SSD Performance

Researchers
Sebastian Carlier & Daan Muller

Company
SARA

Supervisors
Ronald van der Pol
Freek Dijkstra

Abstract

The goal of this report is to find the most efficient choice of parameters to use with Solid State Drives for sequential read performance. Enclosed is the description of SSDs and the most significant parameters to consider which are RAID levels and file systems. The research part is composed of preliminary tests for selecting the most efficient parameters and final tests for concluding the most efficient setup on the given hardware.

The research question this report tries to answer is: ''How can SARA implement Solid State Disks in their setups to improve sequential read performance over conventional spinning disks and what parameters should be used to accomplish this?"

Acknowledgments

For their expert opinions:
Freek Dijkstra
Ronald van der Pol
Mark van de Sanden
For lending us the Areca controller
WebConnexxion

SSD Performance

Table of Contents

Introduction

Solid state storage is not new, it has been around for a while in the form of flash storage.
Flash storage has always had a major drawback of being too slow to be usable in a day to day environment. With the newer technology available today flash storage does become a viable option for everyone. This is where our project comes in, SARA is working on streaming video to walls made up out of multiple Full-HD screens. Some of the setups they are proposing are using up to twelve screens. The point of these screens is to combine them into one screen and therefore be able to have a large overview of a video or picture, but still be able to come in close and look at the details. The problem this imposes is that they need to have systems capable of streaming very high amounts of bandwidth. There are network cards and processors that can keep up with their needs, but so far hard disks have proven to be slow in comparison. The linear growth all the components have gone through over the last years seems to have affected the hard disk market less. This is where solid state storage comes in. These days it is possible to get very high speeds and reliable storage out of flash media. SARA has therefore asked us to research the options they have in getting the most out of a specific set of SSDs. Together with our supervisors Ronald van der Pol and Freek Dijkstra we have set a list of parameters which we can tweak to find out which will give maximum performance for our specific goals.

In this research paper we have explained our research methodology and the tests we have performed. We have the results for our testing and the conclusions we have reached with these results.

SSDs

It was clear when we started this project that the focus would be on SSDs, but what is an SSD and how should we approach our research. To answer these questions we had to do some research into the architecture of the solid state drive and find out where these drives are stronger than conventional spinning disks, and if these relative newcomers have any weaknesses compared to the spinning disks that have enjoyed constant improvement over the last fifty years.

General Architecture of a spinning disk

To be able to really grasp the difference between spinning disks and solid state disks we first have to make clear what exactly goes on when an end user requests a file from storage.
First of all, we will make this clear in the event of a conventional spinning disk.

A disk is made up out of a number of magnetic platters that rotate inside the casing. Most disks have between one and up to three of these disks stacked on top of each other. Every entry that can be looked up by the IO subsystem of an OS knows where to look by an identifier made up out of a couple of different parameters. First of all, let's assume that the disk is made up out of multiple platters, in that case our file is on a specific cylinder, this cylinder is in turn made up out of circular tracks and next these are divided into sectors. Therefore, for every file or even part of a file the file system knows these parameters and is able to retrieve the file. Luckily, file systems do not need to know all these parameters anymore, todays hard disk controllers take care of the parameters and assign every block a block number which uniquely identifies it. This also made it possible for hardware designers to apply a few techniques that would speed up the disk access. Usually a disk would write files on the magnetic material sequentially, which in turn meant that a single large file would be on a couple of blocks on the disk that are close together. Because of the

nature of the spinning disk this posed a problem. A large file could not be read or written quick enough because the read or write head would have to wait for the next block to come flying past before it could write to it. This because read and write heads are not quick enough to read 2 adjacent blocks in one single move. But because the file system was no longer responsible for the specific place on the disk, the designers had a solution. They would have a different layout on the disk. Block one, two and three are no longer next to one another, but rather in such a position that block two could be read right after block one. This way the file system could still use subsequent blocks to write its data, but file access was much quicker.
The next problem is that it is very possible that an end user is trying to retrieve a file from the start at the outside of the disk (because hard drives are written from the outside to the inside) where the OS would normally be stored and next the user might try to access a file created recently, which on a mostly filled disk would be along the center of a platter. This means that in turn the read and write heads of the disk need to be positioned on different parts of the disk. Even the fastest disks made today still have a small latency between the file being requested and it being available at the disks interface because of this movement that is necessary.

General Architecture of an SSD

One of the first things we needed to get clear when we started working with SSDs was the architecture. One of the big problems in designing SSDs is the fact that they need to be backwards compatible with current systems. This meant that the engineers had to make an SSD that would behave like a spinning disk does. Basically this means that they employ the same separation in blocks as the spinning disks do. But because the inside of an SSD and a spinning disk have almost nothing in common, this is quite a challenge. In this chapter we will explain some of the design choices that were made.

First of all, an SSD is made up out of blocks, in most current implementations disks are made out of blocks of 512 KB each, these blocks then consist of pages, which are usually 4 KB each.

A page is the smallest writable area on an SSD, however SSDs do not have the possibility to erase a page, the smallest erasable area on and SSD is a whole block (512 KB, 128 pages)
This introduces the problem of erasing / rewriting small files. On a normal spinning disk a file can simply be deleted or overwritten time after time, but because an SSD will only delete a complete page algorithms have to be in place to make sure no data is lost.
Basically what an SSD will do is move data around a lot to have information packed tightly into blocks, thus creating blocks that can be erased. This will result in the SSD being fragmented more than you would like, but because there is virtually no seek time, and a lot of blocks can be accessed simultaneously this is not a problem. Current file systems however do not take into account that these disks have such little access time, and therefore file systems still take the problems spinning disks used to have into account while in use on SSDs.

I/O requests[1]

Now, the difference in accessing files between a spinning - and a solid state disk is clear. A spinning disk needs to physically move parts to grant the end user or application access to a file it is requesting, an SSD on the other hand does not have this problem and can access any block on the disk in the same tiny amount of time.
To minimize the problem of reading files from spinning disks, operating systems have a system called the IO scheduler. The IO scheduler will keep track of all requested files and put the in a queue. The files with block identifiers close together get queued close to each other so they can be fetched from the file system sequentially, hereby limiting the need for more spinning and moving of the read and write heads across the disk. Even blocks that are requested while the queue is already being processed are allowed to jump the queue. This posed a problem, if a file is heavily used, a lot of read and write requests will be jumping ahead in the queue, which results in the problem that in some cases blocks that are positioned at the other end of the disk have to really wait a long time until they get the attention they require. An application waiting for that specific block on the other end of the disk will be kept waiting for a long

---

1. http://www.linuxjournal.com/article/6931

time, thus bogging down the system unnecessarily. The IO scheduler that worked in this way was included in Linux' 2.4 kernel, it was dubbed the Linus Elevator, because of the similarity to an elevator in trying to be of service to a tall building.

Another way to go is by using another system called the Deadline IO scheduler, the deadline IO scheduler is very similar to the elevator scheduler, but it introduces two extra queues next to the normal one. The write FIFO and read FIFO queues. These queues contain only one type of operation and are sorted by the time new operations are added to it, so there is no jumping the queue in this model. Both queues have a timeout set for the operations, when the timeout is reached for the top operation in each queue, the IO scheduler stops feeding data from the normal elevator like queue and runs the operation that would not get the desired attention if the normal queue was not interrupted. Usually, the timeout for reads in this system is 500 ms, and it will allow up to 5 seconds for a write operation to be completed.

This of course is a nice solution for the problem in the Linus elevator, but it is not a solution to all problems. Right after the almost forgotten read was allowed to jump the queue, when the IO scheduler is busy going along the business it was doing earlier, chances are that the same problem arises again, and a new block right next to the previous one, but also on the other side of the disk needs to be read. This would result in another 500 ms delay until the read operation is allowed to jump the queue again.

This is where the anticipatory IO scheduler comes in, it is exactly the same as the deadline scheduler, but with the addition, that it waits for 6 ms after the forgotten read action was performed to see if another read action is required from that same area, which usually will be the case. If somehow it is not the case, only 6 ms will be wasted.

Wear Leveling

The other problem SSDs bring to the storage world is wear, theory says a NAND flash block can be written and erased about 100.000 times each,

this means that an SSD has a limited lifespan. The way the controller tries to get as much life out of the SSD as possible is through wear-levelling. A brand new SSD which has never been written to will be able to write files at amazing high speeds, but a lot of users started noticing that over time the write speeds were dropping badly. This is because the controller will always start writing a new page in the next available spot, no matter what block it is on. So if you write a 16 GB file to a 160 GB SSD 10 times, the whole disk will have been occupied once, it is only when the disk has been fully written once that wear levelling will start to work and make a difference in speed noticeable.
The problem is that when an SSD has been fully used once, all the blocks are occupied by at least a few pages of data that are not marked for deletion. This means that before any new page can be written to a block, this block needs to be freed up of pages and deleted first.
These pages that are not supposed to be deleted yet are stored in memory or cache on the disk and are written together with the new information. On some earlier controllers people actually noticed that these disks were noticeably slower on using small files than they were on large sequential operations. The problem in some cases turned out to be an implementation problem more than a real hardware problem. Because vendors wanted to be able to claim the highest throughput on their disk, the controller firmware was tuned for this kind of performance rather than good overall performance.
Because this doesn’t normally happen on spinning disks, the file systems used are not designed to eliminate this behaviour. And the introduction of TRIM will make sure that the file systems do not need to be changed drastically.

TRIM

The introduction of TRIM was a change in the ATA specification specifically for SSDs, TRIM will allow a file system or OS to notify and SSD of which pages / blocks can be deleted the SSD can in turn run these operations before the empty spots on the SSD are needed for new writes. This means that the write speed will not drop as far, because there will always be empty pages / blocks to write to (assuming the partition has some free space available).

It does however have a downside, if you have accidentally deleted a file yesterday, but you have only noticed today, chances are that TRIM has already gotten rid of the data in the block. Any undelete / recovery program will not be able to find your data anymore because the SSD has physically removed it from the disk whereas normally the page will be marked for deletion but not actually be deleted until overwritten.
Most calculations show that in a heavy usage situation (i.e. an email server, bittorrent / usenet enthusiast) todays SSDs will wear out in about 5 years. For a consumer this would sound like a bad thing, but in offices a server or desktop computer will usually be written off long before that.

RAID levels

Most people reading this document will know that there are differences in RAID levels. But we feel that maybe not everyone is familiar with all the different options. Because we used a fair amount of different RAID levels and therefore have done some research in the matter we feel we need to touch on this subject in our report.

RAID

RAID is an acronym for Redundant Array of Inexpensive Disks. The acronym explains the reason why these systems and algorithms were invented. Reliable disks used to be even more expensive than they are today. It was almost impossible for any small business or home user to have a reliable or very fast disk at an affordable price. RAID provided the option to combine a number of normal desktop disks into an array. There are quite a few different types of arrays ranging from speed to data security and a compromise between both. We shall explain the levels we have used shortly.
(note: the industry now explains the 'I' in RAID as Independent rather than Inexpensive. Where many large corporate setups in RAID are not made up out of inexpensive disks at all.)

RAID-0 (Stripe)

The zero in RAID-0 is jokingly (but also quite seriously) referred to be the number of files you will be able to retrieve from your array when a single disk would fail. RAID-0's algorithm writes every subsequent block to the next available disk, the size of these blocks are defined by the stripe size in the setup. The advantage for this setup comes from the fact that a spinning disk will have to recover from a write and is inaccessible for a short amount of time. Because these delayed access times are unwanted, using a different disk for each subsequent write will eliminate the time the OS will have to wait for the next block of a file to be written. When one of the disks in a RAID-0 array is damaged beyond repair it means that a

large number of blocks will be lost. Recovering a file system where every Nth block is missing (where N is the number of disks in the RAID-0 Array) is next to impossible.
It is clear that RAID-0 is aimed to provide maximum speed rather than security.
A RAID-0 array will provide all the storage of every disk used.

RAID-1 (Mirror)

RAID-1 is mostly aimed at security, but has some extra performance as a nice added benefit. RAID-0 effectively does nothing more than perform every write to all disks at the same time. This ensures the availability of your data in the event of a disk crashing. The added benefit of speed comes from the fact that every block is available from every disk, and therefore multiple reads can be performed sequentially.
RAID-1 is aimed at security and has an added benefit of read speeds increasing.
A RAID-1 array will provide only the storage of the smallest disk used in the array. The rest of the disks may be larger, but will never be used beyond the size of the smallest disk in the array.

RAID-1+0 / 0+1

RAID-1+0 and 0+1 are a combination of RAID-1 and RAID-0, combining their benefits. What this means in practice is that a set of disks running in RAID-0 are subsequently set in an array in RAID-1 or the other way around. This means that there can be a disk failure without the data being lost because there is never a single piece of data which is only written to one disk.
RAID 1+0 / 0+1 is a balance of speed and security.
These RAID levels provide half the storage of the four disks that are used.

RAID-4

RAID-4 is a whole new system compared to the previous levels, in RAID-4 every block is written to a different disk similar to what happens in RAID-0. The big difference is that in RAID-4 a parity block is calculated

for each set of blocks. This parity block is then written to a separate disk. This makes sure that the array can handle a disk failing, because the data on any disk can be recalculated by using the parity on the parity disk. Because RAID-4 combines a striped set of disks with parity, it is a combination of speed and security.
A RAID-4 array will provide the storage of all the disks involved minus one disk for parity.

RAID-5

RAID-5 is basically the same algorithm as RAID-4. The only difference is that RAID-4 uses a single disk to store all parity information. This means that every parity block is written to the same disk. This proved to be a bottleneck in the algorithm and was slowing down the array. Therefore in RAID-5 the parity is distributed over all the disks. This eliminates the bottleneck in RAID-4 and provides extra performance at no extra cost in speed or amount of disks.

A RAID-5 array will provide the storage of all the disks involved minus one disk worth of storage for parity.

RAID-6

One more RAID level that caught our eye because it was advertised on the box of the Areca controller. RAID-6 basically does the same thing RAID-5 does, but in RAID-6 all the parity blocks are stored on two disks instead of just on one. This means that it can survive two separate disks failing without losing any data.
A RAID-6 array will provide the storage of all the disks involved minus two disks worth of storage for parity.

File systems

Previous research clearly shows that a file system noticeably affects disk performance, therefore it was one of the main parameters to consider when conducting our measurements. Due to time constraints we were forced to limit the scope of our project to five following file systems for preliminary tests:

- Ext4
- Btrfs
- Nilfs2
- XFS
- ZFS

None of these file systems is specifically designed for SSDs, although when SSDs become more common on the consumer market we are sure to see some development in that area. Before conducting our research we considered using JFFS2, UBIFS or LogFS as these are the file systems specifically designed for raw flash memory. They are unfortunately unsuitable for regular SSDs, because those type of disks appear as conventional drives to the system. And not as separate chips of flash memory without a controller like the SSDs do.

This chapter is a brief introduction of the above file systems and explains why they were used. Nevertheless this paper is about SSD sequential read performance, so for in depth information about file systems please refer to the mentioned sources.

Ext4

We decided to test the newest file system from the ext family, because it is one of the common Linux file systems. The final version of this file system was included in the Linux Kernel 2.6.28 on 25 December 2008. The main philosophy behind ext file systems is that they are backwards and forward compatible. Basically each ext file system introduces new features as compared to it's predecessor. You can find a list of supported features under the following link[2] . An important feature of the ext4 file

system, when it comes to sequential reads, is that it uses extents, which limit file fragmentation on the disk by allocating contiguous disk space for that file[3] . This is more beneficial for spinning disks rather than SSDs because of their design.

Previous benchmarks show faster read speeds over the ext3 file system[4] . This was reason enough for us to consider ext4 for our tests rather than its predecessor.

Btrfs

Btrfs also called 'butter-fs' is the newest file system that we tested. It is a GNU licence response to ZFS. At the time of conducting our tests it was already included in the Linux 2.6.29 Kernel. Oracle claims it is still in development[5] . This is good news considering that the company owns Sun (which developed ZFS) since April 2009 and therefore might have stopped Btrfs development and focus on ZFS.

Our benchmarks were performed on v0.19. It is still clearly stated when installing this file system that it should not be used for important data at this stage. Nevertheless promising benchmarks and good opinions from specialists (*Theodore Ts'o, has stated that "ext4 is simply a stop-gap and that Btrfs is the way forward"*)[6] made us choose it. Btrfs developers claim that this file system is being designed to scale for systems with large data storage[7] . This is mainly due to the architecture of this file system.

It uses B+trees data structure. All of the file system meta-data is stored in the tree structure. Keys which are comprise objectid, type of data and

---

2. http://ext4.wiki.kernel.org/index.php/Frequently_Asked_Questions#File_System_Features
3. http://en.wikipedia.org/wiki/Extent_%28file_systems%29
4. http://www.phoronix.com/scan.php?page=article&item=ext4_benchmarks&num=4
5. http://oss.oracle.com/projects/btrfs/
6. http://en.wikipedia.org/wiki/Btrfs
7. http://btrfs.wiki.kernel.org/index.php/Main_Page

offset of data are stored in internal nodes. Data is only stored in the leaf nodes of B+trees, the Btrfs uses a clever way to store large data files, by not keeping them in the B+trees and using extents in the leaf nodes which point to those files. It can also store different types of data on the same physical block which is a fresh approach saving disk space.

Btrfs seems to be a promising file system, possibly substituting the Ext family file systems in the future.

ZFS

ZFS was introduced by Sun Microsystems. Its design differes a lot from all other filesystems. Apart from the classical task of storing data it also takes the role of software RAID.

ZFS was not a first choice for our tests, but SARA was interested in the result so we decided to use it. We tried to keep our benchmarks as homogenous as possible to easily pin point the reason for our findings, unfortunately it was not possible in this case.

ZFS does not comply with GPL standards and therefore can not be included in a Linux kernel, so it had to be tested in FreeBSD. We were not sure how much of an influence could be caused by changing the OS. Another reason was that ZFS has its own type of RAID system implemented in it.

For the purpose of our research we used RAID-Z bundled with ZFS. The idea behind it is similar to RAID-5, it also uses parity stripes spread over disks. The innovation in RAID-Z is that it can use variable stripe size, because it is integrated with the ZFS file system[8] .

The main advantage of checking ZFS' performance was to test if a filesystem with a very different design approach could achieve a respectable score.

---

8. http://blogs.sun.com/bonwick/entry/raid_z

XFS[9]

This file system was first introduced in 1994 by Silicon Graphics Inc. in the IRIX operating system. It is also included in the Linux 2.6.17 kernel. It is a journalling file system designed for scalability. It was chosen by us because some of the sources claimed it achieves high performance and handles large files very well. Same as Btrfs and ZFS it implements B-trees. We did not anticipate for this file system to achieve the best performance with SSDs, due to its age. Nevertheless we wanted to check previously stated claims.

Nilfs2[10]

The name stands for New Implementation of a Log-structured File System. The second version was included in the Linux 2.6.30 kernel. It also utilizes B-trees and copy-on-write for journalling. It is worth noting that NILFS2 enables immediate recovery of corrupt data.
Although we were not concerned about data corruption during our project, we still wanted to benchmark a logging file system, for comparison. We also found claims that it achieves amazing performance if mounted on an SSD[11] .

---

9. http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf
10. http://www.mjmwired.net/kernel/Documentation/filesystems/nilfs2.txt
11. http://www.linux-mag.com/cache/7345/1.html

Predictions

Before we started testing we agreed with our supervisors that we would give our predictions on the outcome of our tests. This way we can force ourselves to think about the different setups and their implications.
We agree that running RAID-0 over as many disks as possible will allow the IO controller to reach its maximum throughput. Because in this setup the algorithm used by RAID-0 forces every subsequent block of data to be on a different disk. This way the IO buffer will always be filled and read speeds are optimized.
We also figure that RAID-5 would be one of the slowest in writing because for every block written to a disk will have to have parity calculated and written before the next ones can be processed. Read speeds from RAID-5 should be pretty good, because taking the parity calculations out of the equation while reading RAID-5 basically means you are reading from a RAID-0 array omitting parity stripes which should be very quick.

Our other theory is that hardware-controlled RAID will always be faster and more efficient than RAID implemented through software. This is because the processing required by the RAID setup can be handled by a processor designed for it specifically, and not by the more universal CPU on the main system.

We have to be very specific on the subject of block sizes, because there are a lot of different ways in which block sizes are possible in our setups. First of all there is the block size used by the file system. Secondly there is a block size on the RAID arrays, and lastly there is a block size in the SSD itself. For clarification the RAID block size is called a stripe size, and the block size on the disk is something we cannot change.

Test Setup

Before we started the project we were clear that we would be testing a lot of different combinations of parameters on a single system. Therefore we would need a benchmark that is easy to repeat and reliable. We also wanted the test environment to be able to run on Linux and lastly we wanted to be able to automate the tests.
In a previous project one of us had worked with the Phoronix test suite. This is a scriptable benchmark suite built up out of modules of independent, freely available tests. The test we wanted to focus on is the performance we could get in the sequential reading of large files from the SSDs, we therefore decided to go with IOzone as our benchmark. IOzone focuses on read and write performance and we had the option to set the file size in our benchmark. We ran an IOzone sequential read test of an 8 GB file 3 times for every chosen setup.

The server

Before we started our project Sara had already purchased a server and the SSDs we could use for this test, we had full control of the system for the entire duration of our project, so we never had to plan our tests or reboots to fit any other schedule.
The hardware we had at our disposal:

Base: Dell Poweredge T610 inc. 8 disk SAS backplane
CPU: Intel Xeon CPU X5550 (Quad core + hyperthreading)
RAM: 6 GB DDR3 @ 1333 MHz
RAID: Dell PERC 6/i SAS controller
HDD: Dell 2.5” 7200 RPM 250 GB (x2)
HDD: Dell 2.5” 7200 RPM 160 GB (x5)
SSD: Intel X25-M G2 160GB (x5)
LAN: Intel 10GbE Network card (Optical)

Operating system

Because we have grown quite familiar with Ubuntu over the last 5 months we decided to run our tests on Ubuntu 9.10. This posed a problem since we were also interested in the performance of the ZFS file system. Therefore we also had a FreeBSD 8.0 installation at our disposal.
To make sure we would not have to reinstall or restore an image of our installed test systems we used the disks that were shipped with the system as our OS partitions. The reason we would have to reinstall or restore the installation if we had installed the OS on the SSDs is that every time we rebuild a RAID array with a different number of disks, on a different controller or in a different setup the system would wipe our disks and leave them empty.
Testing on Ubuntu proved to be straightforward. Installation of the test suite was no challenge either. After familiarizing ourselves with the 'mdadm' commands in Ubuntu to create and manage Software RAID arrays we could start our tests. The first results were not as promising as we had hoped for, we then asked ourselves if we were sure that the test was actually running on the RAID array of SSDs and not just the disks the OS was installed on. We ran the test again and simply checked the activity LEDs on the backplane of the server. This quickly showed us that the SSDs were not being accessed at all. A simple change in the script for our test suite allowed us to change the working directory for the test to the RAID array of SSDs and the results started making more sense.

Preliminary tests

Our test approach was to change one parameter at a time to note its impact on sequential reads and chose the most efficient ones for the final benchmark setup. If we were to consider every combination of file system, file system block size, RAID variations (hardware, software and different stripe size) it would amount to about 1500 tests. This equals about 31 days of continuous testing, which in our time span was impossible to perform. Our supervisors agreed to test the most promising combination of considered parameters.

Firstly we checked which file system would be the most efficient for our preliminary tests. The graph below shows that on a single disk Btrfs has a significant performance increase over the default ext4 file system. Nilfs2 and xfs performed similarly to ext4. These results were obtained in Ubuntu 9.10. Almost all of the preliminary tests were done with Btrfs. The early development of the file system and the impressive results convinced us to use it in the final tests. Since we needed to test different RAID setups we did not check ZFS performance at this point as it uses RAIDz and it would not be suitable for preliminary tests. Nevertheless in later benchmarks ZFS performed close to Btrfs and we decided that it will also be included in the final benchmarks.

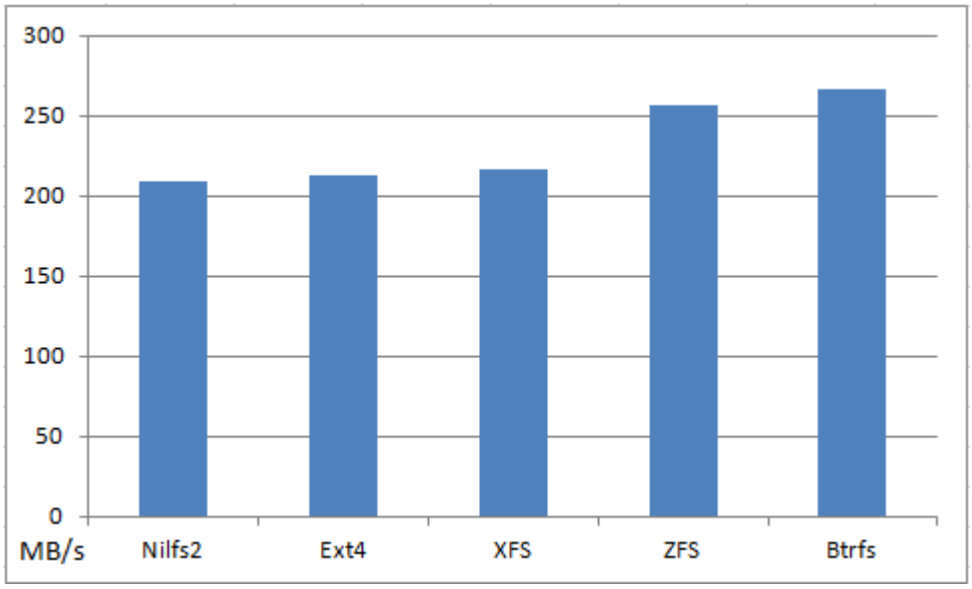

Figure 1: Sequential read performance of file systems on a single SSD.

The next parameters that would have a considerable influence on the results were:

- the RAID level,
- number of disks in a RAID array,
- software or hardware RAID,
- RAID stripe size.

These tests were performed on the Btrfs file system under Ubuntu 9.10 with the Dell PERC 6/i controller.

Testing different RAID levels resulted in a noticeable performance difference. The figure for a 5 disk software RAID-0 setup came up to 657

MB/s, in theory this speed should be possible with only three disks (because 3 x 267 = 800 MB/s) so we decided to see where the bottleneck was, would a RAID-0 array of two, three or four disks provide a similar result? That way we can test if the bottleneck is in the disks or the controller.
After running a two disk software RAID-0 array we concluded that it proved a 89% performance gain over a single disk without any RAID implemented (505 MB/s). To see if this trend would continue we then did the same test on a 3 disk setup on the same software RAID-0 setup this showed us that the growth was substantially less than we had hoped. It only managed to reach 634 MB/s. It is still a performance improvement worth the investment, but not nearly as much as we had hoped.
We figured that the trend was not going to continue in the previous steps, but we ran the 4 disk RAID-0 setup to be sure. It ended up at 671 MB/s. This gave us the result that a four disk software RAID-0 setup is quicker than the same setup on 5 disks. The only explanation we can give for this behaviour is that five disks provide more overhead than performance gain compared to the four disk setup. We can see these results in a graph looking like an inverted parabola, with the top of the arc at 4 disks. However, it is also possible that a six disk RAID-0 setup would give a performance gain once more, but we could not test this further. The graph below shows the scaling of RAID-0 from 2 to 5 disks. It also includes the outcome of the tests done on 7200 rpm spinning disks. Single spinning disk sequential read performance on btrfs is 57 MB/s, and this value is almost 5 times greater with a 5 disk software RAID-0 reaching 274 MB/s. The spinning disks scale ideally.

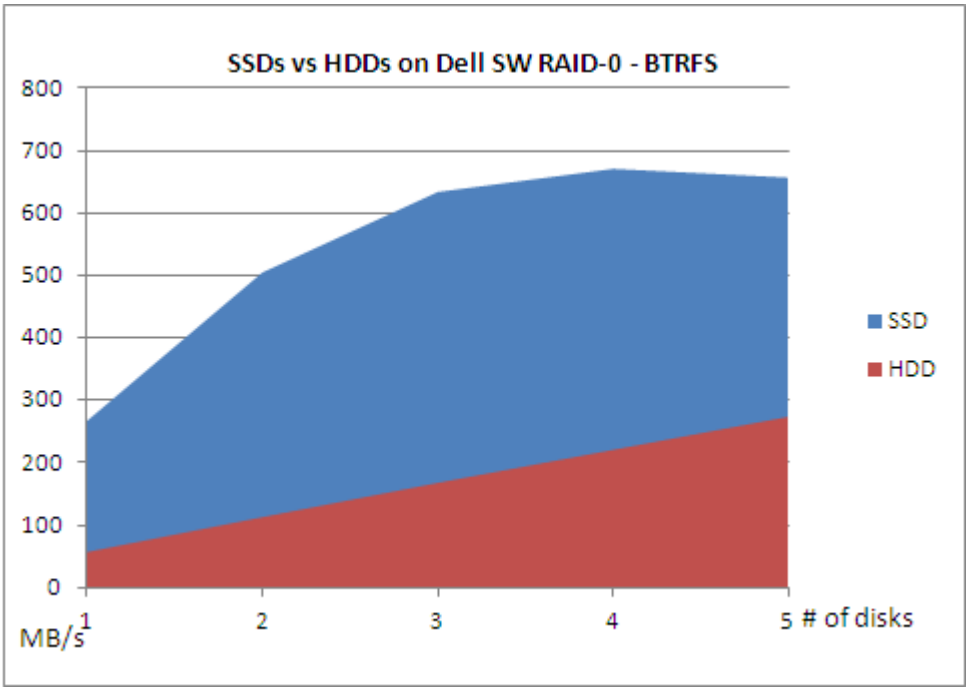

Figure 2: Performance of Btrfs on SSDs and HDDs on software RAID-0.

We continued with different RAID levels, to choose the fastest one for the remaining tests. The approach here was to take the biggest amount of disks we had at our disposal to see how much read speed our system can handle. Obviously RAID-10 and RAID-01 is only possible with an even number of disks so with these options we tested 4 disks. As was stated previously RAID-0 gave the most promising results with 671 MB/s sequential read speed on 4 disks. As we predicted RAID-5 on 5 disks test also returned a good result of 607 MB/s. In our opinion the 64 MB/s loss in performance is worth the security RAID-5 provides. Nevertheless our goal was not data security, but efficiency so we decided to test RAID-0 with 4 and 5 disks in our final tests.

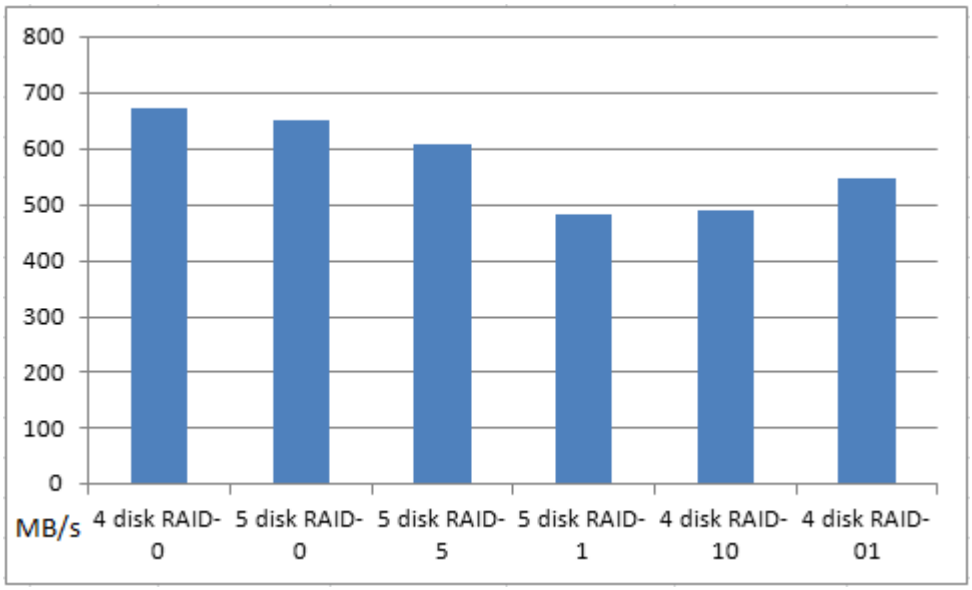

Figure 3: Performance of different software RAID levels with Btrfs on SSDs.

We decided to do some tests on the hardware RAID levels on the Dell PERC 6/i controller expecting better results. First of all we did a few tests on the most promising hardware RAID level, the results on the software RAID had shown us that the quickest option was a four disk RAID-0, our theory was that a five disk should be quicker, we tested both setups on hardware RAID-0. To our surprise, the RAID controller was doing a bad job at feeding data to the system, the maximum number reached by this setup was only 505 MB/s, exactly the same number as a two disk software RAID-0 would manage. We decided to test hardware RAID-0 on four disks and two disks to see how it scales. The numbers were disappointing, the first test gave us 501 MB/s and the second one 453 MB/s. We decided not to test 2 disk performance, but instead check if the stripe size in hardware RAID and the read ahead option would increase this poor performance.

We tested the read ahead option on a 2 disk hardware RAID-0. It did not help increase the SSDs performance, dropping it from 453 MB/s to 421 MB/s. We can only conclude that read ahead is not designed for solid state drives.

Stripe size testing was performed on RAID-0 5 disks. The graph below shows that the default stripe size of 64KB is the fastest setup with 505 MB/s. We see a decrease in performance when selecting a 16KB stripe

size, because the controller performs more operations reassembling the file that is read. Changing the stripe size to values of 256KB and 1024KB (from the 64KB default setting) lowered the disks' performance, respectively to 471MB/s and 362MB/s. Clearly the Dell PERC 6/i controller does not handle big stripe sizes efficiently.

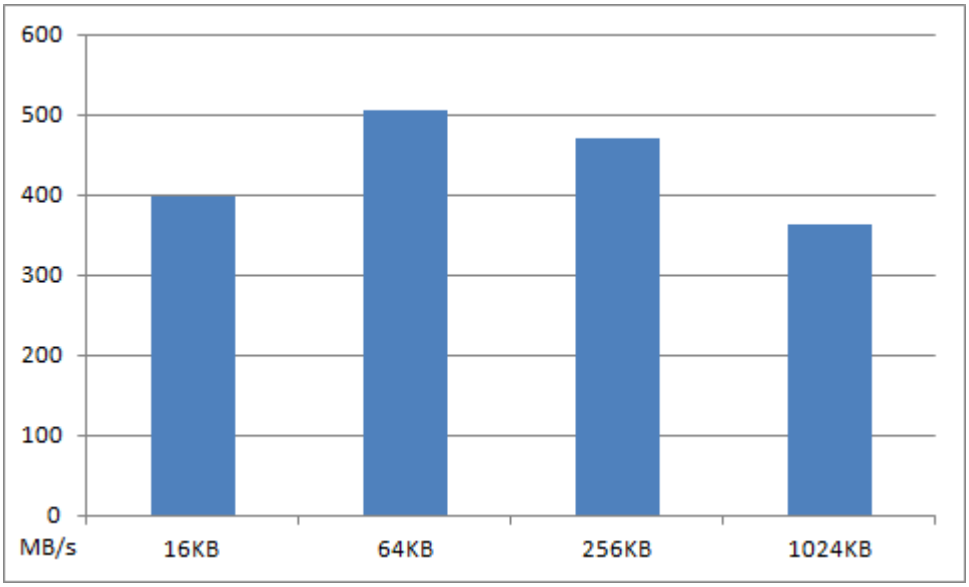

Figure 4: Hardware RAID-0 stripe size
on 5 SSD array with Btrfs.

To conclude this set of preliminary tests we decided to perform the final tests with software RAID-0, default 64KB stripe size with the Btrfs file system and ZFS file system with RAIDz. Before proceeding we still needed to check if the file system block size would have an influence on our results.

Testing the different block sizes had to be done with ext4 as changing the block size in btrfs failed. When we set the file system block size to 1KB, changing it from the default 4KB, we noticed a slight decrease in performance, about 4MB/s (dropping from 213 to 209MB/s). The 2% difference was not big enough to use the file system block size as one of the parameters in our final tests. Although changing the file system block size might have a more noticeable effect on spinning disks, just because their read speeds are much slower.

At this point our preliminary tests were done. Because of the scaling issue in software RAID-0 with Btrfs we decided we should test the following file

systems under Ubuntu 9.10 on software RAID-0 with a varying number of SSDs:

- btrfs
- ext4
- nilfs2
- xfs

This was done to make sure that Btrfs was not responsible for the poor scaling. Moreover we performed some of those tests with the same file systems on spinning disks. Lastly we decided to check ZFS performance on RAIDz with a variable number of disks.

Final tests

We already acquired the Btrfs performance in preliminary tests and the performance of the remaining file systems on a single SSD. The following tests were only performed to show that Btrfs was not the cause of poor scaling.

Ext4

Although ext4 scored only 213 MB/s (54 MB/s worse than Btrfs) we tested it on all the possible software RAID-0 arrays. With 2 disks - a score of 395 MB/s - it almost doubled the single disk performance. The 3 disk array was disappointing as it reached only 503 MB/s, a similar to 2 disk RAID-0 Btrfs. The 3 disk array performed 167 MB/s loosing 46 MB/s per disk. The 4 disk array gave an increase in performance of only 56 MB/s, scoring 559 MB/s. The 5 disk array read data at almost the same speed as the 4 disk, scoring only 578 MB/s. Adding the 4 and 5 disk to the array is not worth the increase in performance, as they add the same throughput (or less) than a customer grade spinning disk. The graph below is a comparison of ext4 RAID-0 performance on SSDs and spinning disks.

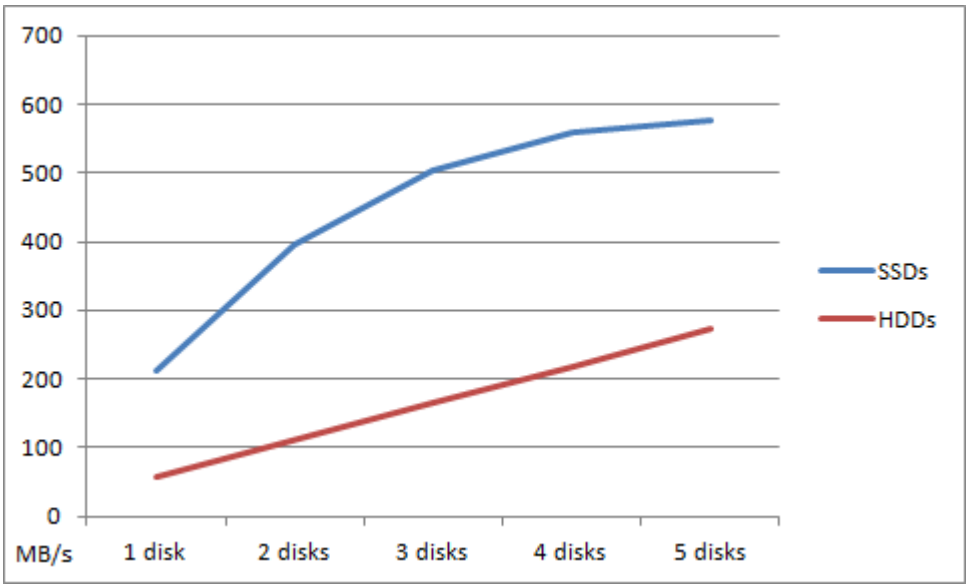

Figure 5: Ext4 file system software RAID-0 scaling

over SSDs and HDDs.

Nilfs2

Nilfs2 performed similarly to ext4 with a slight decrease in performance, the one SSD test resulted in 209 MB/s read speed. On 2 disks and more it scaled similarly to ext4. The maximum read speed was achieved with 5 disk RAID-0 with the speed of 554 MB/s. These results are not a surprise considering that the file system is focused on data security, not performance. Considering the read speed on one spinning disk of 53 MB/s, nilfs2 scaled well on spinning disks achieving a maximum read of 250 MB/s for a 5 spinning disk RAID-0. The graph below compares nilfs2 sequential read speeds on SSDs and spinning disks. We did not test the speed on a 2 spinning disk array though.

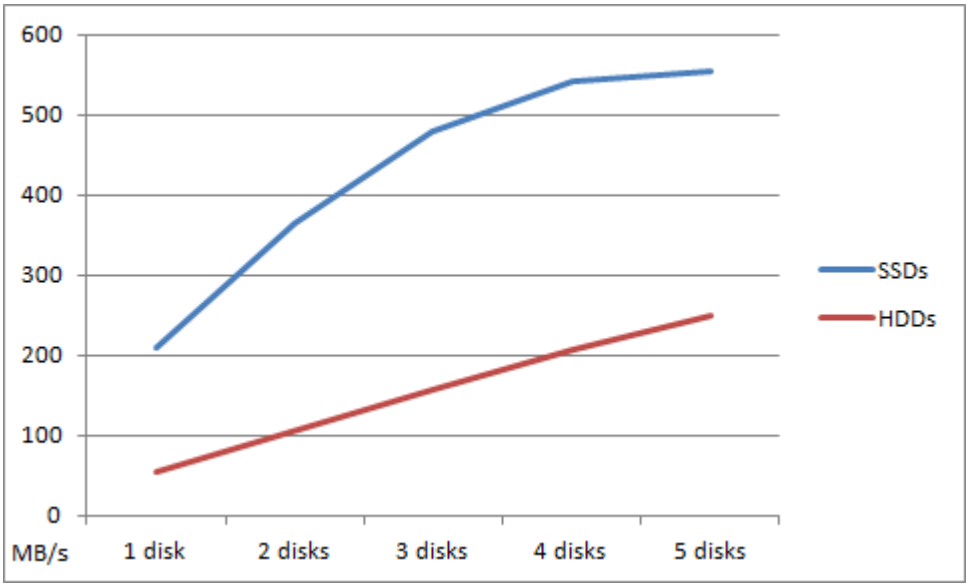

Figure 6: Nilfs2 file system software RAID-0 scaling over SSDs and HDDs.

XFS[12]

Although XFS claims to be optimized for IO performance it only outperformed ext4 and nilfs2 by a margin. With a read speed of 216 on a single SSD and similar scaling to ext4 and nilfs2 it does not come close to the impressive results achieved by Btrfs. XFS maximum performance was 583 MB/s on a 5 SSD RAID-0 array. We tested XFS on 3 and 4 spinning disks in a RAID-0 array, both tests socred respectively 158 MB/s and 215 MB/s. It is safe to conclude that it scales well on spinning disks. The graph below shows XFS performance over a variable array of SSDs compared to the two results on spinning disks.

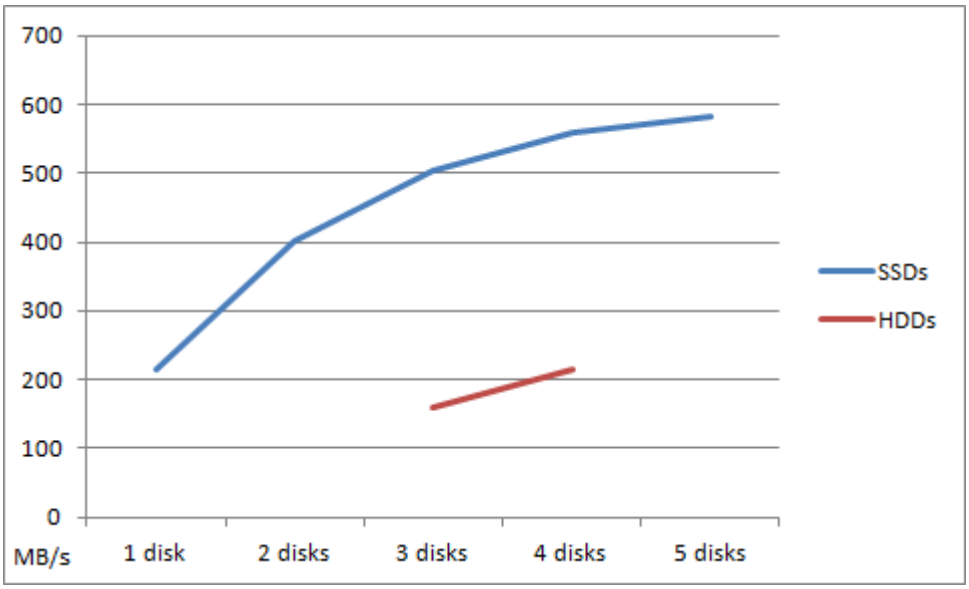

Figure 7: XFS file system software RAID-0 scaling over SSDs and HDDs.

12. http://oss.sgi.com/projects/xfs/

ZFS

After we performed all the tests in Ubuntu 9.10 we wanted to test the performance of the ZFS file system on SSDs with RAIDz in FreeBSD. ZFS performed similarly to Btrfs on a single disk with an output of 256 MB/s. On two disks it scaled worse than btrfs achieving only 454 MB/s, which is 51 MB/s slower than the corresponding Btrfs test. Nevertheless on a 3 SSD RAID-Z setup it achieved 714 MB/s which is faster than any Btrfs setup tested by us. Unfortunately, simiarly to every other file system it did not scale well in a 4 and 5 SSD array, achieving respectively 792 MB/s and 816 MB/s performance.

The graph below shows the scalability of the two most promising setups tested:

- Btrfs with RAID-0
- ZFS with RAID-Z

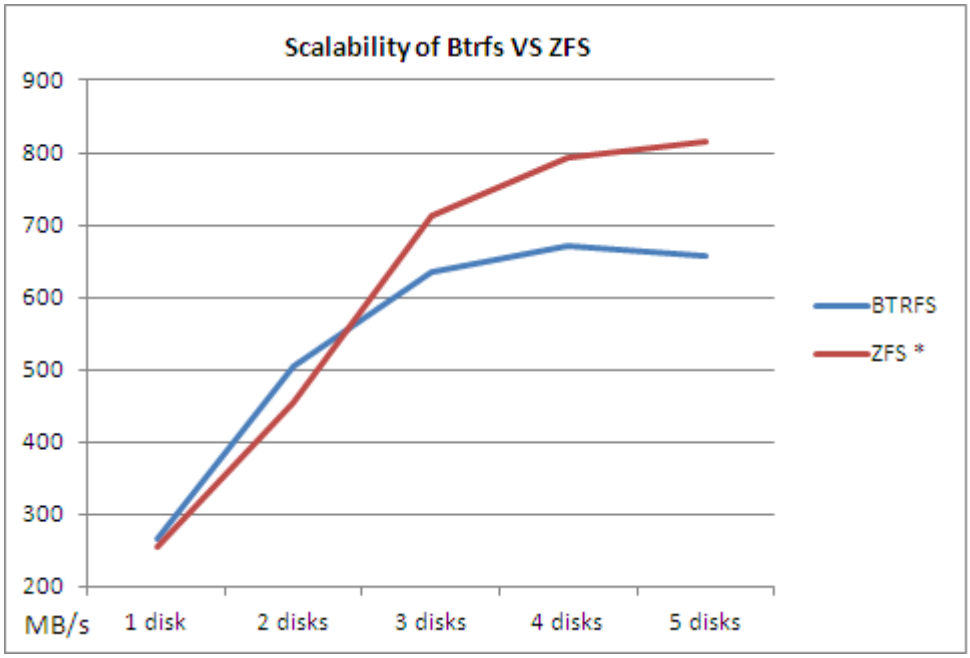

Figure 8: Btrfs and ZFS scaling on SSDs.

The graph below shows the performance of all the file systems tested on a 5 SSD software RAID-0.

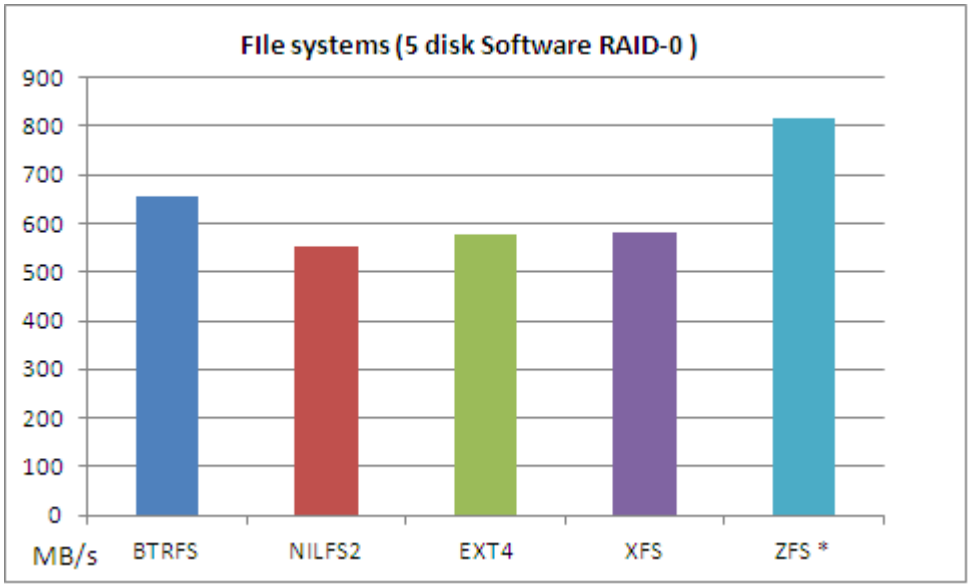

Figure 9: Comparison of file systems on software 5 SSDs RAID-0 (*ZFS on RAID-Z).

Hardware RAID versus software RAID

As we stated in the chapter on predictions, we were quite sure that any hardware controlled RAID setup would be quicker than any software implementation of the algorithm could ever be. Therefore we were quite amazed that in our preliminary tests we found that running the same tests on a hardware controlled RAID card resulted in lower performance than it did in a software setup.

We tried to change all the settings that could make a difference in our read speeds, one of the options in the Dell's PERC controller is called 'Read Ahead'. What this option essentially does in its most aggressive setting is always read as much blocks ahead of the requested read as the on-board cache on the card will allow it to. This then will always ensure that the next block that might be requested is already in the much quicker cache, and does not need to be fetched from the disk anymore.

We did not have much hope for this option; since we are focusing on sequential reads from the SSDs the files read are big enough to fill the cache on the card multiple times over. All the other options we could change on the controller were connected to the cache in writing to the disks, so these were of no use to our testing.

The numbers shown in this hardware-RAID setup gave us the impression that the hardware-RAID controller we were using had problems of processing more than about 500 MB/s in any setup. A setup of 2 SSDs in RAID-0 using the Dell PERC card showed us a throughput of 450 MB/s and getting this number up to 4 or 5 disks did not pass 505 MB/s. We can prove that there was no physical limitation in the disk throughput or the cables used because the software RAID-0 setup using 4 disks would show us a result of over 670 MB/s. Therefore we were convinced that the Dell PERC 6/i card's processing speed was limiting our throughput at this moment.

We discussed our findings with the group at the meeting halfway through our four week project, the limitations we found were not met with the amazement we expected. Some of the people with more experience in RAID setups told us that with this kind of hardware the hardware RAID controller would be slower, since most hardware RAID controllers are not specifically aimed at top performance but more to relieve the CPU from these tasks that will normally be performed just as well by a hardware RAID controller.

The conclusion we could reach from this meeting is that the Dell PERC 6/i card we were using is not meant for high performance arrays, but was designed to run with spinning disks and to relieve the host system’s processor from the RAID functions. Therefore the decision was made to try and get our hands on a different, faster RAID controller. We had to find a controller that was made and specifically designed for high speed disks and maximum throughput. After a couple of days of online research and some input from our fellow students, OS3 staff and a storage expert at Sara we decided to specifically look for a card produced by Areca. In many benchmarks and reviews comparing RAID cards these cards showed to deliver maximum performance and compatibility. (After all we would need to be able to use it in both Linux and FreeBSD.)

After we found the distributor for Areca cards in the Netherlands we checked what options were in stock and could be sent our way swiftly. We had a couple of limitations. First of all we had to find a card which could operate on a PCI-express bus and it would have to connect our disks through a SAS backplane. This meant that we did not have as many options as we had hoped. After reading a lot of reviews on the different cards we came to the conclusion that the Areca ARC 1680 series of cards would be the best bet for our setup. The other card with more than 4 ports that was in stock at the distributor showed to have a limit of about 500 MB/s in any setup. Seeing that we had already managed this speed through our software RAID-0 setup we decided to go with the faster option which was the Areca ARC 1680. Together with our supervisors we decided we would best be served with the 12 port option, it would be able to connect an additional 4 disks through an external housing.

After contacting the distributor our supervisor got them to agree to send us the card so we could test if our setup would benefit from it. They also agreed that we could send it back to them within 30 days to make sure we would get the desired performance from it.

Areca ARC 1680

As soon as we could get our hands on it, we installed the new Areca card into our test server. We could simply attach the supplied cables to our SAS backplane and connect all our disks in the same fashion as before.

Because we had swapped the card, and wanted an honest result in the tests we reinstalled our server with the same Ubuntu version we had used before.
We arranged the SSDs in a RAID-0 array of all 5 disks, this way we wanted to check if the maximum throughput we could reach before was indeed limited by the Dell PERC card we had used before.

The first numbers that came in were much lower than we had expected. The maximum throughput we had read about in the benchmarks we had found online went up to numbers between 800 and 1000 MB/s which

would be a considerable move up compared to the Dell hardware and the Linux Software RAID, respectively 505 MB/s and 671 MB/s.
We installed the same test environment and ran another sequential read performance test on 4 SSDs hardware RAID-0 with btrfs only to find that we could reach no better numbers than 447 MB/s. A number easily surpassed by a 2 disk software RAID-0 array. We tried every number of disks and block size we could manage. Again the a 5 disk hardware RAID-0 performed slower resuting in 431 MB/s. The number never surpassed 447 MB/s.

Testing software RAID-0 on the Areca ARC 1680 controller also returned disappointing results. We tested the setup with Btrfs on every number of SSDs possible. The results were:

- 1 SSD, 186 MB/s.
- 2 SSD array, 368 MB/s.
- 3 SSD array, 495 MB/s.
- 4 SSD array, 565 MB/s.
- 5 SSD array, 583 MB/s.

The graph below compares the results of software and hardware RAID-0 on the Btrfs file system with 4 and 5 disks.

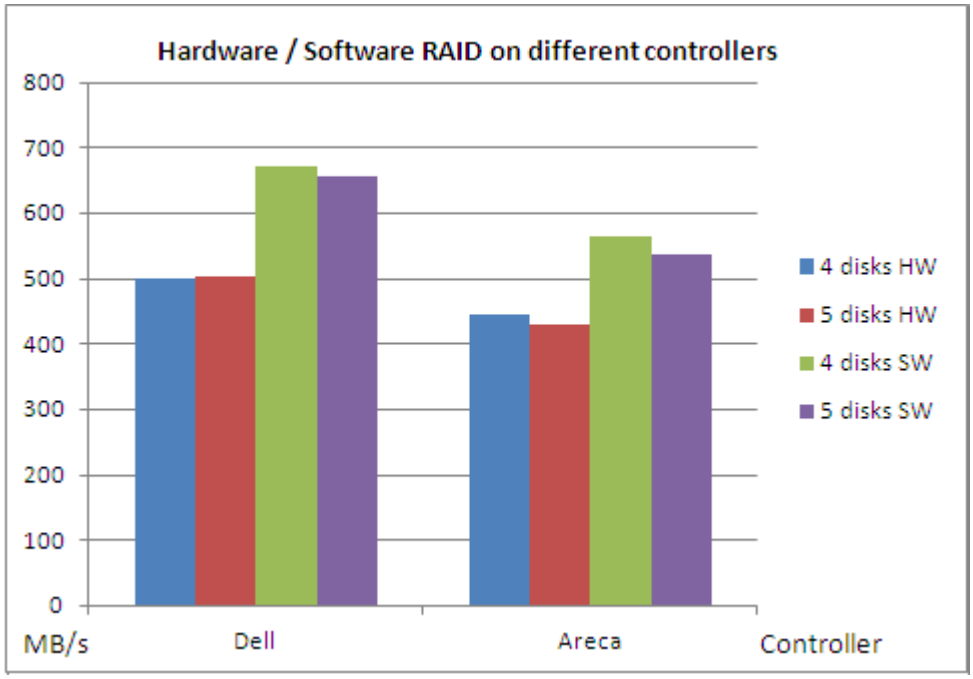

Figure 10: Comparison of software and hardware RAID-0 on the Dell Perc 6/i and Areca ARC 1680 controllers performed with Btrfs.

Before we had ordered the RAID card the distributor had sent a request to Areca to ask them if the 1680 series was fit for use with SSDs. After we had received the card there was a reply from the distributor, Areca explained that the 1680 series is not compatible with SSDs, and there is no proper solution which they could implement in their firmware. The problem is in the Intel IOP 348 CPU used on the 1680 series. A simple update in its code would be the solution to our problem, but Intel refuses to release the source code and refuses to make the changes to the firmware that are necessary. In a response to this, Areca have announced a new series, the ARC 1880. These cards are running a CPU made by Marvell and do not have the same problem. They are however not available yet.

Combining controllers

In our presentation at SARA we mentioned that using multiple controllers would get around the problem of being limited by a single controller's capabilities. One of the spectators at the presentation asked us if we had tested this with the two controllers we now had at our disposal. We did not find the time to do this yet, so we decided to try this since it could prove that the Dell PERC controller was not up to the task.

We connected three SSDs to the Dell PERC controller, because we could clearly see in the graph that after three disks the performance growth takes a big hit. We connected the two remaining SSDs to the Areca controller together with the disks where the OS resides.
Luckily, after rebooting the Linux installation we had been running since the Areca controller had arrived, the system instantly recognized the Dell controller and was able to use the SSDs.
We then created a 5 disk software RAID-0 array over the two controllers with the Btrfs file system. The first results seemed promising, and after the tests were completed we were amazed with a number of 815 MB/s out of these 5 SSDs. Exactly one megabyte short of the ZFS setup we could create on FreeBSD. Adding the proof that we were limited by our RAID controller.

Conclusions

The RAID and file system tests clearly show the advantage of Solid State Disks over spinning disks. The increase in performance can be almost 300% when using SSDs, even though their performance does not scale well on the hardware that was at our disposal.

On all the file systems we tested SSDs greatly outperformed spinning disks. At this moment the most efficient file system to use under Linux is Btrfs. Although it is still in its early stages of development at the time of writing this paper. For that reason we can not advise to use it for the storage of important at the moment, but when the final version appears we recommend trying it.

If other Operating Systems are considered and Linux is not essential, then we strongly advise using ZFS with RAIDz, as it achieved the highest sequential read speeds of all the file systems tested. Clearly the fusion of a file system and a software disk array controller handles IO operations more efficiently than any file system and RAID-0 setup. Moreover it adds data recovery in case of a disk failure.

Solid State Drives have greatly increased IO performance over spinning disks, the drives tested by us can outperform a consumer grade spinning disk approximately 5 times given the same setup. They are a worthy investment where performance is priority.

Unfortunately at the time of this research there is a lack of RAID controllers to easily support such IO performance on a larger scale. It would be ideal to test RAID controllers specifically designed for SSDs. Another good option is to test the SSDs in a setup that is already optimized for hard disk performance. There are appliances available today that are optimized to use fast disks in large arrays. These are setup out of multiple RAID controllers which can be pooled together and will allow you to connect up to 48 disks of your own choice. As we noted in our research a single controller can be outpaced by a set of SSDs.

So to come back to our research question:
"How can SARA implement Solid State Disks in their setups to improve sequential read performance over conventional spinning disks and what parameters should be used to accomplish this?"

We have concluded that first of all it is very useful for SARA to look into SSDs for their storage needs when speed comes into the picture. The parameters we would advise them to use in Linux are the btrfs file system running as many disks as your controller will scale to (while still adding significant performance with each extra disk). When BSD is an option to use ZFS is very good on performance, but the RAID possibilities of a controller are less important than the throughput it can manage for independent disks. Block sizes in file systems and RAID stripe sizes have not managed to prove any improvement over the default values recommended by the manufacturer in our tests.

References

[1] Robert Lowe. Linux Journal, Kernel Korner - I/O Schedulers, http://www.linuxjournal.com/article/6931 , February 2004

[2] Ext4 Wiki, http://ext4.wiki.kernel.org/index.php/Frequently_Asked_Questions#File_System_Features , January 2010.

[3] Wikipedia article on extents, http://en.wikipedia.org/wiki/Extent_%28file_systems%29 , 1st February 2010.

[4] Michael Larabel. Real World Benchmarks of the EXT4 File-System, http://www.phoronix.com/scan.php?page=article&item=ext4_benchmarks&num=4 , December 2008.

[5] Oracle's official Project: Btrfs website, http://oss.oracle.com/projects/btrfs/ , January 2010.

[6] Btrfs Wiki http://btrfs.wiki.kernel.org/index.php/Main_Page , January 2010.

[7] Jeff Bonwick. Jeff Bonwick's Blog: RAID-Z http://blogs.sun.com/bonwick/entry/raid_z , November 2005.

[8] XFS File system Structure published by Silicon Graphics, XFS developer, http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf , 2006.

[9] Nilfs2 documentation, http://www.mjmwired.net/kernel/Documentation/filesystems/nilfs2.txt , December 2009.

[10] Jeffrey B. Layton. NILFS: A File System to Make SSDs Scream, http://www.linux-mag.com/cache/7345/1.html , June 2009.

[11] XFS official website http://oss.sgi.com/projects/xfs/ , January 2010.