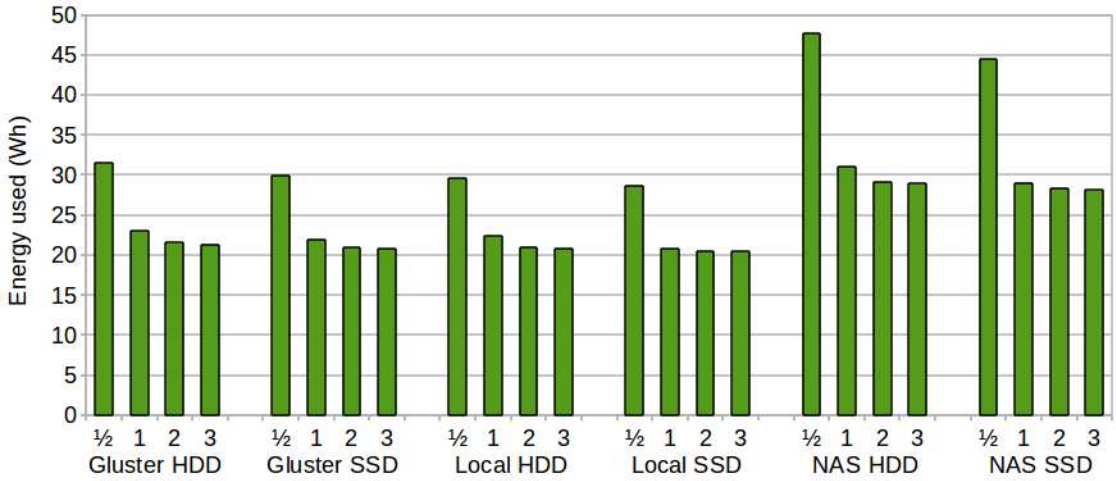

Figure 6.3: Total energy consumption of the cluster per CMS job. Results are grouped by data storage scheme and drive used. Relative slot sizes of 0.5 – 3 was used.

6.3 Read-ahead and performance

The test results showed that increasing the read-ahead value had no effect on performance with SSDs. With HDDs it had a negative effect on almost all cases. The only exception was the XFS file system with the relative slot size of three. In this case, increasing the read-ahead value had performance gain of 2% on average job processing time. Interestingly, the worst performance loss of 6% was also measured when using XFS and HDDs, but with the relative slot size of one. The results were filtered to include only the best performing set of configuration. The absolute results are illustraded in Figure 6.4. Figure 6.5 illustrades the results in relation to default read-ahead value of 4kb. A positive number represents the performance gain in relation to 4kb read-ahead value of the same data storage scheme and drive used. The results were all measured with the noop scheduler. As NAS exploited RAID technology, using noop for scheduling should be best choice as discussed in Section 3.3.3.

6.4 Read-ahead and energy efficiency

The test results showed that increasing the read-ahead value had a small positive effect (one percent or less) on the energy efficiency with SSDs, excluding the XFS file system, which was not affected by the change in read-ahead. With HDDs, the effect was mostly negative, excluding the Ext4 file system, which performed slighty better. The results are illustraded in Figure 6.6. The numbers represent the change in energy consumption of the whole cluster (including NAS) as a function of the read-ahead value. Read-ahead value of 4 kilobytes is used as a point of reference and the rest of the configuration is left untouched. A positive number equals less energy. There

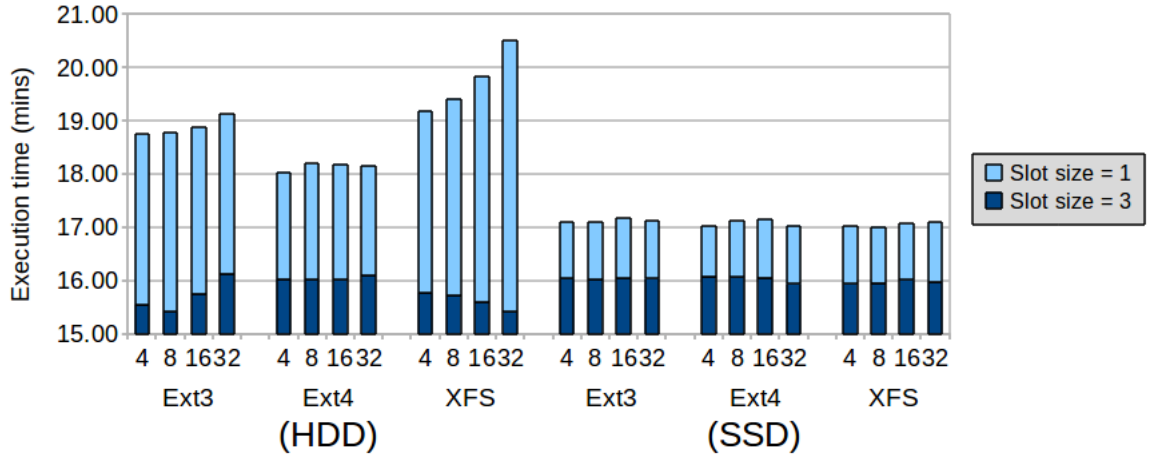

Figure 6.4: Comparing different read-ahead values in NAS configuration. Numbers 4 – 32 represents read-ahead in kilobytes.

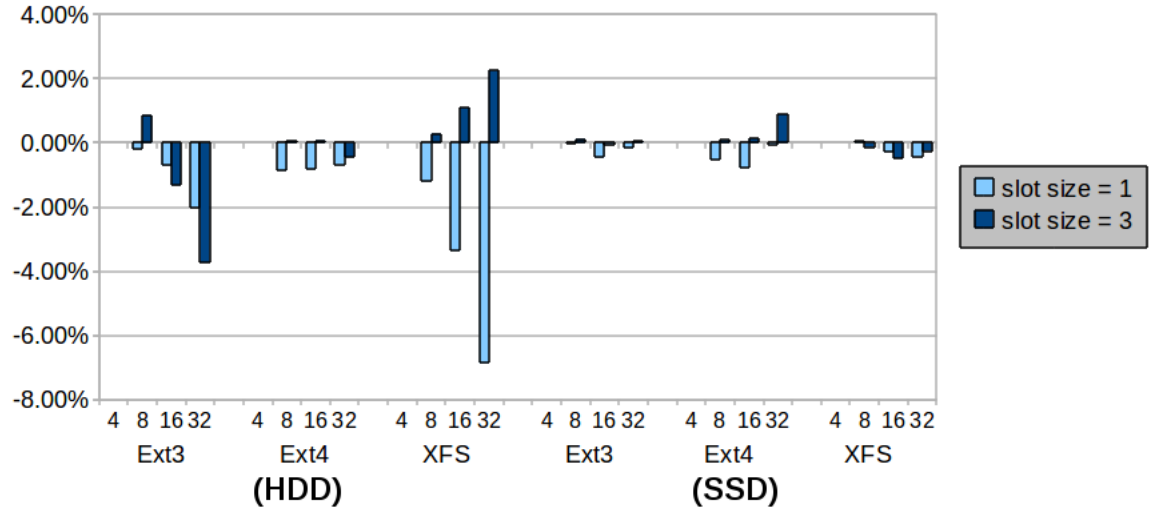

Figure 6.5: The change in performance as a function of the read-ahead value. A positive number equals faster processing time. Numbers 4 – 32 represents read-ahead in kilobytes.

seems to be no clear pattern between performance and energy efficiency, although some similarities can be recognised.

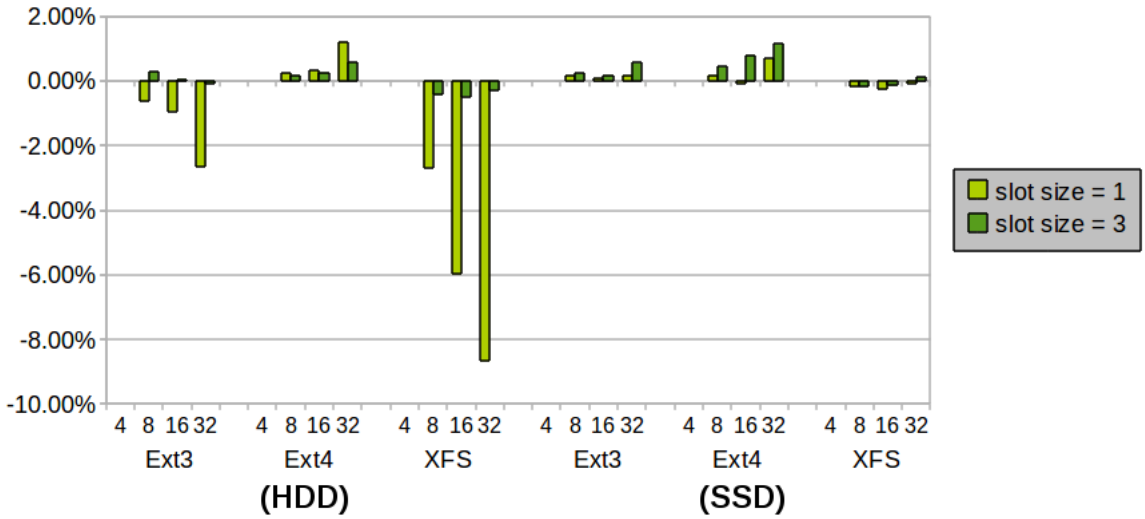

Figure 6.6: The change in energy consumption as a function of the read-ahead value. A positive number equals less energy used. Numbers 4 – 32 represents read-ahead in kilobytes.

6.5 File system

The test results showed that the choice of the file system had a much greater effect on HDDs than on SSDs. The performance variation between the best and the worst performing file system, on otherwise similar configuration, was 1 – 6 % on HDDs, but only 0.1 – 0.7 % on SSDs. The difference in energy efficiency was upto 6 percent on HDDs and less than 1.5 percent on SSDs. These results are represented more closely in Table 6.2. The absolute results of the benchmark tests are represented in Table 6.1.

In general, the differencies among the file systems with SSDs were small and it did not matter if the relative slot size was one or three. With HDDs, increasing the relative slot size from one to three led to more variation among the file systems. Most likely this is happening because increased number of parallel CMS jobs created more I/O requests and the I/O pattern became more complex. This was necessary to differentiate the file systems and under heavier utilisation some differencies between these file systems started to emerge.

We believe there are two reasons why there was so little differencies among the file systems with SSDs. First, these file systems are built with mostly HDDs in mind. Second, SSDs are also more effective by default than HDDs, hence the significance of the file system is much smaller for SSDs. In other words, the SSDs are effective, regardless of the file system.

Of the three file systems tested here, the best choice for SSDs seemed to be the XFS file system and the Ext4 file system for HDDs.

Table 6.1: Comparing file systems on different drives and data storage schemes.

			slot size = 1		slot size = 3	
scheme	**drive**	**File System**	**Wh**	**time**	**Wh**	**time**
Local	HDD	Ext3	23.8	19.00	21.2	15.55
		Ext4	**22.4**	**18.08**	**20.8**	15.56
		XFS	22.5	18.23	20.8	**15.48**
	SSD	Ext3	21.0	**16.52**	20.6	**15.56**
		Ext4	21.0	16.54	20.6	15.57
		XFS	**20.8**	**16.52**	**20.4**	15.57
NAS	HDD	Ext3	32.0	18.44	29.2	**15.21**
		Ext4	**31.0**	**18.02**	29.0	15.52
		XFS	32.6	19.09	**28.9**	15.46
	SSD	Ext3	29.4	17.05	28.5	16.03
		Ext4	29.2	17.01	28.4	16.02
		XFS	**29.0**	**16.58**	**28.1**	**15.56**

Table 6.2: The variation of energy efficiency and performance with different file systems on otherwise similar configurations.

		slot size = 1		slot size = 3	
scheme	**drive**	**energy**	**performance**	**energy**	**performance**
Local	HDD	6.2%	4.6%	1.8%	0.9%
	SSD	1.0%	0.1%	1.0%	0.1%
NAS	HDD	4.8%	5.8%	1.1%	3.2%
	SSD	1.4%	0.7%	1.5%	0.7%

6.6 I/O Scheduler

The test results showed that changing the I/O scheduler on SSDs is almost insignificant. Excluding the lone case of using the cfq scheduler on local data storage scheme and the relative slot size of one, the variation between different schedulers was only one percent or less. In terms of time and energy this equals to only one tenth of a watt hour per job or about 10 seconds on average job processing time. These results are represented in Table 6.3 for local data storage scheme and in Table 6.4 for NAS. The variation is represented in Table 6.5. We believe that the explanation is quite simple. I/O scheduling was designed to improve the shortcomings caused by the mechanical nature of the HDDs. In theory, SSDs should not benefit from

I/O scheduling at all, as discussed in Section 4.2. This being said, the best choice for the I/O scheduler on SSDs is the noop scheduler, as predicted and as the tests here confirmed.

Table 6.3: Comparing different I/O schedulers on local drive.

Local		slot size = 1		slot size = 3	
drive	**scheduler**	**Wh**	**time**	**Wh**	**time**
HDD	as	22.9	18.54	20.9	16.04
	cfq	23.5	19.43	21.0	16.14
	dl	**22.4**	**18.08**	**20.8**	15.55
	noop	22.5	18.16	**20.8**	**15.48**
SSD	as	20.9	17.06	20.5	15.57
	cfq	21.4	17.52	20.5	**15.56**
	dl	**20.8**	**16.52**	20.5	15.58
	noop	**20.8**	**16.52**	**20.4**	15.57

In general, the variation was much greater when the relative slot size of one was used. This was the case for both HDDs and SSDs. This is a bit counterintuitive as higher relative slot size should generate more I/O requests and more variation to the I/O pattern. Thus the significance of the scheduling should become more important. However, it could be argued that the reason for this is something else than the scheduling itself. The fact that SSDs should not benefit from the scheduling, as mentioned earlier, and that this phenomenon was also perceived with SSDs, back up this assumption.

Table 6.4: Comparing different I/O schedulers on NAS appliance.

NAS		slot size = 1		slot size = 3	
drive	**scheduler**	**Wh**	**time**	**Wh**	**time**
HDD	as	32.2	19.10	29.2	15.52
	cfq	31.1	18.07	28.9	15.33
	dl	**31.0**	18.07	**28.9**	**15.21**
	noop	31.1	**18.02**	**28.9**	15.33
SSD	as	29.2	17.10	**28.1**	15.58
	cfq	**29.0**	**16.58**	**28.1**	15.59
	dl	**29.0**	16.59	**28.1**	15.58
	noop	**29.0**	17.01	**28.1**	**15.56**

If excluding the anticipatory scheduler (as), the other schedulers did not had any remarkable differencies with HDDs on NAS as shown in Table 6.4. As already stated in Section 6.3, this is because NAS exploits RAID technology and do not benefit

from I/O scheduling. On the contrary, excessive I/O scheduling can degrade the performance of the RAID considerably. As discussed in Section 3.3.3, the anticipatory scheduler waits for consecutive I/O requests and keeps the drive idle for a short period of time. This behaviour is most likely the reason for the poor performance of the anticipatory scheduler.

Table 6.5: The variation of energy efficiency and performance with different schedulers on otherwise similar configurations.

		slot size = 1		slot size = 3	
scheme	**drive**	**energy**	**performance**	**energy**	**performance**
Local	HDD	5.0%	8.1%	0.7%	2.7%
	SSD	2.7%	5.5%	0.2%	0.1%
NAS	HDD	3.6%	5.9%	1.0%	3.2%
	SSD	0.8%	1.2%	0.2%	0.3%

The combined effect of choosing the right file system and the most suitable I/O scheduler can be seen in Table 6.6. The results show, that in the worst case a HDD-based configuration could suffer a performance loss of 7 – 23%. With SSD-based configuration, the changes are that the system is within two percent from the best possible configuration, but a performance degrade of 6% is possible. The trend for the energy efficiency is similar, but this was expected as the energy efficiency stems from the performance, as already mentioned in Chapter 2.

Table 6.6: Comparing different file systems and I/O schedulers together. Numbers represent the variation of best and worst case for otherwise similar setups.

drive	**scheme**	**slot size**	**energy**	**time**
HDD	Local	1	21.9%	23.4%
		3	5.6%	7.0%
	NAS	1	10.1%	16.1%
		3	3.0%	7.8%
SSD	Local	1	3.7%	6.1%
		3	1.3%	0.5%
	NAS	1	2.3%	1.8%
		3	1.6%	1.0%

6.7 The best case

This section represents the best-case results for each data storage scheme: the RAID on NAS, the local drives directly on compute nodes, and the distributed file system

created with shared drives on compute nodes and GlusterFS software. Results are shown for both solid state and hard disk drives. Also, the results are distinguished for using both relative slot sizes of one or three. Table 6.7 shows the actual energy consumed and the processing time needed to complete an individual CMS job for each setup.

Table 6.7: The best result measured for each drive type and data storage scheme.

		slot size = 1		slot size = 3	
scheme	**drive**	**Wh**	**time**	**Wh**	**time**
GlusterFS	HDD	23.0	19.12	21.3	15.08
GlusterFS	SSD	21.9	18.03	20.8	16.19
Local	HDD	22.4	18.08	20.8	15.48
Local	SSD	20.8	16.52	20.4	15.57
NAS	HDD	31.0	18.07	28.9	15.46
NAS	SSD	29.0	16.59	28.1	15.56

The most energy efficient setup was, quite predictably, the local drive approach using SSDs and relative slot size of three. This setup consumed only 20.4 watt hour per job on average. The best performing, e.g. the fastest setup, was the distributed file system model using HDDs and relative slot size of three. The most surprising thing was that this setup outperformed others clearly with a marginal of almost 40 seconds. First we suspected an error, but after reviewing the data, we discovered that as good runtime was also recorded when using a different I/O scheduler on an another test run. Also, HDDs outperformed SSDs in all three data storage schemes if the relative slot size was three. Although, the marginals were a lot less, only about 10 seconds. We are not certain why the results differ so much when using GlusterFS, but our educated guess is that it derives from the GlusterFS software and the way it is implemented. Either the cache of the GlusterFS (and the cache of the HDD) is working very well or the GlusterFS could not adapt to work with SSDs and the SSDs were just clogged with the excessive I/O traffic.

When studying the energy efficiency (with a relative slot size of three), it can be observed that HDDs consume 0.4 – 0.8 watt hour more than SSDs. As the average processing time is about one-fourth of an hour, the difference in power usage is approximately quadruple and thus 1.6 – 3.2 watts. In this study four or three drives were used depending if the drives were in the NAS or in the compute nodes (only having three compute nodes). This means that one HDD consumed around 0.5 – 1 watt more energy than one SSD.

When comparing different data storage schemes, it is not fair to just compare the energy consumption. NAS is consuming much more energy per CMS job than

other schemes. This is obviously because there is one more server machine running. This leads to about 7 – 8 watt hour of overhead per job with NAS compared to others. The NAS appliance had 32 drive bays, but only 4 was used. Leaving seven-eights of potential resources unused, so it could be argued that the overhead is more likely close to 1 watt hour per drive. Again, the average processing time being approximately one-fourth of an hour, this equals to around four watts per drive of power overhead. The NAS needed roughly 115 watts of power when running idle without any drives installed, so the "about four watts per drive" for fully loaded 32-drive NAS appliance is a pretty good estimate.

Of course, there is no guarantee that the results would apply if increasing the number of drives and I/O load of the NAS. These results are only suggestive at best. However, they do reveal that relocating the data away from the compute nodes do not improve the performance of the compute nodes notably. In other words, storaging data and providing data access to other nodes is not a burden for the compute node.

7. CONCLUSION

The goal of this study was to find out wheter or not solid state drives are suitable to be used in cluster computing and if they really are superior to hard disk drives in a spirit of green IT. No problems were encountered while introducing solid state drives into the cluster environment. In the process of doing this research, an extensive background study was made on the differencies of these two drive types. Understanding these differencies did not raise any significant concerns, which would prevent using solid state drives in cluster computing.

The tests results revealed that selecting the solid state drives over hard disk drives do not provide any performance gain. Hard disk drives proved to outperform the solid state drives in all three data storage schemes used in these tests. When solid state drives suffer from high retail prices and low storage capacities at the moment compared to hard disk drives there is no reason to choose solid state drives over conventional hard disk drives from performance point of view.

It is true, that solid state drives consume less energy. This was measured to be around one watt per drive. Even if taking into account the effect of power usage efficiency (PUE) (discussed in Chapter 2), which multiplies this by a factor of 1.2 – 2 depending on the data center, it is not justifiable to declare solid state drives to be more energy efficient. The reason is, that the storage capacity of the hard disk drives multifold to solid state drives. One gets more storage space per kWh with hard disk drives.

The results speak for themselves. This study found that overprovisioning the compute nodes increases job throughput. Scheduling more than one job per core have a positive correlation with the average processing time. This indicates there are unused resources in clusters, which use the number of cores as a basis of submitting jobs. It was also discovered, that a performance loss of over 20% can exist if the used file system and scheduler is not properly selected. Results indicate that the differencies between solid state and hard disk drives are quite small and the right configuration matters more than the drive type used. These results can provide a sound basis for optimisation of other cluster environments. What is good to understand is to optimise things that matter most and this research can give some hints of what those things might be.

If taking a closer look at the results from strictly energy efficiency point of view,

one may be fooled into thinking, that it is the watts that matter. If optimisation can lower the power usage by one watt as done here, but at the same time improve the performance by two percent, it is the performance increase that really saves energy. At least in a fully utilised environment as in cluster computing.

The purpose of this study is to be a review about solid state drives and their energy efficiency. Providing the theoretical background of using solid state drives in cluster computing. This study could be used as a starting point to anyone who is interested of solid state drives and cluster computing. This study also reported the experiencies of implementing these theories into practice. This pragmatic use case can be used as a frame of reference, which helps to understand the concepts attached to the topic. Also many assumptions predicted by the theory was confirmed in practice.

This study propose, that a further study is not needed immediately, but if the prices of solid state drives decline and their storage capacities increase to match those of hard disk drives a new study should be conducted. Also, the feeble performance of solid state drives with GlusterFS software was most likely because the software could not operate with the drives. Although the reason can also be in poor configuration, this could require more investigation.

This study had some interesting findings. In general, the hard disk drives were performing better than expected.

BIBLIOGRAPHY

[1] Tom's Hardware: Review notebook hard drive (Western Digital Scorpio WD3200BEKT). Website, 2008. http://www.tomshardware.com/reviews/notebook-hard-drive,2006-7.html (cited 30-Nov-2010).

[2] Intel Solid-State Drive Optimizer. Intel White Paper, 2009.

[3] Wear Leveling Technology. Apacer White Paper, 2009.

[4] An Introduction to Gluster Architecture. Gluster White Paper, 2011.

[5] Designing a Very Efficient Data Center. Facebook Engineering's Facebook Notes (blog), 2011.

[6] LHC Computing. Website, 2011. http://public.web.cern.ch/public/en/lhc/Computing-en.html (cited 17-Feb-2011).

[7] Worldwide LHC Computing Grid - Grid Topology. Website, 2011. http://gstat-wlcg.cern.ch/apps/topology/ (cited 17-Feb-2011).

[8] Worldwide LHC Computing Grid - Site Capacities. Website, 2011. http://gstat-wlcg.cern.ch/apps/capacities/sites/ (cited 17-Feb-2011).

[9] Lal Shimpi Anand. The SSD Anthology: Understanding SSDs and New Drives from OCZ. Website, 2009. http://www.anandtech.com/print/2738 (cited 1-Feb-2011).

[10] Eric Anderson and Joseph Tucek. Efficiency matters! *ACM SIGOPS Operating Systems Review*, 44, 2010.

[11] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. Gonzalez Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. Marcos Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, and M. Tadel. ROOT - A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 180(12):2499–2512, 2009. 40th Anniversary Issue.

[12] Jens Axboe. Linux Block IO - present and future. In *Proceedings of the Linux Symposium*, pages 51–62, 2004.

[13] Rick Bauer and Sol Squire. Building the Green Data Center. SNIA tutorials, 2008. http://www.snia.org/education/tutorials/2008/spring/green/Bauer-Squire_Building_the_Green_Data_Center.pdf (cited 17-Feb-2011).

[14] Kenneth P. Birman. *Reliable Distributed Systems - Technologies, Web Services and Applications*. Springer, 2005.

[15] Aaron Carroll. I/O Scheduling on RAID. Bachelor's Thesis. The University of New South Wales. School of Electrical Engineering and Telecommunications, 2008.

[16] CERN. *The CMS WorkBook*, 2009. `https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook` (cited 17-Feb-2011).

[17] Corsair. *CSSD-F40GB2 - Product Specification Sheet*, 2011. `http://www.corsair.com/cssd-f40gb2.html` (cited 16-Feb-2011).

[18] Claudio Grandi, David Stickland, and Lucas Taylor. *The CMS Computing Model*. CERN, December 2004.

[19] Ilkka Haikala and Hannu-Matti Järvinen. *Käyttöjärjestelmät*. Talentum, 2004.

[20] Intel. *Intel X18-M/X25-M SATA Solid State Drive - Product Manual*, January 2010.

[21] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Disk Schedulers for Solid State Drives. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 295–304, New York, NY, USA, 2009. ACM.

[22] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A Space-efficient Flash Translation Layer for CompactFlash Systems. *IEEE Transactions on Consumer Electronics*, 48, 2002.

[23] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, and Sang-Woo Kim. A Case for Flash Memory SSD in Enterprise Database Applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1075–1086. ACM, 2008.

[24] Adam Leventhal. Flash Storage Memory. *Communications of the ACM*, 51(7):47–51, 2008.

[25] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux Symposium*, volume 2, pages 21–34, 2007.

[26] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In

Proceedings of the 4th ACM European conference on Computer systems, pages 145–158, New York, NY, USA, 2009. ACM.

[27] Tapio Niemi, Jukka Kommeri, Kalle Happonen, Jukka Klem, and Ari-Pekka Hameri. Improving Energy-Efficiency of Grid Computing Clusters. *Lecture Notes in Computer Science*, 5529:110–118, 2009.

[28] Steven L. Pratt and Dominique A. Heger. Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers. In *Proceedings of the Linux Symposium*, pages 425–448, 2004.

[29] Fons Rademakers and René Brun. ROOT: An Object-Oriented Data Analysis Framework. *Linux Journal*, (51), 1998.

[30] Erik Riedel and Patrick Stanko. Green Storage Products: Efficiency with ENERGY STAR & Beyond. SNIA tutorials, 2010. `http://www.snia.org/education/tutorials/2010/fall/green/RiedelStanko_Greenstorage_II_101011.pdf` (cited 17-Feb-2011).

[31] Tony Roug. Using MLC NAND in datacenters. SNIA tutorials, 2010. `http://www.snia.org/education/tutorials/2010/spring/solid/TonyRoug_Using_SSD_MLC_NAND.pdf` (cited 17-Feb-2011).

[32] Karsten Schmidt, Yi Ou, and Theo Härder. The Promise of Solid State Disks: Increasing efficiency and reducing cost of DBMS processing. In *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*, pages 35–41, New York, NY, USA, 2009. ACM.

[33] Joseph D. Sloan. *High Performance Linux Clusters with OSCAR, Rocks, openMosix and MPI.* O'Reilly media, Inc., 2005.

[34] Roderick W. Smith. Linux on 4KB-sector disks: Practical advice. IBM White Paper, 2010.

[35] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the XFS file system. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 1996. USENIX Association.

[36] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the Energy Efficiency of a Database Server. In *Proceedings of the 2010 international conference on Management of data*, pages 231–242, New York, NY, USA, 2010. ACM.

[37] Western Digital. *Western Digital Scorpio Black - Product Specification Sheet*, 2010. http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701275.pdf (cited 30-Nov-2010).

[38] Alan G. Yoder. Green Storage Technologies and Your Bottom Line. SNIA tutorials, 2010. http://www.snia.org/education/tutorials/2010/spring/green/Alan%20Yoder_Green_Storage_Technologies.pdf (cited 17-Feb-2011).

A. SHELL SCRIPT: SCHEDULING A SET OF TEST RUNS

```
#!/bin/bash
#
# File name: run_tests.sh
# Run from the frontend.
#

if [ "$USER" != "root" ]; then
  echo " Warning: You should log in as root"
  exit
fi



### read_ahead = 0 kb ###
rocks run host compute "hdparm -a 0 /dev/sdb1" > /dev/null
rocks run host compute "echo 0 > /sys/block/sdb/queue/read_ahead_kb" > /dev/null



### NOOP SCHEDULER ###
rocks run host compute "echo noop > /sys/block/sdb/queue/scheduler" > /dev/null

### process/node: 2, 4, 8 & 12 ###
rocks run host "sync; echo 3 > /proc/sys/vm/drop_caches" > /dev/null
sync > /dev/null; echo 3 > /proc/sys/vm/drop_caches;
/opt/gridengine/bin/lx26-amd64/qconf -mattr queue slots 2 dell.q
/home/mtuomine/IO_tests/jobs/CMS_TauAnalysis/frontend.sh 3 12 cms_02-procs_local-SSD-Ext4-noop
rocks run host "sync; echo 3 > /proc/sys/vm/drop_caches" > /dev/null
sync > /dev/null; echo 3 > /proc/sys/vm/drop_caches;
/opt/gridengine/bin/lx26-amd64/qconf -mattr queue slots 4 dell.q
/home/mtuomine/IO_tests/jobs/CMS_TauAnalysis/frontend.sh 3 12 cms_04-procs_local-SSD-Ext4-noop
rocks run host "sync; echo 3 > /proc/sys/vm/drop_caches" > /dev/null
sync > /dev/null; echo 3 > /proc/sys/vm/drop_caches;
/opt/gridengine/bin/lx26-amd64/qconf -mattr queue slots 8 dell.q
/home/mtuomine/IO_tests/jobs/CMS_TauAnalysis/frontend.sh 3 12 cms_08-procs_local-SSD-Ext4-noop
rocks run host "sync; echo 3 > /proc/sys/vm/drop_caches" > /dev/null
sync > /dev/null; echo 3 > /proc/sys/vm/drop_caches;
/opt/gridengine/bin/lx26-amd64/qconf -mattr queue slots 12 dell.q
/home/mtuomine/IO_tests/jobs/CMS_TauAnalysis/frontend.sh 3 12 cms_12-procs_local-SSD-Ext4-noop



### ANTICIPATORY SCHEDULER ###
...
```

[Anticipatory, deadline and cfq are handled in a similar manner to noop.]

B. SHELL SCRIPT: ONE TEST RUN

```
#!/bin/bash
#
# File name: frontend.sh
# Run from the frontend.
#

if [ "$USER" != "root" ]; then
  echo " Warning: You should log in as root"
  exit
fi

if [ ! -n "$1" ] || [ ! -n "$2" ] || [ ! -n "$3" ]; then
  echo " Usage: run <number of nodes> <number of runs> <logmessage/folder name>"
  exit
fi

# shell parameters
nodes=$1
runs=$2
logmessage=$3

# Files and directories
jobid=$$
workDir=/state/data/TauAnalysisOutputRoots/$jobid
rootDir=/home/mtuomine/IO_tests
jobDir=$rootDir/results/$logmessage
rrdDir=$jobDir/rrds
logfile=$rootDir/myjoblog.txt
jobinfo=$jobDir/job_info.txt

mkdir --parents $workDir
mkdir $jobDir
mkdir $rrdDir

# WattsUp logging
$rootDir/watts/wattslog.py --device=/dev/ttyUSB0 > $jobDir/wattslog_nodes.log &
wattslogid_1=$!
$rootDir/watts/wattslog.py --device=/dev/ttyUSB1 > $jobDir/wattslog_nas.log &
wattslogid_2=$!

# Clear the job queue on before exiting (on termination)
trap '{ qdel -u root; kill $wattslogid_1; kill $wattslogid_2; kill $iopid; exit 0; }' SIGINT

# Write log updates
echo "================================================================================" >> $logfile
echo "CMS_TauAnalysis ### `date` ### Job ID: $jobid" >> $logfile
echo " - Run for $runs cycles on $nodes nodes" >> $logfile
echo " - $logmessage" >> $logfile
echo "CMS_TauAnalysis ### `date` ### Job ID: $jobid" >> $jobinfo
echo " - Run for $runs cycles on $nodes nodes" >> $jobinfo
```

```
echo " - $logmessage" >> $jobinfo
echo " - NAS scheduler: `rocks run host nas-0-0 'cat /sys/block/sdb/queue/scheduler'`" >> $jobinfo
echo " - NAS read_ahead_kb: `rocks run host nas-0-0 'cat /sys/block/sdb/queue/read_ahead_kb'`" >> $jobinfo

# Echo screen
echo
echo " ### Jobs started: `date` ###"
echo "  - Run for $runs cycles on $nodes nodes"
echo "  - $logmessage"
echo

# Submit jobs on compute nodes
for run in $(seq 1 $runs);
do
   for node in $(seq 1 $nodes);
        do
           qsub -q dell.q -b yes $rootDir/jobs/CMS_TauAnalysis/node_cmsRun.sh
                                    $node $run $logmessage
        done
done

echo "  Jobs running..."

active=1
iterations=0
while [ $active -eq 1 ]
        do
           tmp=`qstat | wc -l`
                if [ $tmp -eq 0 ]
                then
                     active=0
                fi
                sleep 1
                let iterations=$iterations+1
                if [ $iterations -eq 3600 ]
                        then
                        echo "Saving RRDs - `date`"
                        timestamp=`echo \`date +%l%MS\` | sed '/^$/d'`
                        mkdir $rrdDir/$timestamp
                        cp -r /var/lib/ganglia/rrds/testCluster $rrdDir/$timestamp/
                        iterations=0
                fi
        done

kill $wattslogid_1
kill $wattslogid_2

# Write log updates
echo " - All jobs finished: `date`" >> $logfile
echo " - All jobs finished: `date`" >> $jobinfo

# Copy the final RRDs
mkdir $rrdDir/final
cp -r /var/lib/ganglia/rrds/testCluster/ $rrdDir/final/

echo " ### All jobs finished: `date` ###"

# Wait 1 minute for compute nodes loads to settle
sleep 60
```

C. SHELL SCRIPT: RUNNING AND TIMING A CMS TAUANALYSIS JOB

```
#!/bin/sh
#
# File name: node_cmsRun.sh
# Script running on the compute nodes
#

# My variables
JOB_ID=$$
NODE=$1
RUN=$2
CALLER_ID=$3
HOST=‘uname -n‘
let "TENS = $RUN / 10"
let "ONES = $RUN % 10"

WORK_DIR=/state/data/TauAnalysisOutputRoots/$CALLER_ID/$HOST-run$TENS$ONES
ROOT_DIR=/home/mtuomine/IO_tests
JOB_DIR=$ROOT_DIR/results/$CALLER_ID
# configuration files are named [node]-[root-file].py, e.g. 01-01.py
LOG_FILE=$JOB_DIR/$HOST-$JOB_ID.out

CMS_ROOT=/state/partition1/cms
SRC=workspace/CMSSW_3_6_1/src
CONF_DIR=$ROOT_DIR/jobs/CMS_TauAnalysis/conf_files
CONF_FILE=0$NODE-$TENS$ONES.py


# Create work directory for output root file
mkdir --parents $WORK_DIR


# Set environment
cd $CMS_ROOT
source environment
cd $CMS_ROOT/$SRC
cmsenv


# Run TauAnalysis
exec > $LOG_FILE 2>&1
cd $WORK_DIR
echo "### cmsRun started: ‘date‘ ###"
time cmsRun $CONF_DIR/$CONF_FILE
echo "### cmsRun finished: ‘date‘ ###"
```

D. POWER USE PROFILE OF NAS WITH BOTH DRIVE TYPES

The Figure D.1 illustrates the power usage of NAS under different configuration and loads. The idle graphs represent the power use of freshly booted machine, starting after 10 minutes from boot up and running approximately 50 minutes. The loaded graphs represents running a set of twelve CMS jobs on three nodes, totalling a test run of 36 CMS jobs, lasting also around 50 minutes. With the idle graphs, a drop of 2 watts can be seen for both drives in the middle of the figure. This is most likely due some stand by mode, which is activated after fixed wait period. The overall power need of an NAS appliance is increased by 2 watts after adding four SSDs to the setup and 7 watts after adding four HDDs. The power need does not increase notably under load with SSDs, but HDDs consume an additional 6 watts. Because the small overhead of loaded versus idle case with SSDs, the increased power need of other components, excluding the drives, can be thought as minimal. Rough estimation would be that SSDs consume only half a watt of power, both idle and operational. Similar numbers for HDDs would be 2 watts when idle and 4 watts when in an operational state.

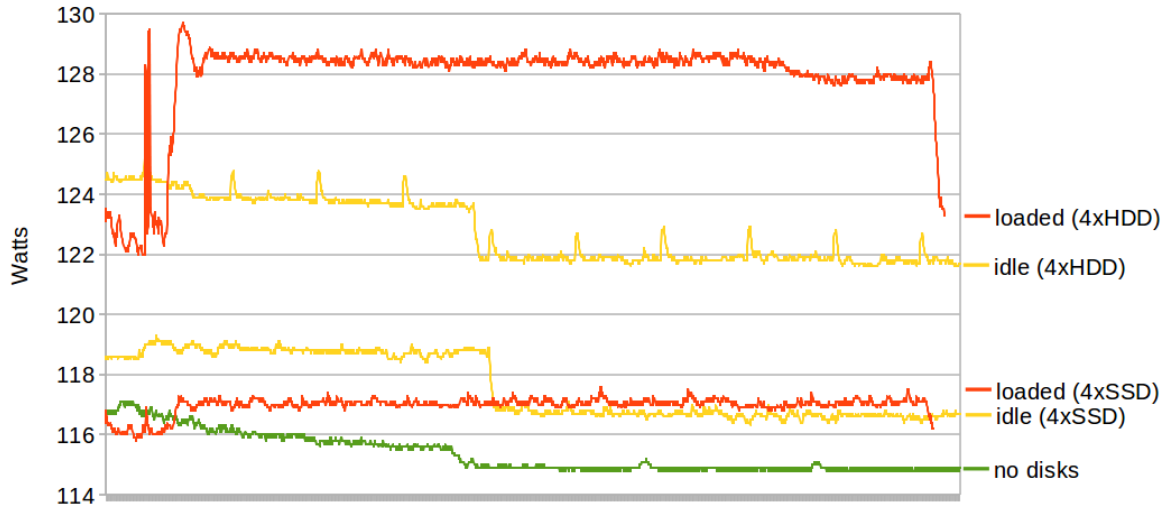

Figure D.1: The power Use profile of NAS with SSDs and HDDs.