BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

I/O SUBSYSTEM OPTIMALIZATION USING SSD

OPTIMALIZACE VÝKONU DISKOVÉHO SUBSYSTÉMU PŘI POUŽITÍ SSD DISKŮ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **PETR BĚLOUSOV**
AUTOR PRÁCE

SUPERVISOR **TOMÁŠ KAŠPÁREK, Ing.**
VEDOUCÍ PRÁCE

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Centrum výpočetní techniky Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel:	**Bělousov Petr**
Obor:	Informační technologie
Téma:	**Optimalizace výkonu diskového subsystému při použití SSD disků** **I/O Subsystem Optimalization Using SSD**
Kategorie:	Operační systémy

Pokyny:

1. Nastudujte dostupné možnosti akcelerace diskových polí pomocí SSD disků.
2. Navrhněte uživatelské scénáře, kde diskové pole tvoří úzké hrdlo ve výkonu a k nim pak výkonnostní testy pro změření tohoto úzkého hrdla.
3. Navrhněte možnosti využití SSD disků pro optimalizaci dříve uvedených scénářů a zopakujte výkonnostní testy. Uvažujte mimo jiné systémy bcache a LVM cache.
4. Diskutujte získané výsledky a zlepšení za použití SSD disků a zobecněte získané závěry.

Literatura:

- mkfs.xfs, https://linux.die.net/man/8/mkfs.xfs, 2016-11-07
- BCache and/vs. LVM cache, http://blog-vpodzime.rhcloud.com/?p=45, 2016-11-07
- BCache, kernel documentation, https://www.kernel.org/doc/Documentation/bcache.txt, 2016-11-07
- DM-cache, kernel documentation, https://www.kernel.org/doc/Documentation/device-mapper/cache.txt, 2016-11-07

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí:	**Kašpárek Tomáš, Ing.**, CVT FIT VUT
Datum zadání:	1. listopadu 2016
Datum odevzdání:	17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

[illegible]

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstract

The bachelor thesis examines IO subsystem optimization using SSD cache to speed up HDDs. I examined possible server loads and identified those that are suitable for caching. In the first part I introduce 2 caching solutions, LVM cache and B-cache with their management capabilities and 2 filesystems Ext4 and XFS. In the second part IO performance of LVM cache and B-cache with Ext4 and XFS filesystem is benchmarked and compared to an uncached HDD array.

Abstrakt

Bakalářská práce zkoumá optimalizace výkonu diskového subsystému za využití SSD disků. Prozkoumal jsem možné serverové zátěže a vybral z nich podmnožinu vhodnou k urychlení pomocí cache. V první části představuji 2 kešovací systémy, LVM cache a B-cache, spolu s možnostmi jejich správy a 2 souborové systémy Ext4 a XFS. V praktické části je naměřen výkon souborového subsystému za využití LVM cache a B-cache spolu se souborovými systémy Ext4 a XFS a jejich výkon porovnán vůči poli rotačních disků.

Keywords

B-cache, LVM cache, Logical Volume Management, caching, RAID, SSD, tiered storage, benchmarking, Linux

Klíčová slova

B-cache, LVM cache, Řízení logických disků, kešování, RAID, SSD, tieringové úložiště, výkonnostní testy, Linux

Reference

BĚLOUSOV, Petr. *I/O Subsystem Optimalization Using SSD.* Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Kašpárek Tomáš.

I/O Subsystem Optimalization Using SSD

Declaration

I declare that I have created this thesis myself under the supervision of Ing. Tomáš Kašpárek. I have cited all the bibliographic sources and publications used for the creation of this thesis.

....
Petr Bělousov
May 17, 2017

Acknowledgements

I would like to thank Ing. Tomáš Kašpárek for his leadership and insightful comments. I would also like to thank Mr. Martin Žídek from Master Internet for the testing hardware.

Contents

1 Introduction **3**
1.1 Motivation . . . 3

2 Server configuration models **4**
2.1 Individual configurations . . . 4
2.2 Summary . . . 5

3 Server load models **6**
3.1 Very small files . . . 6
3.2 Small files . . . 7
3.3 Mixed sized files hot storage . . . 7
3.4 Large files . . . 8
3.5 Very large datasets (10s-100s of PB) . . . 8
3.6 Summary . . . 8

4 HDD and SSD differences **9**
4.1 Performance span . . . 9
4.2 Other metrics . . . 12

5 Available caching technologies **14**
5.1 LVM cache . . . 14
5.2 B-cache . . . 16
5.3 Filesystems . . . 19
5.4 Tiered storage . . . 21

6 Testing **22**
6.1 Testing environment . . . 22
6.2 Testing methodology . . . 23

7 Results **25**
7.1 Test results . . . 25
7.2 Performance across server loads . . . 35

8 Conclusion **38**

Bibliography **39**

A Contents of the memory media **41**

B Iozone test results **42**

Chapter 1

Introduction

1.1 Motivation

Drive technologies are always advancing. Despite that it is necessary to keep even older technologies in mind, especially due to cost/power ratio. In this regard, redundant drive arrays composed of large, but slower hard-drives are ideal for SSD acceleration. High throughput, fast access times and lowering costs per gigabyte are making SSDs ideal to accelerate or replace slower HDD storage.

In this paper I shall first identify typical scenarios that could benefit from SSD accelerating cache. Typical scenarios will then be reduced to a few potentially testable.

I will introduce the differences between HDD and SSD technology in various metrics and explain the benefits of both technologies.

Next I will examine the two selected caching solutions available on Linux and compare them. Other available technologies will be briefly mentioned. In the same chapter I will also discuss briefly chosen filesystems and their advantages.

In chapter 6 I will introduce the testing system and the testing methodology as well as which quantities will be observed.

In chapter Results I will be comparing the tested configurations according to their effectiveness in loads introduced in chapter 3.

Finally I will summarise the results and provide a recommendation based on the results.

Chapter 2

Server configuration models

In this chapter I will attempt to find as many potential scenarios of server usage as possible. Each scenario should represent a typical and reproducible server load.

2.1 Individual configurations

- **Video surveillance in tunnels in Brno**
  160 cameras, storage, ad-hoc indexed frames searching, sequential writing, low throughput, high throughput random reading while searching
- **Wikipedia**
  Text storage, popular pages in high demand, high random read, high reliability and availability, small dataset size.
- **Modern audio-video Wikipedia**
  High sequential throughput per request, but due to high amount of requests turned mostly random, both read and write but mostly read, seeking through key-frames random reading.
- **Booking server**
  High availability, reliability, relatively small database, high throughput, random access
- **Storage server (Google drive, OneDrive)**
  High reliability, large capacity, mixed access, sequential and random, read and write.
- **Discussion forum**
  Rich text format, embedded video, pictures, popular posts in high demand, random access, older posts lower demand, can be on slower storage. Fast random read speeds while searching.
- **News site**
  Text, pictures, video, latest news in high demand, slower storage for older.
- **Windows update server**
  Long term storage for older updates, high demand during rollout

- **Development test server**
  High speed mixed load, both read and write, random and sequential. Contents backed up to a different server and loaded up when needed. High sequential read during backup and write during load.

- **Banking system**
  Reliability outweighs everything else, unacceptable data loss, security, encryption

Table 2.1: Scenario comparison

Scenario	Random read	Random write	Sequential read	Sequential write
Video surveillance	high search	low	low	medium
Wikipedia	high	medium	low	low
Booking	high	high	low	low
Storage	medium	high	medium	high
Forum	high search/popular	medium	medium	low
News site	high search	medium	high	high updating
Windows update	high on rollout	high on push	high on rollout	high on push
Test	medium	medium	high on backup	high on load
Banking	medium	medium	medium	medium

2.2 Summary

In configurations mentioned above you can potentially find one applicable to your personal needs and follow its aggregation through the paper. It can serve as a guide on how you can optimize your server configuration. Please note that there are more criteria to be aware of. Those can include

- used server hardware
- used SSD (longevity, performance, size, cost)
- which random or sequential data is requested (from random parts of the disc or still the same block/superblock)
- upgradeability

Also worth mentioning is that the size of dataset you are in charge of matters, optimizations in this paper focus on the use-case where it's impractical or impossible to use SSD-only array. That means that workloads in this paper focus on datasets in the single to double digits TB of data. Smaller datasets are better handled in a different way, such as pure SSD array. If you already have some server hardware with mechanical discs, consider if the SSD usage in this paper is relevant to your needs. The costs of upgrading to an SSD-only array are quite high (older motherboards/CPUs do not support many NVME SSDs). You can also transition mechanical array to a hybrid array and later to purely SSD one as an alternative.

What has to be mentioned as well, SSD caching outlined in this paper is a solution to a general purpose server, where workload is either unknown, changing or for whatever reason (time, compatibility, knowledge) impossible to optimize on application level. Application based optimizations are always more cost effective and bring better results.

Chapter 3

Server load models

Because working with 10 different scenarios would be difficult, I have aggregated them into 4 model scenarios that should cover different server loads. These summarised scenarios will serve as the basis for building tests.

I will demonstrate the aggregation process in table 3.1. Information about the scenarios can be found in chapter 2.1.

Very small files	Booking, Forum
Small files	News site, Wikipedia
Big files	Storage, Surveillance, Windows update
Mixed files and reliability	Test server, Banking

Table 3.1: Aggregation of the scenarios

3.1 Very small files

(Session handler)

Very small data, individual files smaller than or the same as the size of SSD sector(4kB). High volume of both writing and reading requests, mostly random.

Aggregates Booking server and Forum from scenarios, as well as session handler in a web server. This kind of model is usually better handled through purely SSD based storage arrays. It is however quite common to see a combination of several use case scenarios in a single server (typically LAMP) with basic configuration.

Unoptimized session handling writes create hundreds of thousands to millions of files in a single directory. Directory access and read/write operations in such a directory are slowed down significantly, server administrator can usually find out that it is happening by service quality disruption only.

Testing In this scenario I will simulate high writes and rewrites of high volume of small files in a single directory. I will simulate this workload with Filebench. From the README of Filebench:[21]

> Filebench is a file system and storage benchmark that can generate a large variety of workloads. Unlike typical benchmarks it is extremely flexible and allows to specify application's I/O behavior using its extensive Workload Model Language (WML). Users can either describe desired workloads from scratch or use

> (with or without modifications) workload personalities shipped with Filebench (e.g., mail-, web-, file-, and database-server workloads). Filebench is equally good for micro- and macro-benchmarking, quick to setup, and relatively easy to use.

Using Filebench language I created a workflow consisting of creating 500000 small 4kB files, writing into those files and closing them, all in single directory.

3.2 Small files

(Database)
News site, Wikipedia and general database are represented by this model. File sizes are still small, but the volume of them, combined with non-textual kinds of media (video, pictures, audio), the overall data size exceeds reasonable SSD capacities. Currently, these setups generally use some form of MySQL database with external storage, usually a RAID 6, 60, 10 array with hardware redundancy (not only in the array, but whole redundant servers). Snapshots are used for quick recovery.

Testing MySQL 5.7 will be combined with Sysbench benchmark suite to represent this scenario. Sysbench is a modular cross-platform multi-threaded benchmark tool designed to test system performance of database hardware without the need for complex database setup. I am using a testing database of 100GB divided into 128 files generated by Sysbench. An IO intensive workload is set for 5 minutes repeated 2 times in order to allow for the caches to warm up.

3.3 Mixed sized files hot storage

(Rapid development in Microsoft Azure)
This model aggregates Testing and Banking. Man-hours of developers are expensive, therefore it is crucial for them to be able to test their work quickly. Raw performance is needed, even if it comes at a higher cost. Reliable operation is crucial as well. Currently these development environments are usually done in a large NSF RAID array segmented to individual users.

Testing This model will be represented by an IO heavy workload, simulated with the help of Fio benchmark. From the README of Fio:[2]

> Fio spawns a number of threads or processes doing a particular type of I/O action as specified by the user. fio takes a number of global parameters, each inherited by the thread unless otherwise parameters given to them overriding that setting is given. The typical use of fio is to write a job file matching the I/O load one wants to simulate.

The job file will reflect the desired read/write ratio as well as various file sizes. The reliability portion of this scenario will examine how each setup handles unavailable caching drive.

3.4 Large files

(Storage Google Drive, OneDrive)

Even when the speed isn't the highest priority it's always desirable. This model represents scenarios Storage, Video surveillance and Windows update. Large data files with either small metadata (video surveillance key frames) or relatively small highly active data (windows update, new version rollout).

3.5 Very large datasets (10s-100s of PB)

(YouTube)

The cost of SSDs needed to contain all this data would be astronomical, therefore reliable high capacity HDDs are used. Programming specific use of cache, CDN (content delivery network) and content-based storage tiering necessary. SSD caching described in this paper isn't effective enough for this scenario, therefore it will not be tested.

3.6 Summary

Out of the potential models I have selected 3 with the most potential for SSD caching which should cover relevant use cases. These include Very small files (Session handler), Small files (Database) and Mixed sized files hot storage (Rapid development in Microsoft Azure). The testing will include a generic IO speed benchmark using IOzone which will be included in the Appendix to provide a more in depth look at the speed differences in various IO sizes for interested parties.

Chapter 4

HDD and SSD differences

4.1 Performance span

The difference between even the fastest hard-drive and SSD is tremendous. The difference in speeds of hard-drives is mostly due to different rotational speeds (15k drive outperforms a 7.2k drive) and the drive cache size. Delay between finding the data and returning it is dependent on the controller speed, arm movement to the right track and rotational delay.

In solid-state drives the approach is different. With no moving parts the delay is mostly limited by the drive controller and the actual NAND package itself. Greater speeds can be achieved with parallelism, having the controller accessing multiple flash packages at the same time. The size of the operations as well as the number of requests can have a significant impact on drive speed. High speed SSDs can be bottlenecked by the connection to the computer, as explored deeper in section 4.2.

To show the difference with actual numbers, please refer to table 4.1.

	Intel P3500[1]	Intel S3500[2]	Seagate ST900MP0006[3]	Seagate ES.3[4]
Type	NVME SSD	SATA SSD	15000 RPM HDD	7200 RPM HDD
Interface	PCI express	SATA	SATA	SATA
Capacity	400GB	800GB	900GB	4000GB
Transfer speed	2200MB	500MB	300MB	175MB
Average latency	0.020 ms	0.057 ms	2 ms	4.16 ms
Idle power	4W	0.9W	5.7W	6.73W
Load power	12W	5W	7.6W	11.27W
Cost$/GB (as of 9.2.2017)	0.90	0.85	0.61	0.05

Table 4.1: NVME SSD, SATA SSD, 15k HDD and 7.2K HDD comparison (worst value red, best green)

Dirty storage performance

Neither SSDs nor HDDs operate at the exactly same level of performance regardless of remaining capacity. SSDs with no available empty cells slow down their write speeds considerably, because they must erase the contents of the cells before writing to them. In the same way, HDDs have different write speeds on the inner part of the cylinder and outer part. The difference is due to the same rotation speed, while having longer tracks. That is the reason why smaller 2.5„ platters are used in high speed (10k and 15k RPM) HDDs.

SSD technologies

Currently, SSDs are composed of a controller, RAM buffer and the actual flash cells in several packages. Each of these can have a significant impact on the drives performance. However, as controllers and buffers are exclusive to each manufacturer, I will only examine the differences between current flash cells technologies and their impact on drive performance as well as some upcoming technologies that might be relevant in the future.

SLC

In SLC (single-level cell) each cell in the flash package contains a single bit of data. SLC has the highest write speed, lower power consumption and higher endurance than MLC and TLC. However to store the same amount of data manufacturer must make a lot more SLC cells than MLC and TLC, making SLC based SSDs more expensive. SLC used to be found in enterprise-grade and longevity focused drives, but manufacturers have since focused more on incorporating only a small SLC cache to speed up multi-bit cell flash.

[1] http://ark.intel.com/products/82846/Intel-SSD-DC-P3500-Series-400GB-12-Height-PCIe-3_0-20nm-MLC

[2] http://ark.intel.com/products/75685/Intel-SSD-DC-S3500-Series-800GB-2_5in-SATA-6Gbs-20nm-MLC

[3] http://www.seagate.com/www-content/datasheets/pdfs/enterprise-performance-15k-hddDS1897-1-1608GB--en_GB.pdf

[4] http://www.seagate.com/www-content/product-content/constellation-fam/constellation-es/constellation-es-3/en-us/docs/constellation-es-3-data-sheet-ds1769-1-1210us.pdf

MLC

MLC (multi-level cell) allows more than one bit to be stored in a single cell. TLC can be in certain situations considered as a subtype of MLC, when mentioning MLC, I will be using the two-bit per cell definition. It stands in the middle between SLC and TLC both performance- and cost-wise. MLC is slowly in decline, while cheaper than SLC, MLC doesn't offer enough storage density to be enticing to most manufacturers.

TLC and further

TLC (triple-level cell) are the most cost-effective type of flash cells today. They enable the storage of 3 bits into a single cell through the use of 8 different voltage states. Some manufacturers refer to TLC as 3-bit MLC. The primary benefit of TLC is its lower cost per GB due to higher data density. The controller must be able to deal with the inevitable errors, whose occurrence is higher due to more voltage levels and lower difference between each level.

Demand for higher storage density led some manufacturers to invest into QLC (quadruple-level cell) technology which offers 33% increase in storage size per package, but requires advanced error correction, controllers and firmware to account for lower difference between each cell state.

Modern SSD configuration

To lower the manufacturing cost and maximize performance per GB of storage, many manufacturers combine the advantages of TLC cells and SLC cells. By utilizing a small SLC cache as well as large amount of TLC packages sustained high load goes through the SLC cache, minimizing performance drop-off. Other notable technique of maximizing the price/performance ratio is 3D-NAND[9](by Intel and Micron), or V-NAND[17](by Samsung). Both technologies stack multiple layers of flash cells vertically, connecting the layers. This approach allows for higher cell density than regular planar cells, higher endurance, lower interference and lower price per GB cost.

3D Xpoint

On 19. March 2017 Intel unveiled its DC P4800X Series SSD with certain advancements from NAND technology. It is based on Intel 3D XPoint^{TM}(pronounced cross point), combining memory and storage into a single high bandwidth storage unit. It utilizes 4 lane PCIe 3.0 connection with NVMe interface and features lower latency than NAND flash, high sustained random IOPS(500 000) and much higher endurance(12.3 PBW compared to NAND based DC P3500 with 1.095 PBW).[10]

Unfortunately, Optane drives are at the moment going to be available only in small 375GB size for the data center market and 16-32GB for consumers for a higher price per gigabyte than competing NVME SSDs. Unlike NAND it doesn't have to erase existing data before writing new data boosting both endurance and performance. The storage is also bit addressable eliminating the need for block/page with wasted space and making wear-leveling and garbage collection easier. More information about the DC P4800X drive as well as benchmark results can be found in the review [5] by Paul Alcorn.

[5] http://www.tomshardware.com/reviews/intel-optane-3d-xpoint-p4800x,5030.html

Future

Future storage technologies that are currently in development, but may be relevant in the future fall under the NVM (non-volatile memory). These include PRAM, nvSRAM, RRAM, MRAM and others. When considering one of these consider also compatibility (if there is any) both in hardware (connectors, motherboard) and software(communication protocol, BIOS, applications etc.) with existing or new hardware. New technologies tend to be more expensive.

Out of all these I will highlight the MRAM based Everspin nvNITRO.

Everspin nvNITRO On 8. March 2017 Everspin Technologies, Inc. announced MRAM based storage accelerator nvNITRO. [6] MRAM(Magnetoresistive random-access memory) allows for very high IOPS (1.5 million) and unlike NAND based storage MRAM doesn't suffer from cell degradation eliminating the need for wear leveling as well as read/write speed degradation. It should theoretically offer unlimited number of writes. The biggest disadvantage of MRAM is low storage density, due to which nvNITRO will be available in the 512MB to 16GB sizes.

4.2 Other metrics

Power consumption

Often neglected metric of a drive is its power consumption in idle and under load. SSDs have lower power demands and using them could potentially mean a lower power bill. Please note that high speed PCIe SSDs have much higher power consumption than SATA SSDs. To get a rough idea of the difference, please refer to the table 4.1.

Another point worth considering is spinup surge power requirement, the amount of power needed when mechanical drives are starting up and the associated delayed spinup at startup in larger arrays, where HDDs are started in waves to avoid overloading the power delivery system.

SSDs don't have any mechanical parts to spinup at start therefore do not posses the same challenges. SATA based SSDs have lower power requirements than high performance PCIe based storage. It is also possible to utilize different power states to lower load power requirements of SSDs. You can find more information about the difference in power consumption on Intel SSDs in this article.[6]

Incorporating power consumption measurements unfortunately reaches beyond the scope of this paper.

Reliability

Reliability of HDDs and SSDs varies, but both should perform well while covered by warranty. Most enterprise-level discs should come with a 5-year or longer warranty. SSDs have a limited number of erase cycles on each cell storing the data and includes over-provisioning in case of cell failure.

The DWPD (drive writes per day) metric is often used as an endurance metric of SSDs. It represents the amount of data that can be written to the drive every day until the life expectancy of the cells is depleted. For example Intel S3500 has an endurance rating of

[6] https://itpeernetwork.intel.com/managing-power-consumption-of-intel-data-center-ssds/

450 TBW which amounts to about 0.3 DWPD. Intel SSDs specifically include a “fail safe„ that renders the SSD unusable once its erase cycles have been depleted and the drive could become unstable. This metric can be monitored in SMART.

HDDs on the other hand don’t have a similar indicator and the end of HDD reliability has to be observed through relocated sector count, uncorrectable errors and other criteria such as temperature and noise.

Expected lifespan

Both HDDs and SSDs have only limited lifespan. Even though manufacturers try to make drives last as long as possible, mechanical and electrical limitations of hardware makes drives fail. You can find the expected percentage of healthy HDDs after four years in use in an article here[7]. Most common reasons of failure include bearing and motor failure, head crash, circuit failure, sector magnetisation failure and miscellaneous mechanical failures.

SSDs don’t suffer from mechanical failures (due to the fact that there are no moving parts at all), they are however susceptible to electrical failure, NAND wear-out, bad soldering, bad sectors, controller failure, firmware update error.[5] You can get a better idea on the expected lifetime of HDDs at Backblaze[3], where they announce the reliability statistics of HDDs from their own servers. SSD endurance is different, but you can learn more about it in an endurance test here [8].

Connections

Both HDD and SSDs must be connected to a system using an industry standard connector. Modern drives use one of serial connections available, such as SATA (Serial ATA) or SAS (Serial Attached SCSI) connectors. In their 3rd revision these allow maximal speeds of 6 and 12 Gb/s respectively, which is far more than most HDDs can deliver. However, for modern SSDs these connectors can create a bottleneck, so faster SSDs use PCI-E connection in the form of either typical PCI-E 16x/4x slot (server oriented SSDs) or an M.2 slot (consumer oriented SSDs). Some SSDs have a U.2 connector, more suitable for 2.5“ form factor which combines SAS and PCIe technologies into a new connector. These faster SSDs can achieve potential speeds of up to 40 Gb/s through 4 lanes of PCI-E 3.0.

[7]https://www.backblaze.com/how-long-do-disk-drives-last.html

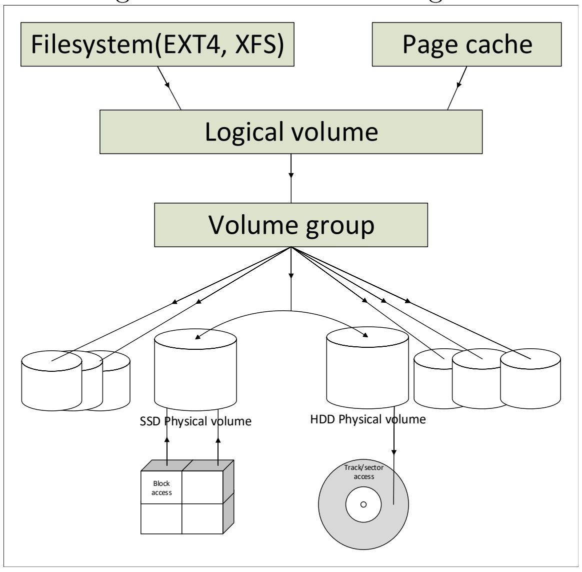
Chapter 5

Available caching technologies

5.1 LVM cache

LVM cache is an extension of the highly popular Linux LVM (*Logical Volume Manager*) built on the dm-cache kernel component. Where traditional LVM offers better hard-disc partition management by grouping several physical drives into a single addressable logical drive, LVM cache adds the advantages of mixing different drive technologies.

Figure 5.1: LVM cache diagram



It has been integrated into the Linux kernel since version 3.9. Full support out of the box requires OS creator support as well. For Centos/RHEL version 7.1 and later and 6.7 and later are required. The basic principle of operation consists of combining a small and fast cache logical volume to improve the performance of a large and slow origin logical volume. Cache is further split into two parts, cache data LV and cache metadata LV which can be stored on either the same or two different drives. However, caching drive(s) as well as origin drive(s) need to be in the same volume group.

LVM cache offers 3 caching policies, mq, smq and cleaner.[11] Since version 2.02.128 the mq (multiqueue) policy is being deprecated, making the smq (stochastic multiqueue) the default used policy. Mq offered tunable parameters to finetune cache behavior, specifically sequential threshold, random threshold, read, write and discard promote adjustment. Since version 2.02.128 these tunables are accepted, but have no effect because mq has been turned to an alias for smq.

Smq requires less memory, features better level balancing by switching the least recently used entry from the higher level. This stochastic behaviour allows for better detection of hotspots and quicker cache promotion when IO patterns change.

The last cleaner policy can be used to flush all dirty blocks from cache to origin device.

Conversion To enable LVM cache the origin LV must be in the same VG as the caching drive. LVM cache contains all the necessary utilities for proper management. These include *lvcreate, lvs, pvs, lvconvert, lvremove, lvdisplay, lvrename.*

There are 2 major versions of LVM, the original LVM1 introduced in kernel 2.4 and the newer LVM2 that has been in kernel since version 2.6 but provides backwards patches for 2.4 as well. For LVM cache we need LVM2 with its userspace tools with at least version 2.01.15.

Monitoring and configuration LVM cache supports basic monitoring capabilities through *lvs -o* command with appropriate options. Available since version 2.0.2 these are not mentioned in official man page, but include the options explained in table 5.1.

Alternatively, *lvdisplay -m* has been updated in version 2.0.2.169 to display these information as well. A sample output of *lvdisplay -m*:

```
   --- Logical volume ---
  LV Path                /dev/vg/lvol0
  LV Name                lvol0
  VG Name                vg
  LV UUID                Y4uWuN-TBGk-duer-aPWl-yBWn-iFFR-RU1gg1
  LV Write Access        read/write
  LV Creation host, time linux, 2017-03-01 20:52:39 +0100
  LV Cache pool name     lvol2
  LV Cache origin name   lvol0_corig
  LV Status              available
  # open                 0
  LV Size                12,00 MiB
  Cache used blocks      10,42%
  Cache metadata blocks  0,49%
  Cache dirty blocks     0,00%
  Cache read hits/misses 112 / 34
  Cache wrt hits/misses  133 / 0
  Cache demotions        0
  Cache promotions       20
  Current LE             3
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

Table 5.1: lvs -o options related to cache management

Option	Meaning
cache_total_blocks	the total number of blocks of cache device
cache_used_blocks	the number of blocks of cache device used for caching
cache_dirty_blocks	the total number of dirty blocks
cache_read_hits	the number of times data has been successfully read from cache
cache_read_misses	the number of times data hasn't been read from cache
cache_write_hits	the number of times data has been written to cache
cache_write_misses	the number of times data hasn't been written to cache
segtype	display the LV segment type(for cached LV *cache* or *cache-pool*)

LVM supports both *writethrough* caching as well as *writeback* caching. Writethrough is selected by default due to higher security. Writeback caching may result in data loss if the caching device is lost. Caching mode can be changed when creating the cache LV or later with *lvconvert –cachemode* option.

Removal It is possible to remove the caching device without disrupting the origin volume. This is done with either *lvremove <volume group/name of caching LV pool>* or the safer *lvconvert –uncache <volume group/name of caching LV>*. Both these commands flush any remaining dirty data to the origin LVs before removing the cache.

You can also remove both the caching and origin drive at the same time with *lvremove <volume group/name of cache LV>*. Please note that the drive must be unmounted for this operation.

Error handling LVM cache set to writethrough caching has no trouble when caching device is unavailable, cache is simply not utilized and the performance drops to HDD performance level. Unavailable cache in writeback mode with dirty blocks may result in errors or corruption.

More information about LVM cache can be found in the LVM cache manual page [13].

5.2 B-cache

B-cache (*block cache*) is a kernel block level cache. It also allows the use of a fast SSD-based storage as a cache for one or more slower hard disk drives. B-cache is also filesystem agnostic.

B-cache is available in some distributions out of the box, such as Ubuntu 14.10 or newer or Fedora 20 or newer. Centos and RHEL as well as other distributions that include the 3.10 or 3.11 Linux kernel have support for B-cache, but don't include it natively. In these systems it is necessary to compile either the B-cache module and insert it or the whole kernel with B-cache enabled.

B-cache terminology is slightly different from LVM cache, origin device is called backing device in B-cache.

Conversion B-cache requires specific superblock on the backing device in order to monitor and manage the devices correctly. It is possible to convert LVM logical volumes, raw devices, partitions with some free space or shrinkable filesystem. The conversion can be done with the help of **blocks** [1], a block device conversion tool. This can be done inplace, although a backup before attempting to convert is advisable. Where direct conversion to B-cache is not available, it is possible to convert to LVM first and then convert from LVM to B-cache.

Monitoring and configuration B-cache monitoring is less user friendly than LVM cache, though it provides basic userspace tools for monitoring, most configuration is done through direct file manipulation. You can find both configuration and information files in the */sys/block/bcache<n>/bcache*(where n is the number of currently attached B-cache) folder and subfolders. The management capabilities that can be found in those folders are outlined in the table 5.2. Table 5.3 aggregates configurable parameters available in B-cache.

Table 5.2: B-cache management tools, folder names are relative to /sys/block/bcache<n>/bcache

make-bcache	Command to format the block device for use with b-cache.
bcache-super-show	Command that prints information about caching or backing device.
/stats_total/	B-cache statistics folder, contains total statistics.
bypassed	Statistic for all IO that bypassed the cache.
cache_hits	Number of cache hits per IO.
cache_misses	Number of cache misses per IO.
cache_hit_ratio	Hit/miss ratio in percentage values(0-100).
cache_bypass_hits	Number of hits per IO that is supposed to bypass cache.
cache_bypass_misses	Number of misses per IO that is supposed to bypass cache.
cache_readaheads	Number of times cache readahead occurred.
/cache/cache0/	Symbolic link, cache options
dirty_data	Amount of dirty data in the cache.
trigger_gc	Writing to this file force runs the garbage collection on cache device.
block_size	Block size of the cache device.
priority_stats	Data access statistics. Determine working set size.
written	Total data written to the cache.

Another disadvantage of B-cache is that newly created B-cache volumes are assigned in /dev/ folder with the name bcache<n> where n is an increasing whole number. This series of numbers is not reset until server restart, making scripting automation rather difficult. Each B-cache device also has a unique UUID and you can also find the related configuration and information files in */sys/fs/bcache/<cache set UUID>* for the caching device and */sys/fs/bcache/<cache set UUID>/bdev<n>* for the backing device.

Removal It is possible to remove caching device while mounted, B-cache automatically switches to passthrough mode when caching device is unavailable. Device can be detached by writing 1 to */sys/block/<dev path of the caching device>/bcache/detach* and unregistering it by writing 1 to */sys/fs/bcache/<cache set UUID>/stop.*

If the cache is missing during startup, backing device isn't started up. Startup without the cache can be forced by writing 1 into the *running* file of the backing device.

[1] https://github.com/g2p/blocks

Table 5.3: B-cache configurable parameters

cache_mode	Cache mode, possible values writethrough, writeback, writearound, none
readahead	Default value 0, if a cache miss occurs B-cache rounds up read up to this value
sequential_cutoff	Threshold of sequential IO that once passed will bypass the cache
writeback_delay	Number of seconds B-cache waits before writing back new dirty data in cache
writeback_percent	B-cache tries to keep this percentage of cache dirty by throttling writeback to backing device
flash_vol_create	Echoing a size to this file (in human readable units, k/M/G) creates a thinly provisioned volume backed by the cache set
io_error_halflife	Error decay in number of IOs
io_error_limit	Number of errors accepted before disabling the cache. If the decaying error count reaches this limit dirty data is written out and cache is disabled
journal_delay_ms	Number of milliseconds that journal writes are delayed, default 100
cache_replacement_policy	Change cache replacement algorithm, possible values lru (least recently used, default), fifo(first in first out) and random

Error handling B-cache provides configurable cache error handling. By default, there are several error handling scenarios:

- Reading from the cache in any mode and an error occurs, the read is repeated from the backing device.
- Caching mode is set to writethrough and a write to the cache errors out the data in cache is invalidated and the cache is bypassed.
- Caching mode is set to writeback and an error occurs then the error is passed on to the filesystem.

All write errors above the configurable threshold (default 0) result in the caching mode being set to passthrough and cache device shutdown.

B-cache and ZFS Although B-cache is filesystem agnostic, there has been a bug [2] making ZFS on top of B-cache unreliable.

More information about B-cache can be found in the B-cache user manual [4].

Other notable caching solutions

The selected caching solutions that I am using for testing are reliable, tested many times before with proper kernel support as well. Some other solutions which are currently under development might be more relevant for you in the future.

Bcachefs [3] is a next generation copy on write filesystem developed by the team working on B-cache, focusing on reliability, tiering/caching, compression and other features. It

[2]https://bugzilla.kernel.org/show_bug.cgi?id=71441
[3]https://bcache.evilpiepirate.org/Bcachefs/

implements some B-cache caching capabilities into it's own filesystem. Unfortunately It's not yet upstream, so a kernel has to be built to enable it.

EnhanceIO [4] is based on older no longer mantained Flashcache developed by Facebook. EnhanceIO has an advantage that unlike B-cache it doesn't require file system conversion. Unfortunately, the official EnhanceIO appears to be broken on newer kernels with no development, although there are some promising forks such as [5].

For Windows based solution [6] Primocache might be worth considering as well, Storage Space Tiering[15] in Windows Server is the built-in option, however the best is probably Sandisk DAS cache.[18] It is multiplatform, supporting both Windows Server as well as various Linux distributions(from Red Hat, SUSE and VMware). Pricing is available upon request. To learn more about tiered storage refer to the section 5.4.

5.3 Filesystems

To store any data at all it is necessary to format the volume with a proper filesystem. A filesystem provides a control mechanism in how and where a file is located in the storage medium. It takes its naming scheme from paper-based storage and organisation. Each group of relevant data is called a file. The structure and rules to organise and manage these files is the filesystem itself.

Linux is able to utilize different filesystems very well mostly due to kernel-level storage abstraction. The so-called VFS (Virtual Filesystem Switch) provides a unified model that can represent any particular filesystem operations easily.

Each filesystem was designed with different requirements and limitations. Some are used on special types of media (eg. optical discs), some are universal. Some filesystems provide a solution for locally attached storage while others are geared towards network or virtually attached storage.

EXT4

EXT4(*fourth extended filesystem*) was developed as a continuation of the EXT3 filesystem with expanded support for large filesystems, scalable beyond today's requirements. Built from the ground up with nanosecond timestamps, preallocation as well as fast extent support. Extents allow for metadata overhead reduction, compressing many block pointers of a large file into an extent, enabling faster access and lower overhead. Another useful feature is the fast fsck enabling the fsck to skip checking unused inodes. More information can be found here [7].

EXT4 is forward conversion compatible with EXT3 filesystem (allows for easy migration from EXT3 to EXT4) without the need of reformatting. For information about the limits of EXT4 please refer to the table 5.4. Available in Linux kernel version 2.6.28 and later.

Resizing tools allow for growing while mounted, shrinking requires the file system to be unmounted first. EXT4 is the default filesystem for Ubuntu, mostly due to its stability and acceptable sustained performance.

[4] https://wiki.archlinux.org/index.php/EnhanceIO
[5] https://github.com/lanconnected/EnhanceIO
[6] https://www.romexsoftware.com/en-us/primo-cache/

backup and restoration Backup of the EXT4 partition using *dump* tool should be done on an unmounted partition as backing up mounted filesystem can have an unpredictable result. Restoration of the data can be done with *restore* tool.

XFS

High performance 64-bit journaling filesystem created by Silicon Graphics, Inc in 1993 [19] Originally created for the IRIX operating system and it was the first filesystem that implemented delayed allocation. It was merged into the mainline Linux kernel in version 2.6 Included tools allow for online resizing to grow inplace but doesn't allow shrinking. XFS is the default filesystem for CentOS and many distributions ship with XFS included such as:

- Mandrake Linux 8.1 and newer
- SuSE Linux 8.0 and newer
- Gentoo Linux 1.0 and newer
- Slackware Linux]8.1 and newer
- Knoppix 3.1 and newer
- Turbolinux 7.0 and newer
- JB Linux 2.0 and newer
- Debian 3.1(„Sarge“) and newer
- The Fedora Project Fedora Core 2 (default filesystem since Fedora 22) and newer

backup and restoration XFS provides native backup and restoration utilities *xfsdump* and *xfsrestore* respectively. Xfsdump also allows consistent online backup without the need for unmouting the filesystem.

Filesystem limitation comparison

Table 5.4: Ext 4 and XFS limitations

	Ext 4	XFS
Max volume size(recommended)	1 EiB(16TiB)	8EiB
Max filesize	16TiB	8EiB
Max subdirectories	64000 (flag dir_nlink for unlimited)	unlimited
Max number of files	$4 * 10^9$(approx. 2^{22})	2^{64}
Supported OS	Linux, FreeBSD, Mac OS X, Windows	Linux, FreeBSD, IRIX

Other file systems

From other file systems that were considered I would like to point out Btrfs which requires proper setup and mounting and for some people [7] doesn't ahve the performance expected without additional tweaking.[8] According to the mailing list of BTRFS it is not entirely stable with LVM cache which is why it has not been included in the testing [12].

ZFS is not supported in mainline kernel and due to its increased complexity as well as not enough information on reliability and recovery in the event of failure on Centos/RHEL it is not recommended at the moment.

5.4 Tiered storage

Tiered storage is a more general term than SSD caching. It utilizes several different storage media with different speed in a hierarchical topology with faster storage on a higher level than slower storage. A typical tiered storage would contain RAM, small fast NVME SSD(s), larger SATA SSD(s), slower mechanical HDD(s) and final archival type of storage such as tapes. Although harder to deploy with high initial costs, a well implemented tiered storage can outperform cached arrays significantly due to application and/or platform specific optimizations.

You can learn more about tiered storage[20] in the sources [16][14].

[7] https://blog.pgaddict.com/posts/friends-dont-let-friends-use-btrfs-for-oltp

[8] https://www.reddit.com/r/linux/comments/2uz8ez/rlinux_tell_me_your_btrfs_nightmares_success/

Chapter 6

Testing

From the available caching technologies and filesystems the chosen technologies are combined with the scenarios outlined in section 3. In table 6.1 you can see the tested combinations with their respective labelling used in the graphs below.

Table 6.1: Tested combinations

HW	Uncached HDD raid	LVM cache	B-cache
Filesystem	Ext4		
	XFS		
Test scenarios	Very small files Filebench (graph label 500k)		
	Secure MySQL database sysbench (graph label sync database)		
	Regular database Fio (graph label database read/write)		
	Mixed sized files Fio (graph label read/write)		
	Mixed sequential Fio (graph label sequential read/write)		
	General performance IOzone (see Appendix B)		

6.1 Testing environment

All tests were conducted on a DELL PowerEdge R730xd with 2 Intel Xeon®E5-2620 v3 6 core 12 thread processors at 2.4 GHz, 8 modules of 16GB DDR4 memory running at 1866MHz for a total of 128 GB of RAM. 8 rotational HDDs 4TB Seagate ES.3 SATA 6Gb/s 7200 RPM were configured in RAID 10 for a total of 14902 GB of primary storage. Caching SSD partition was created from 2 Intel DC 3500 800 GB SATA 6Gb/s SSDs in RAID 1. All drives were connected to the system through the integrated PERC H730 Mini RAID controller with battery backup and 1GB DDR3 onboard cache.

Unfortunately no NVMe SSD was available at the time of testing.

CentOS Linux release 7.3.1611 (Core) with kernel version 3.10.0-514.10.2.el7.x86_64 recompiled with B-cache support was used as the testing operating system. Utilized software:

- sysbench version 0.4.12 [1]
- fio version 2.17 [2]

[1] https://github.com/akopytov/sysbench

[2] https://github.com/axboe/fio

- Filebench version 1.5-alpha1 [3]

SATA based drives were selected due to excellent backward compatibility, while PCIe connected SSDs have a much higher throughput, older servers will not have enough PCIe connections to utilize the speed or even to connect the SSDs to the system at all. SSDs with SATA interface also have the lowest cost and represent the lowest improvements compared to faster PCIe SSDs.

Measured quantity

Measurements are divided into several categories. Random read, random write, sequential read, sequential write and file creation. The cache is only successfully utilized to speed up access when the data that is accessed has been accessed before. Different cache modes also differ wildly. Writethrough caching doesn't offer speed enhancement while writing, because the write isn't confirmed to the OS until the operation is safely done on the underlying HDD, making the writing IO bound by the speed of HDD. Writeback caching should allow for higher writing performance for the sake of lower data safety. LVM cache and B-cache report the write complete to the OS immediately after writing data to the cache, flushing the data to HDD when possible/effective. If the cache is disconnected (due to power outage or anything else except safe removal) before completely writing dirty data back the data is lost.

6.2 Testing methodology

Tests were done as follows. Baseline configuration without any caching technology consisted of formatting the array using *mkfs* tool variants *mkfs.ext4* and *mkfs.xfs*. I have prepared a bash script to automate test run.

LVM cache LVM cache preparation procedure was to create volume group test-data with the SSD mirror located at /dev/sdc and HDD array at /dev/sdb.

vgcreate test-data /dev/sdb /dev/sdc

Next I created origin logical volume orig-data spanning the entire physical volume /dev/sdb.

lvcreate -n orig-data -l 100%PVS test-data /dev/sdb

To create cache pool I used a one-step method that automatically creates both cache data LV as well as cache metadata LV.

lvcreate –type cache-pool -l 100%PVS -n cache_pool test-data /dev/sdc

Alternatively, this step can be reproduced with these 3 commands:

lvcreate -n cache_pool -l 99%PVS test-data /dev/sdc

lvcreate -n cache-metadata -l 1%PVS test-data /dev/sdc

lvconvert –type cache-pool –poolmetadata test-data/cache-metadata test-data/cache_pool

Finally, origin LV and cache pool LV are combined into cache LV.

[3] https://github.com/filebench/filebench

lvconvert –type cache –cachepool test-data/cache_pool test-data/orig-data

LVM cache volume defaults to writethrough caching mode, for writeback I detached the cache and reattached in writeback mode, ensuring that the cache has been flushed.

lvconvert –type cache –cachepool test-data/cache_pool –cachemode writeback test-data/orig-data

It is possible to change the caching mode without detaching and reattaching the cache by using the following command:

lvconvert –type cache –cachemode writeback test-data/orig-data

I chose not to utilize this to ensure a clean testing environment.

B-cache B-cache preparation procedure consisted of writing the B-cache cache superblock on caching device using *make-bcache*

make-bcache -C /dev/sdc

and writing the backing superblock on the backing device.

make-bcache -B /dev/sdb

Caching device must then be attached to the appropriate backing device by echoing its UUID.

echo 2d2b7129-1ab0-4994-bfb6-33e54d518c96 > /sys/block/bcache0/bcache/attach

It is also possible to format the devices and attach them at the same time.

make-bcache -B /dev/sdb -C /dev/sdc

Finally we can verify the configuration by running *bcache-super-show /dev/sdb* command to obtain the following output.

```
sb.magic                ok
sb.first_sector         8 [match]
sb.csum                 E8439B5FE3C78ED2 [match]
sb.version              1 [backing device]

dev.label               (empty)
dev.uuid                538801c7-1e98-464b-b2eb-e719eb7ac0f7
dev.sectors_per_block   1
dev.sectors_per_bucket  1024
dev.data.first_sector   16
dev.data.cache_mode     0 [writethrough]
dev.data.cache_state    1 [clean]

cset.uuid               086edafe-55bc-4e5c-864b-56f375393c51
```

Chapter 7

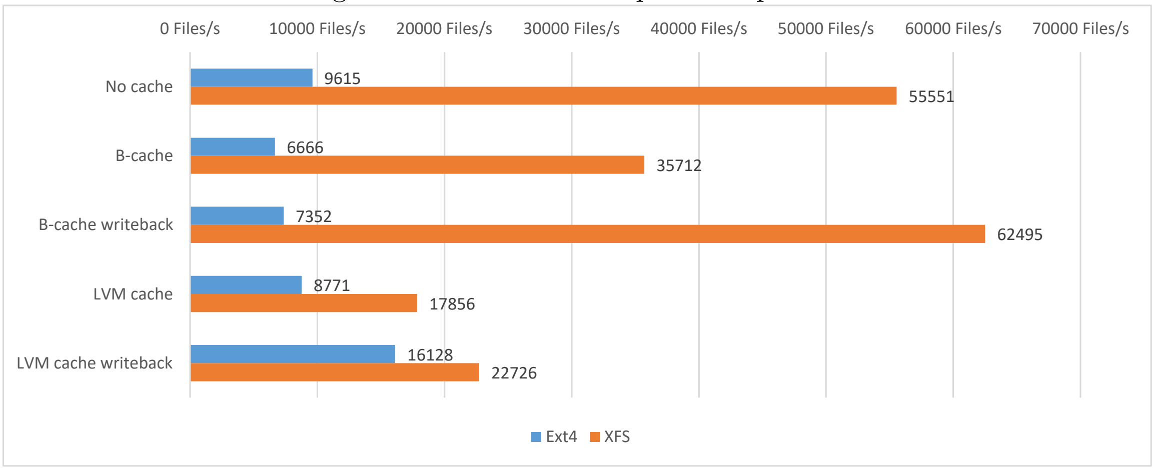
Results

7.1 Test results

Please note that certain graphs utilize a logarithmic scale in order to account for large differences between individual result values. For comparison sake please refer to the summarizing graph 7.17 on page 36.

File creation

Figure 7.1: File creation speed comparison



The very small files (Session handler) model server load.

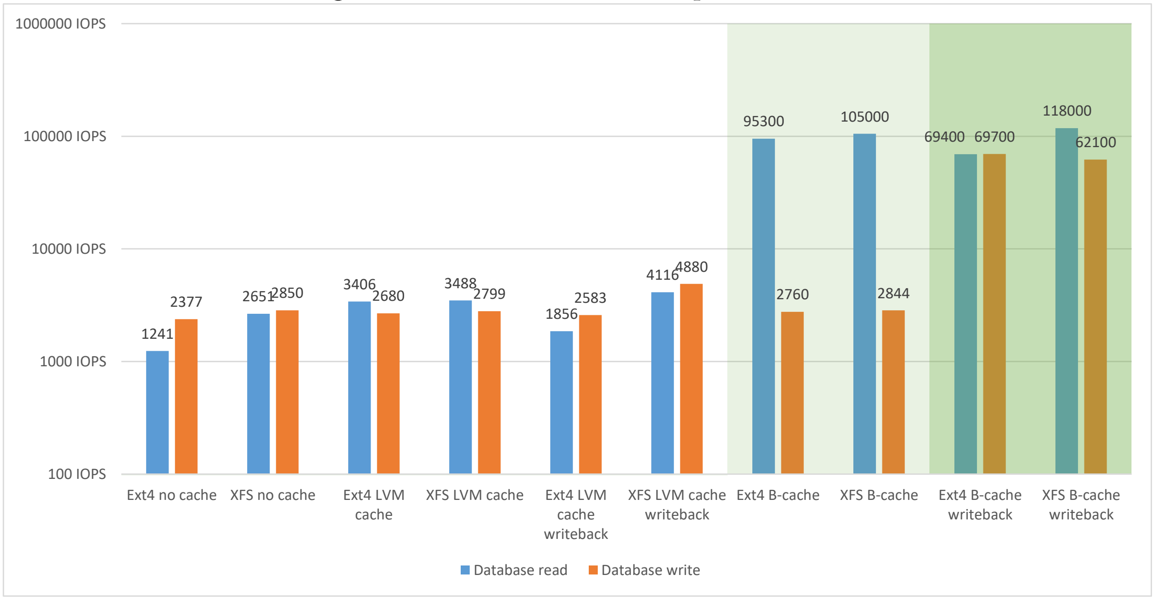
In this part of testing it is possible to see that the limitations imposed on file creation in large volume of files in a single directory is heavily filesystem dependent. With half a million files created the file creation speed in Ext4 filesystem decreases (cache management slows down the operations) (see figure 7.1). Journaling writes of the Ext4 filesystem slow down the performance of the whole filesystem. For Ext4 the only viable caching method is LVM cache in writeback mode, offering 67% increase in files created per second.

XFS offers a similar trend, writethrough cache management slows down most of the operations by 36-60% compared to uncached array. XFS favours B-cache writeback, with performance increase over uncached array of 13%.

Compared together, switching the filesystem from Ext4 to XFS allows for 4-7x increase in performance regardless of other criteria, for the best possible performance choose B-cache in writeback mode formatted with XFS.

Database performance

Figure 7.2: Database scenario performance



The small files (MySQL database) model server load.

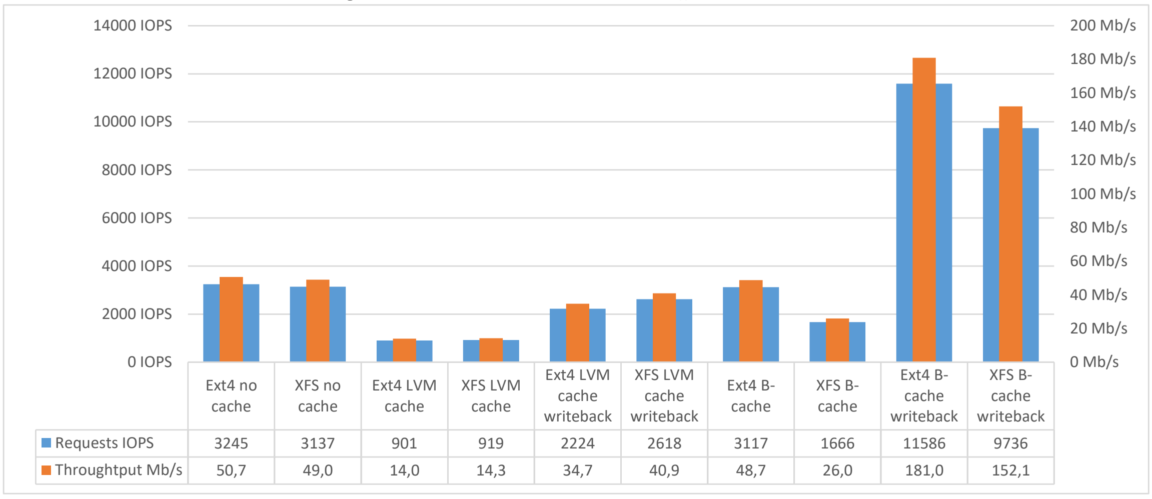
The test representing the small files model server load with the help of a simulated database. I ran two different tests, in fio and sysbench. Fio gives me more granular control and allows for easy warmup of the caches, while sysbench shows a more secure MySQL database performance.

First let's look at Fio results. I measured the read and write performance of random IO of simulated database. In order to properly utilize cache I first ran a warming up test that isn't included. Pre-warmup performance was within the margin of error from uncached array.

Graph 7.2 shows that all caching technologies and filesystem combinations tested performed better than baseline Ext4 uncached array. B-cache, especially in the writeback setting with XFS performed the best, with 44x the baseline performance in read and almost 21x in write. Ext4 with B-cache writeback also performed admirably, increasing the array performance 55x in read and 29x in write. On the other end of spectrum LVM cache in writethrough mode in combination with XFS added enough complexity to the setup to performer worse than the XFS uncached array by 1.79%, a trait that repeated with B-cache writethrough XFS setup as well, although only by 0.21%.

The second database testing was more secure. The test utilized a database stored in 128 files each 800 Mb. Periodic *fsync()* every 100 requests ensures flushing of the changes to the drives. This setting drastically decreased the performance of nearly all caching solutions except for B-cache in the writeback mode as can be seen in figure 7.3.

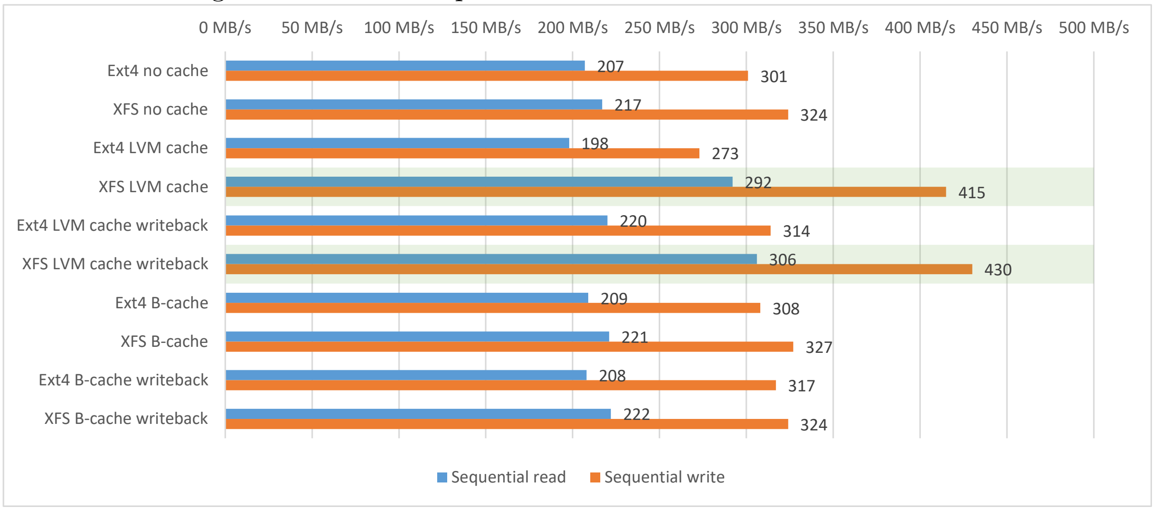
Figure 7.3: Secure database performance



	Ext4 no cache	XFS no cache	Ext4 LVM cache	XFS LVM cache	Ext4 LVM cache writeback	XFS LVM cache writeback	Ext4 B-cache	XFS B-cache	Ext4 B-cache writeback	XFS B-cache writeback
🟦 Requests IOPS	3245	3137	901	919	2224	2618	3117	1666	11586	9736
🟧 Throughput Mb/s	50,7	49,0	14,0	14,3	34,7	40,9	48,7	26,0	181,0	152,1

Mixed access

Figure 7.4: Mixed sequential and random files of various sizes



Mixed sized files hot storage(Rapid development in Microsoft Azure)

The only test involving sequential access. Spinning HDD arrays are better suited for large sequential file access than small SSDs.

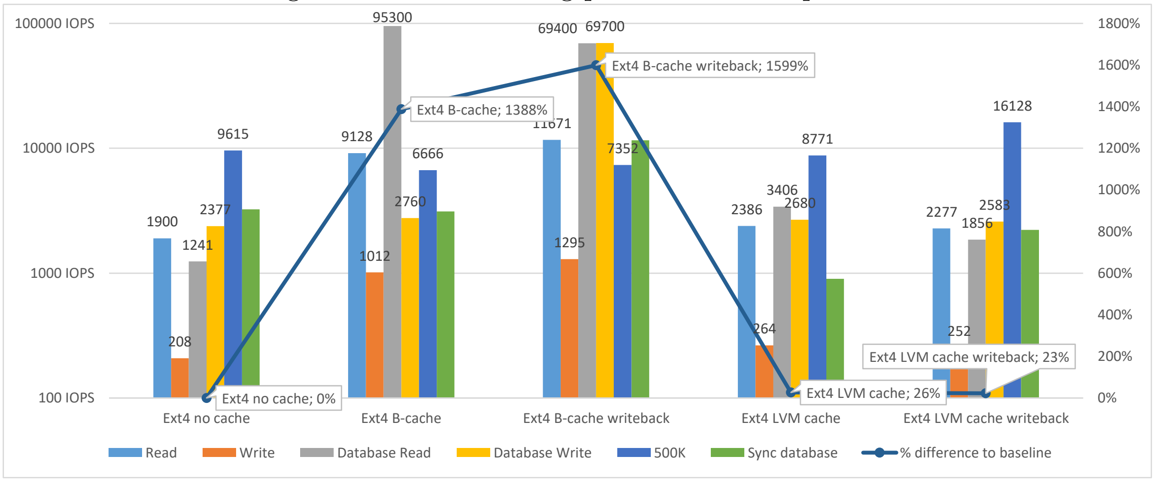
This test nicely demonstrates one of the main differences between B-cache and LVM cache. While LVM cache with XFS manages to increase especially write speed significantly, by 28% and 32% in writethrough and writeback mode respectively, B-cache hovers only between 0-5% above uncached array. This is expected, because B-cache is configured to let sequential IO bypass its cache entirely. This enables B-cache to hold more randomly accessed data where the speed difference between the array and cache speed is much higher than with sequential data.

File systems

Ext4 performance comparison across caching solutions In graph 7.5 you can see the performance summary across utilized caching solutions as well as average percentage increase of performance relative to baseline uncached RAID array. Ext4 benefits greatly from caching. Except for the file creation 500k test, B-cache in the writeback mode offers the highest average speed increase of almost 1600%. Writethrough B-cache offers even higher database read performance, but overall it performs equally (mixed read and write) or worse than writeback.

LVM cache on the other hand performed significantly worse than B-cache, offering only a modest 26% in writethrough and 23% increase in writeback over baseline.

Figure 7.5: Ext4 caching performance comparison



XFS performance comparison across caching solutions

XFS offers a more balanced performance compared to Ext4. As you can see in graph 7.6, for raw speed B-cache writeback mode wins every test with a comfortable 13x average increase over the baseline. Writethrough caching mode of B-cache is also quite effective, depending on the desired load. Synchronous secure database as well as write intensive database is not viable for writethrough B-cache. Overall, writethrough B-cache offers comparable performance in both Ext4 as well as XFS.

LVM cache utilizes XFS more effectively than Ext4 across all tests. Writethrough mode offers 47% and writeback mode 22% average increase over baseline. XFS is much more effective than Ext4 in rapid file creation. B-cache writethrough mode dominates this category as well.

Cache statistics

In graph 7.7 you can compare the effectiveness of cache promotion algorithms of LVM cache and B-cache. LVM cache writeback mode has much worse hit ratio than any other caching method in the tests. B-cache has very balanced hit ratio, all configurations scored close to 55%. Please note that this includes cache warming as well as the fact that certain tests do not utilize cache very much.

Figure 7.6: XFS caching performance comparison

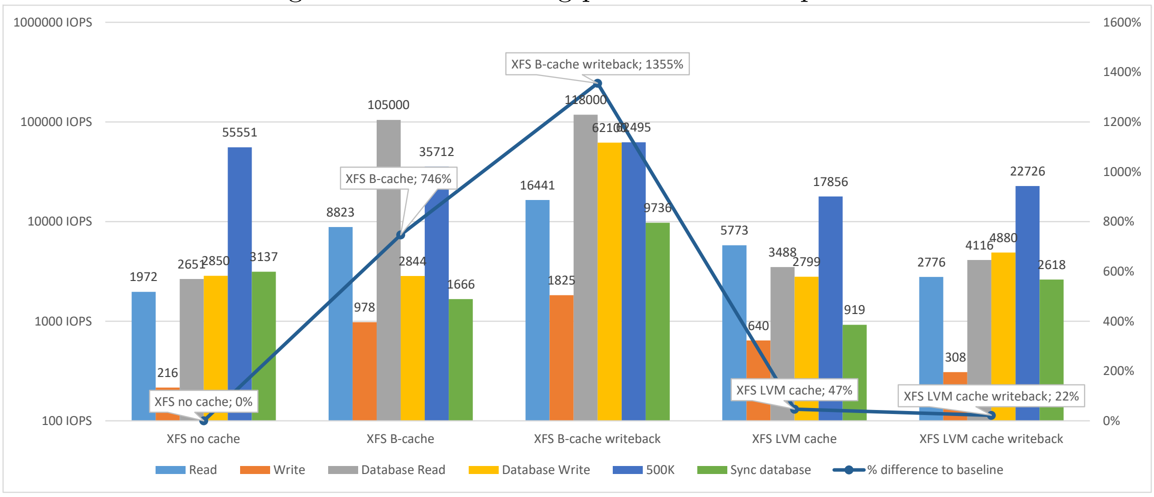
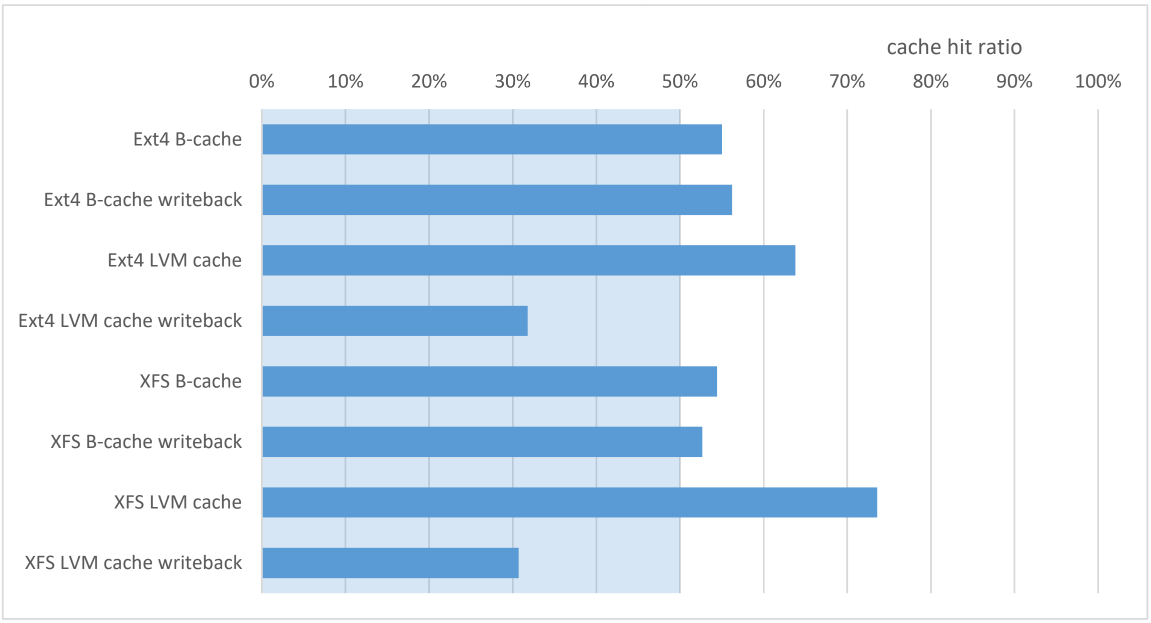
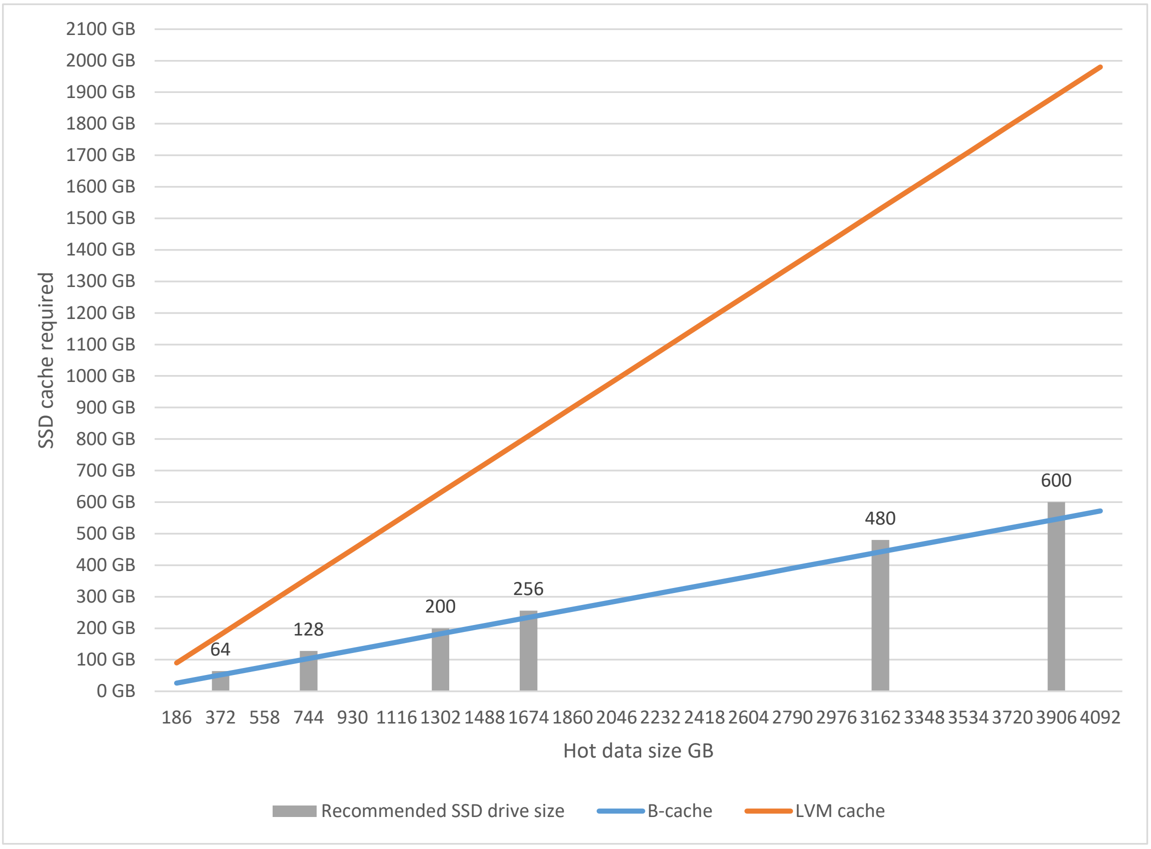


Figure 7.7: Cache hit/miss ratio



Another difference between B-cache and LVM cache is how much of the available cache does it actually need. B-cache required 4% of the available 650GB SSD, while LVM cache occupied much larger 13.8%. From these figures I extrapolated the following graph 7.8. It shows the relation between the SSD space occupied by B-cache and LVM cache respectively and the size of hot data that the cache is able to effectively cache. It also shows that B-cache is more conservative in the SSD utilization, minimizing writes to the SSD better than LVM cache.

Figure 7.8: Hot storage size and cache size requirements

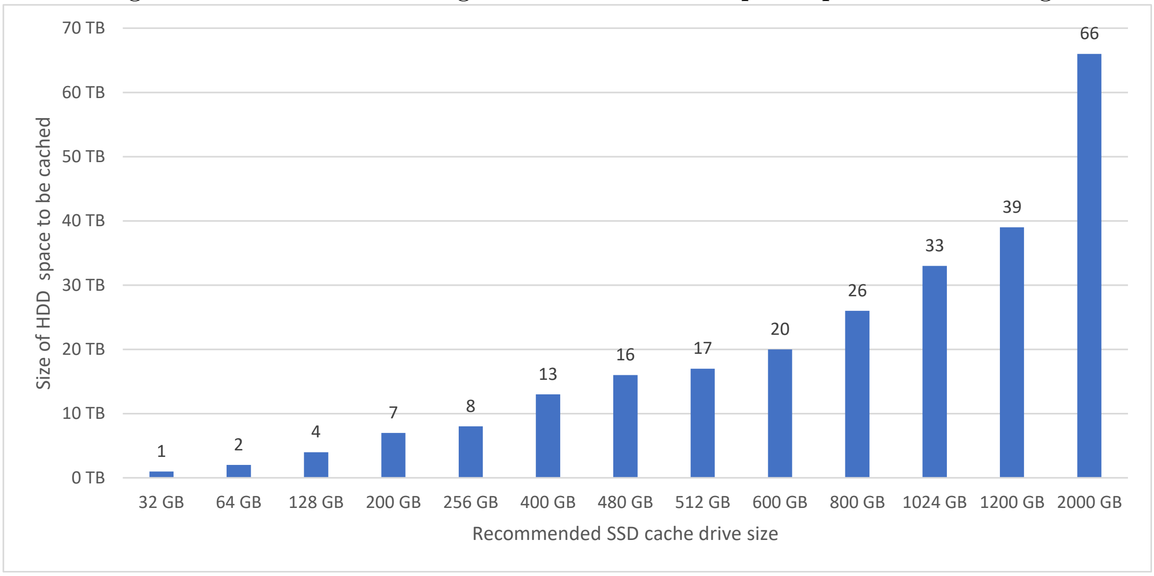


Based on the relation between cache size requirements and the available SSD disk size variants I formed a graph 7.9 showing the SSD drive size ranging from 32 to 2000GB and the ideal size of HDD storage this SSD would most effectively cache. The graph shows that really small SSDs are only viable for consumer space. That also explains why the first publicly available SSDs of the Intel 3D Xpoint (as introduced higher on page 11 in future technologies) are 16 and 32 GB in size. For larger arrays based on the storage requirements I would recommend SSDs in the 480-600 GB range due to higher performance, higher endurance and lower price/gigabyte. For higher capacity I would recommend skipping drives over 600 GB in size due to very high acquisition costs for now. SSD storage costs are constantly lowering.

CPU and drive utilization

In figure 7.10 we can observe the impact of caching on drive utilization and CPU usage reported by Fio in the database load. While LVM utilizes less than 20% SSD bandwidth in

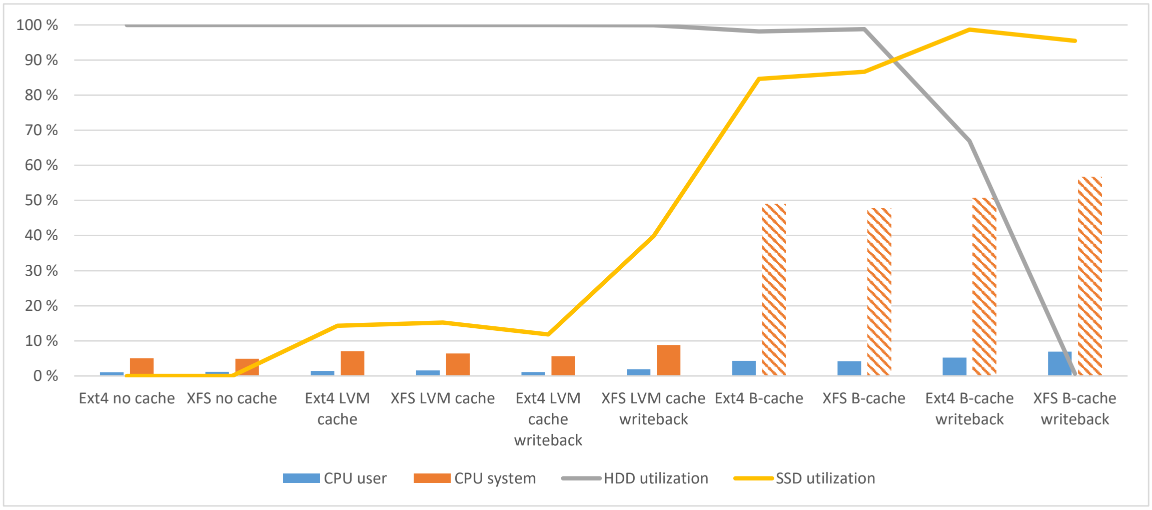
Figure 7.9: SSD cache storage size to total HDD space optimal for caching



writethrough mode and less than 40% in writeback mode, B-cache Utilizes SSD bandwidth from 84 up to 95% while lowering HDD utilization by 2% in writethrough mode and 33% and 98% in writeback mode. This clearly demonstrates why B-cache writeback mode is so much more effective in this load. While LVM doesn't confirm successful write until the data is on the HDD even in writeback mode (which leads to a high HDD utilization) B-cache reports writes immediately after writing into cache, making HDD writes delayed and HDD utilization lower.

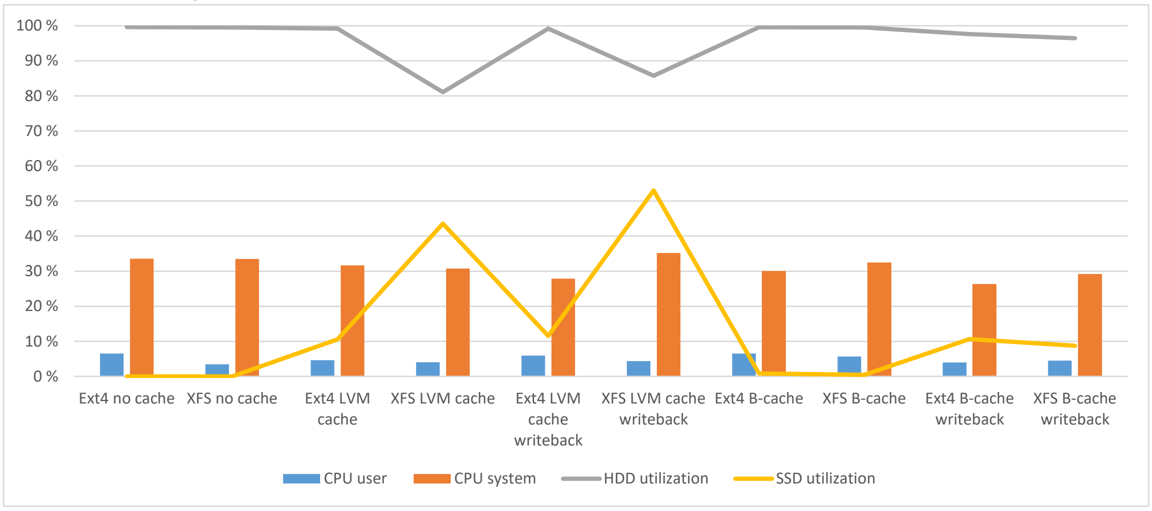
In the CPU utilization part it might look like B-cache is a huge CPU hog, according to the source [1] Centos/RHEL 7.X the CPU load reported by the OS is unrealistically high in some cases. I also observed CPU load through the iDRAC (integrated Dell Remote Access Controller) of the server and during testing it never went above 10% even with B-cache. On a system with newer version of B-cache this should be patched.

Figure 7.10: CPU and drive utilization comparison, database load



In another test (mixed access, results can be seen in graph 7.4) the drive utilization of XFS LVM cache and XFS LVM cache writeback mode coincide perfectly with the results of the tested scenario. In this test the reported CPU load of B-cache is correct and slightly (approx. 10-20%) lower than with the uncached array.

Figure 7.11: CPU and drive utilization comparison, mixed access



External journal

Moving the journal of a file system to a different (preferably faster) device is in theory an easy upgrade to performance.

Ext4 external journal

In order to mitigate filesystem performance penalty of Ext4 journal I conducted a test of Ext4 performance with journal located externally on a small SSD partition. As can be seen in the graph 7.12 Ext4 benefits from externally located journal especially in the rapid file creation scenario (labeled 500k in the graph) and periodically synchronised database. For an average increase of only 7% over internal journal the added configuration and higher possibility of failure is this setup therefore not recommended over higher introduced caching solutions.

XFS external log

Graph 7.13 shows the improvements of putting XFS log on an external SSD device. While performance did increase by 13% in the file creation scenario, external log offered worse performance than internal in read and write and minimal improvements otherwise. An average increase of 1% makes XFS external log not recommendable.

Price/performance

I took the price of the HDDs and SSDs from current listing and compared the price and performance of an uncached RAID 10 of 2x4 4TB HDDs and the same array with a single 800GB SSD to represent possible writethrough configuration, 2 800GB SSDs in RAID 1 for

Figure 7.12: Ext4 external journal

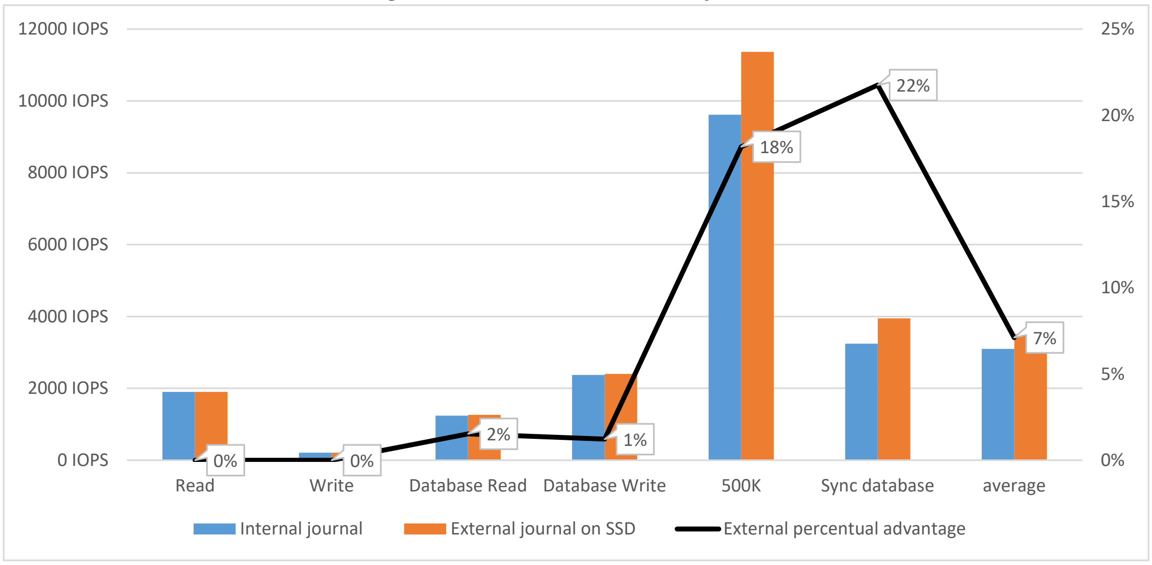
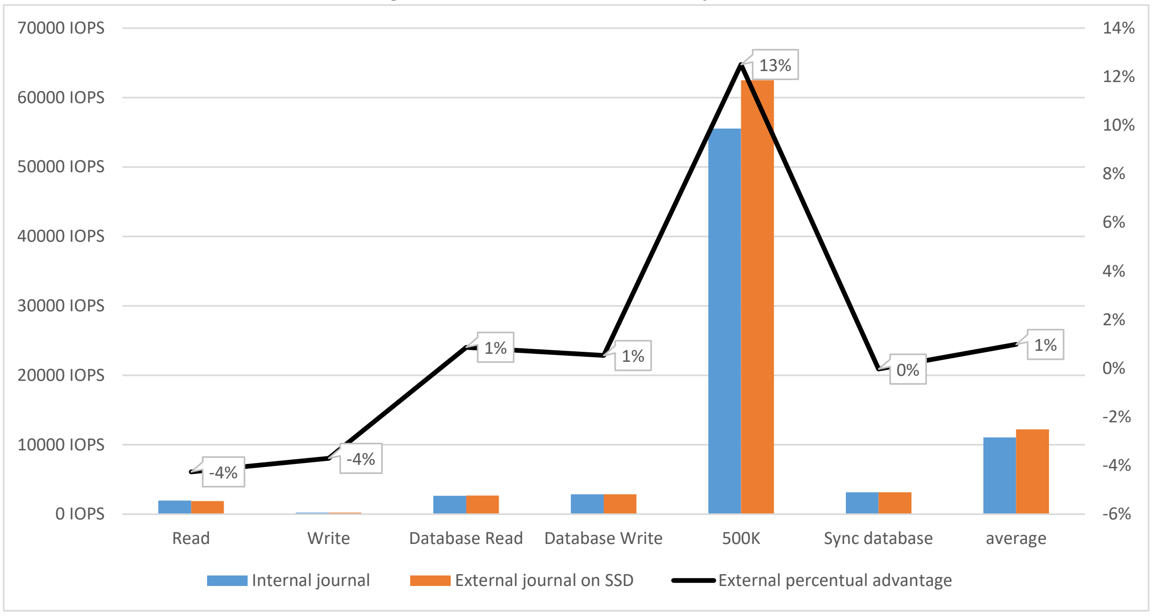


Figure 7.13: XFS external journal

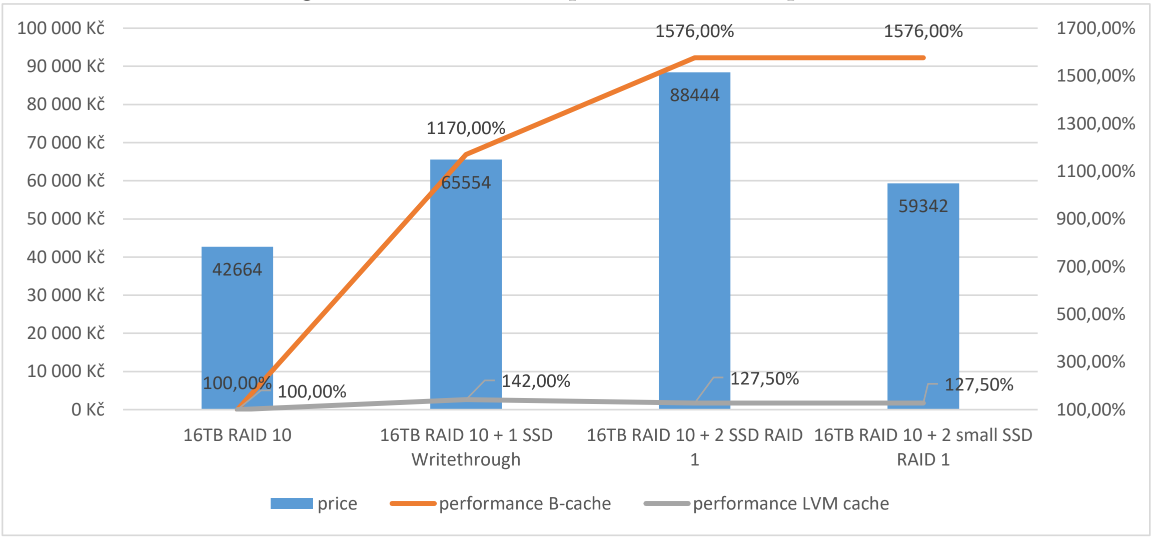


writeback and additional security and 2 cheaper, 480GB SSDs in RAID 1. If the SSD is in the same performance level (SATA with AHCI) the performance should be comparable. I chose more expensive datacenter focused drives due to their higher endurance and sustained performance. Commercial SSDs cost less but are only advisable for writethrough caching.

As can be seen in figure 7.14 B-cache writethrough mode offers over 10x the performance for 53% more cost than uncached array. Writeback B-cache mode offers even more, over 14x the performance, but for higher costs as well. For 107% more acquisition costs approximately 2.3TB worth of hot storage can be effectively sped up. If lower capacity drives are utilized than for 39% more costs than baseline only about 1.4TB of cache managed hot storage will be sped up, with performance close to that of the more expensive larger SSDs.

LVM cache on the other hand offers for the same price increase only modest performance gains. Overall performance gains are lower than the increase in price. However, for specific situations such as mixed sequential and random access as demonstrated in graph 7.4 or file creation speed in Ext4 (see graph 7.1) the 41% and 67% increase in performance may be justifiable.

Figure 7.14: Price and performance comparison



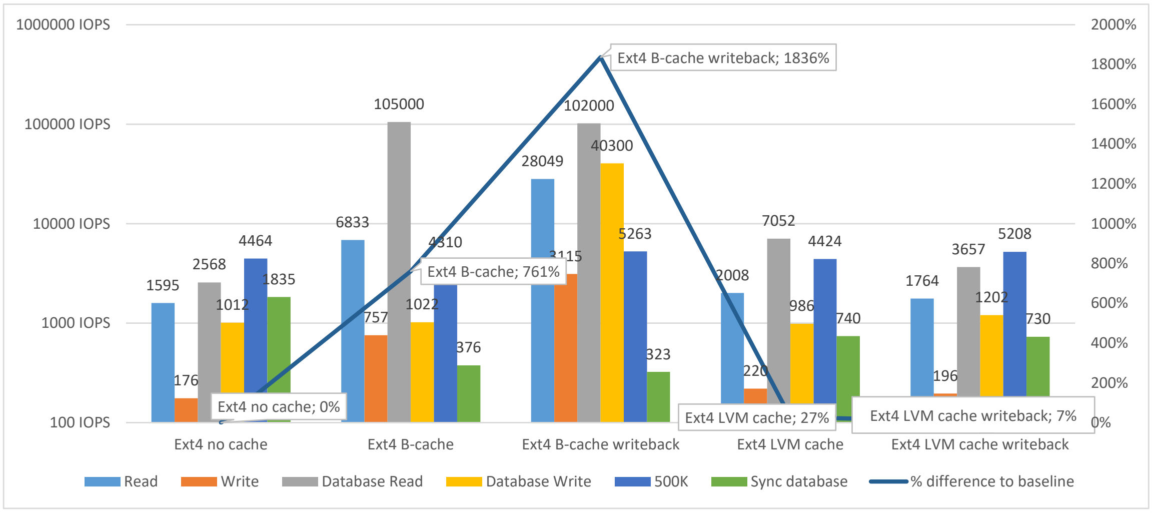
The results without RAID controller cache

All the above graphs and results were obtained with the 1GB DDR3 cache on the PERC H730 mini RAID controller turned on. I switched the RAID controller cache setting to writethrough which should be equivalent to absent cache.

Disabling the controller cache resulted in an average 11% decrease in IOPS in Ext4 compared to enabled cache. Surprisingly, disabling the controller cache actually increased the performance of B-cache in writeback mode in Read, Write and Database read. In file creation test it decreased compared to enabled controller cache, but managed to be faster than the uncached array, unlike in the previous testing. The Sync database, safe database with periodic sync() calls suffered the most from disabled controller cache.

As can be seen in graph 7.15, overall, the caching solutions formatted with Ext4 with controller cache disabled compared to the uncached array performed similarly to the setup with controller cache enabled. B-cache with writeback caching still dominates both the overall and most individual tests, with B-cache writethrough mode in the second place and LVM cache behind.

Figure 7.15: Ext4 caches performance with controller cache disabled



In XFS disabling the controller cache has a surprisingly high impact on the performance of B-cache writeback mode. As the graph 7.16 shows, XFS without controller cache is much more beneficial for writethrough modes of both B-cache and LVM cache. As far as LVM cache is concerned, it scored significantly better than with controller cache. While writethrough mode is ahead LVM cache writeback mode in average score, writeback offers much more consistent IOPS as well as higher write speeds.

Disabling the controller cache has the same impact on XFS performance as on Ext4 (as shown in figure 7.15). In XFS LVM cache and especially the writeback mode is affected the least.

Summary of results

The summarising graph of all the results with linear scale can be found in graph 7.17. You can see the spikes in performance that B-cache is able to provide in certain scenarios as well as general comparison of all test results.

7.2 Performance across server loads

Summarising across the different model server loads as introduced in chapter 3, we can observe the following.

Very small files

In the Very small files model server load we can observe 2 trends. Better filesystem support for fast allocation as well as better journal write performance of the XFS filesystem triumphs. In Ext4 the added complexity of most caching solutions can actually hurt performance significantly. In this scenario the best performing cache is the LVM cache in writeback mode, offering a respectable 67% increase in throughput.

Figure 7.16: XFS caches performance with controller cache disabled

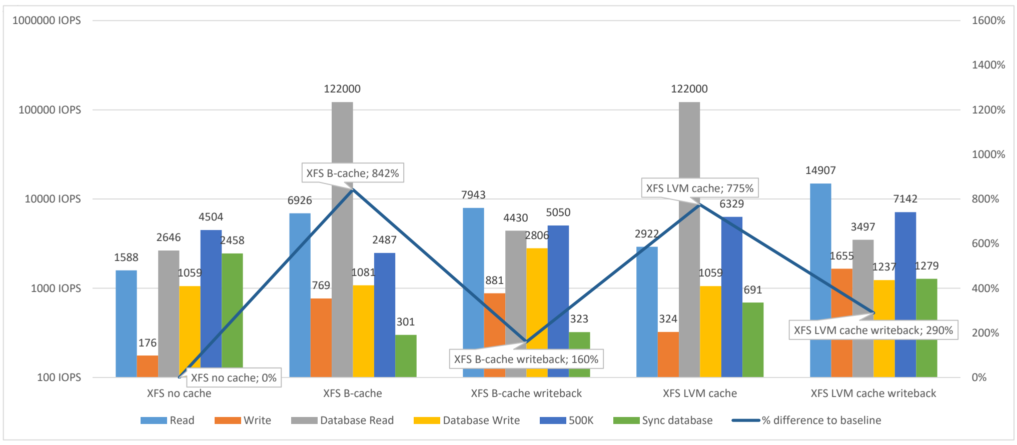
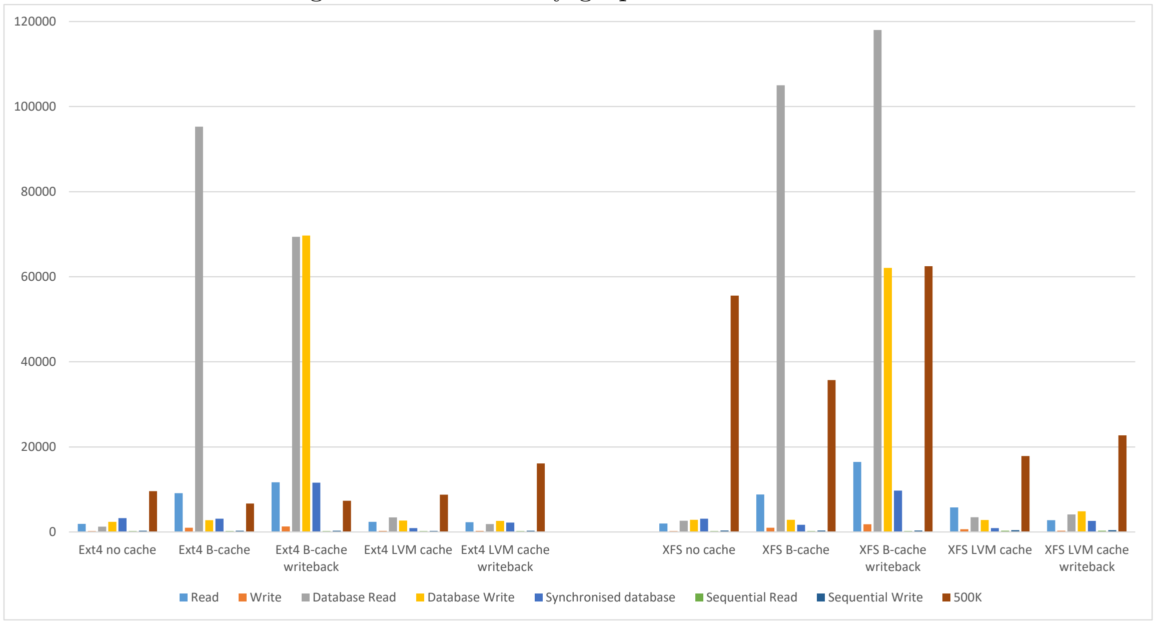


Figure 7.17: Summary graph of all test results



Small files

The small files scenario split into 2 different versions. The relatively low risk, high performance one (News sites, wikipedias, discussion forums etc. henceforth called general database) and the high availability, high risk safe databases (Booking sites, banking, operation critical database, will be referred to as secure database).

In the general database scenario B-cache in all its variations (Ext4 and XFS in writeback and writethrough modes) performed significantly better than LVM cache in read operations. The B-cache in writeback mode with Ext4 and XFS excelled in both read and write operations with 55x and 43x increase over the baseline in read and 28x and 20x increase in write operations over the baseline respectively.

The secure database is less effective, especially due to the fact that it forces the caches to be flushed regularly. The only cache that resulted in a performance increase was B-cache in writeback mode. It was better than uncached Ext4 array by 257% and by 210% better than XFS variant.

Mixed sized files hot storage

For this server load the mix of different files as well as random and sequential access was examined. The paragraph 7.1 with the graph 7.4 shows that this server load is best suited for XFS with LVM cache in both writethrough and writeback mode with slight 4.7% advantage in read and 3.6% advantage in write in favour of writeback.

Large files

Although not part of the tested scenarios the Mixed sized files tests also revealed information relevant to this potential load and the difference between B-cache and LVM cache.

B-cache features a sequential bypass mechanism that on one hand does not cache (and subsequently speed up the access to) the sequentially accessed data, but on the other hand protects the caching drive from rewriting large areas that can maintain the previously cached data.

Very large datasets

As stated in chapter 3 this type of load is not financially advisable for caching. In the event that money is not an obstacle consider either some sort of tiering as introduced in section 5.4 or identify the type of files stored and scale up the setup used in this paper considerably. (B-cache and LVM cache utilizing raid 0 of 2 or more drives may potentially yield even better results than in my testing.)

Chapter 8

Conclusion

The goal of this bachelor thesis was to examine the available HDD array acceleration with SSD drives. During the benchmarking I observed great differences between the tested caching solutions.

LVM cache is the more user friendly, offering easy management and integration into existing LVM managed environment. Ideal for servers that are already configured with LVM and want an extra bit more performance. Depending on the workload LVM can still provide about 23% average increase over uncached array making it an easy and affordable inplace upgrade.

By pure numbers B-cache triumphs over LVM cache by a large margin. B-cache is able to serialize random writes (especially in the writeback mode) and achieve performance an order higher than both uncached and LVM cached array. B-cache doesn't perform the best with Ext4 filesystem, in some cases (such as creation of a large amount of small files) falling below the speed of uncached array. XFS and writeback mode of B-cache dominate the tests. B-cache offers 7-13x the performance of uncached drive, provided that the hot storage size is small enough.

Writeback cache mode offers significantly better write speeds, at the cost of data security. An additional drive in a mirror configuration is recommended in order to prevent data loss. Writethrough is the more secure of the two caching modes and offers slightly lower, but still significant improvements.

Further work on SSD caching could investigate caching at filesystem level (ZFS L2ARC or bcachefs) and analyse NVMe SSD performance with caching solutions.

Bibliography

[1] *Solid state drive caching to speed up your spinning drives.* [Online; visited 25.04.2017].
Retrieved from: https://www.3pillarglobal.com/insights/solid-state-drive-caching-to-speed-up-your-spinning-drives

[2] Axboe, J.: *Fio Readme.* [Online; visited 20.02.2017].
Retrieved from: https://github.com/axboe/fio

[3] *How long do drives last.* [Online; visited 25.01.2017].
Retrieved from:
https://www.backblaze.com/how-long-do-disk-drives-last.html

[4] *Bcache user documentation.* [Online; visited 7.02.2017].
Retrieved from: https://www.kernel.org/doc/Documentation/bcache.txt

[5] eProvided: *SSD drive failure - What causes Solid State Drives to stop working and fail?* [Online; visited 6.02.2017].
Retrieved from: http://www.eprovided.com/data-recovery-blog/ssd-drive-failure-causes-solid-state-drive-stop-working-failures/

[6] *Everspin Announces nvNITRO^{TM} NVMe Storage Accelerator Family.* [Online; visited 25.04.2017].
Retrieved from: https://www.everspin.com/news/everspin-announces-nvnitro%E2%84%A2-nvme-storage-accelerator-family

[7] *Ext4 Howto.* [Online; visited 14.03.2017].
Retrieved from: https://ext4.wiki.kernel.org/index.php/Ext4_Howto

[8] Gasior, G.: *The SSD endurance experiment.* [Online; visited 28.01.2017].
Retrieved from: http://techreport.com/review/27909/the-ssd-endurance-experiment-theyre-all-dead

[9] *3D NAND Technology animation.* [Online; visited 12.03.2017].
Retrieved from: http://www.intel.com/content/www/us/en/solid-state-drives/3d-nand-technology-animation.html

[10] *Intel Optane SSD DC P4800X Product brief.* [Online; visited 25.04.2017].
Retrieved from: http://www.intel.com/content/www/us/en/solid-state-drives/optane-ssd-dc-p4800x-brief.html

[11] *Device mapper cache policies.* [Online; visited 21.04.2017].
Retrieved from: https://www.kernel.org/doc/Documentation/device-mapper/cache-policies.txt

[12] *Re: btrfs und lvm-cache?* [Online; visited 27.04.2017].
Retrieved from: https://www.spinics.net/lists/linux-btrfs/msg50427.html

[13] *LVM cache man page*. [Online; visited 7.02.2017].
Retrieved from: http://man7.org/linux/man-pages/man7/lvmcache.7.html

[14] *Azure Blob Storage: Hot and cool storage tiers*. [Online; visited 25.01.2017].
Retrieved from: https://docs.microsoft.com/en-us/azure/storage/storage-blob-storage-tiers

[15] *Storage Space Tiering*. [Online; visited 15.03.2017].
Retrieved from: https://blogs.technet.microsoft.com/larryexchange/2015/12/02/understand-storage-space-tiering-in-windows-server-2012-r2/

[16] *Data classification: Key to a successful Tiered Storage Strategy*. [Online; visited 25.01.2017].
Retrieved from: http://www.mosaictec.com/pdf-docs/whitepapers/DataClassification-TieredStorage.pdf

[17] *V-NAND Technology*. [Online; visited 12.03.2017].
Retrieved from: http://www.samsung.com/semiconductor/products/flash-storage/v-nand/

[18] *SanDisk DAS Cache*. [Online; visited 22.03.2017].
Retrieved from: https://www.dell.com/en-us/work/learn/server-technology-components-sandisk-das-cache

[19] *A brief history of XFS*. [Online; visited 20.02.2017].
Retrieved from: http://xfs.org/docs/xfsdocs-xml-dev/XFS_User_Guide/tmp/en-US/html/ch01s02.html

[20] Staimer, M.: *Tiered data backup storage strategies*. [Online; visited 25.01.2017].
Retrieved from: http://searchdatabackup.techtarget.com/tip/Tiered-data-backup-storage-strategies

[21] Tarasov, V.: *Filebench Readme*. [Online; visited 13.03.2017].
Retrieved from: https://github.com/filebench/filebench

Appendix A

Contents of the memory media

/data/ Excel and txt outputs of tests, each combination of filesystem and caching solution in a subfolder
/documentation/ Latex source version of this bachelor thesis
/tests/ Test input files
xbelou03-IO_Subsystem.pdf pdf version of this bachelor thesis

Appendix B

Iozone test results

In the following graph you can see the IOzone results of caching solutions. To allow for better visibility and easier comparison I scaled all graphs containing read results from 0 to 25 million and all graphs containing write results from 0 to 8 million. As you can see on an example figure B.1, all graphs are color coded to show the different value ranges and contain an unmeasured subset. Please note that IOzone test did not include pre-warming of the caches, therefore represents a cold access.

Figure B.1: Example annotated graph

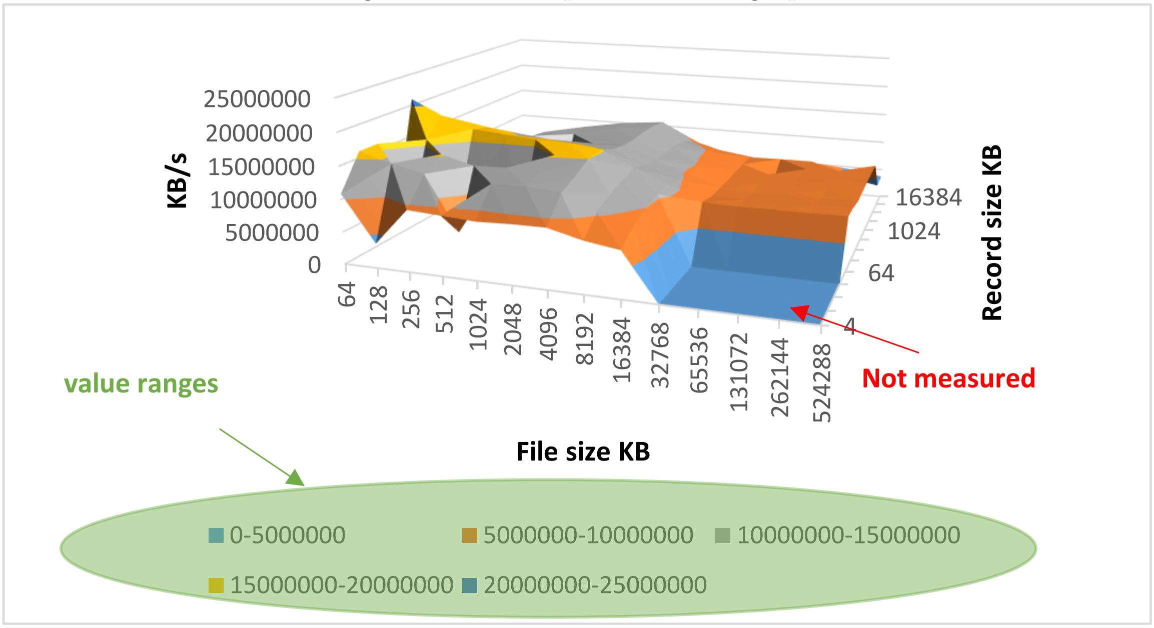


Figure B.2: Ext4 random read no cache

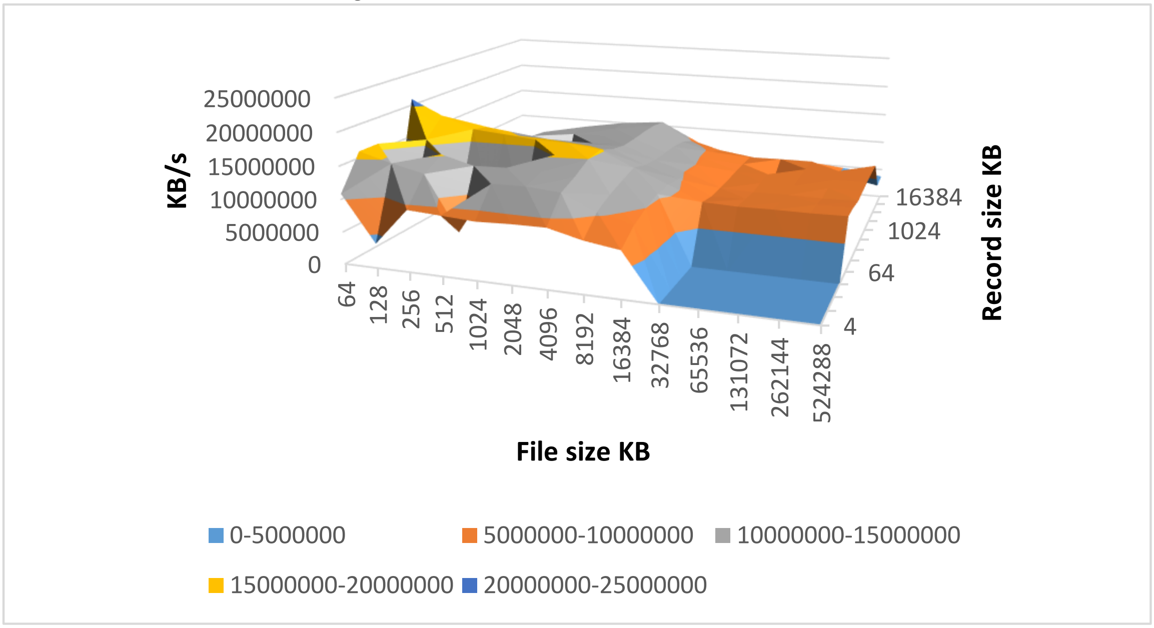


Figure B.3: Ext4 random read LVM cache

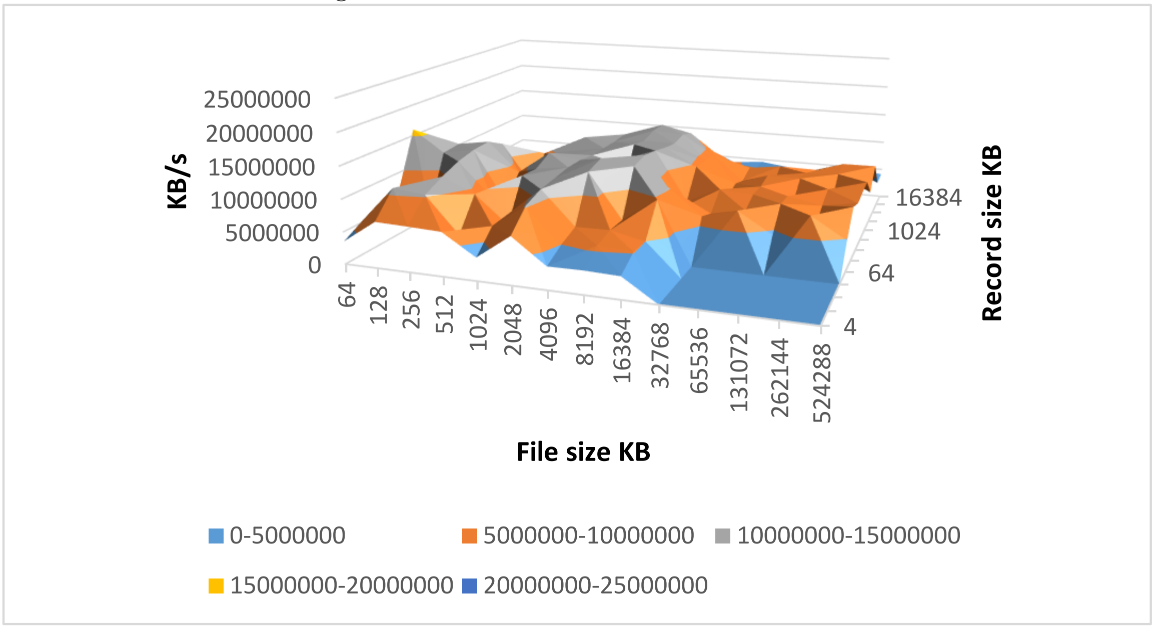


Figure B.4: Ext4 random read LVM cache writeback

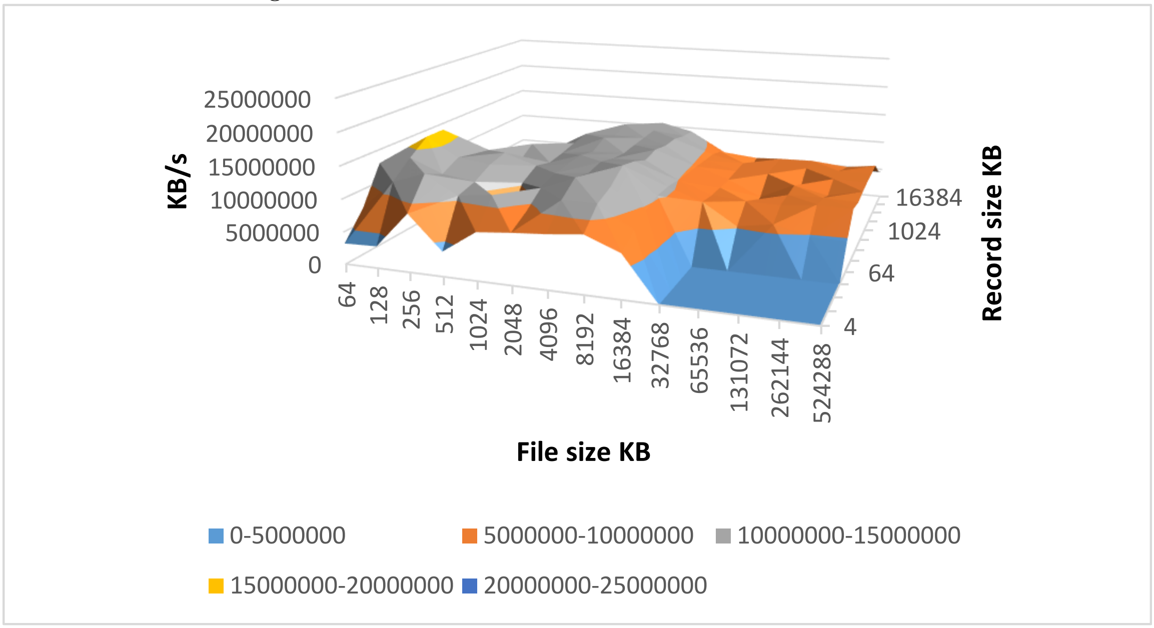


Figure B.5: Ext4 random read B-cache

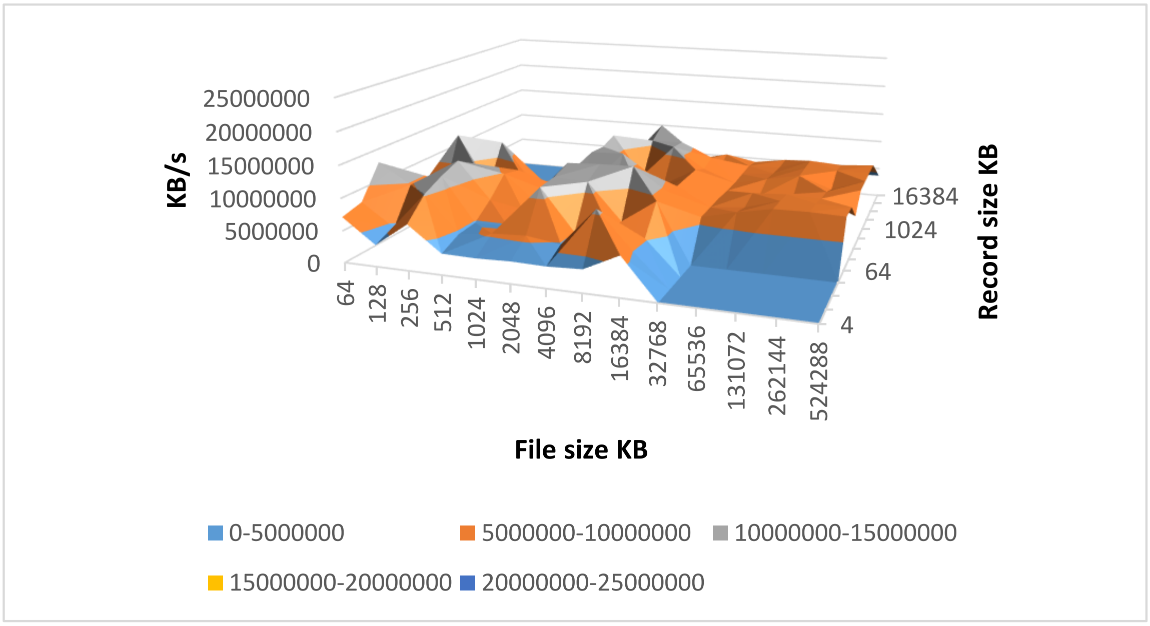


Figure B.6: Ext4 random read B-cache writeback

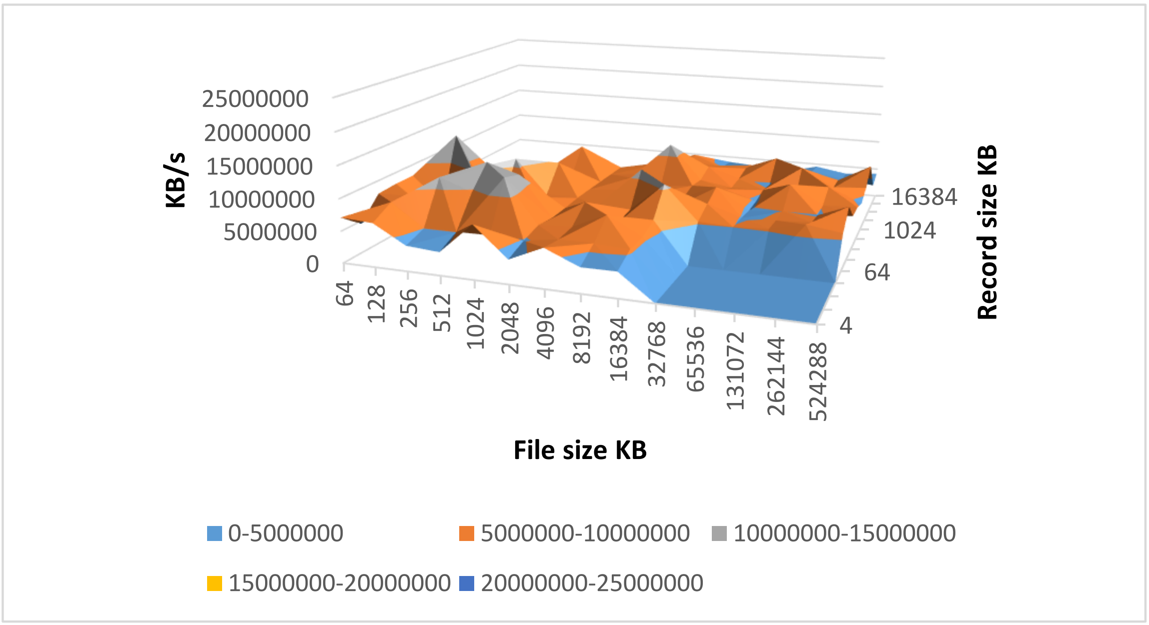


Figure B.7: XFS random read no cache

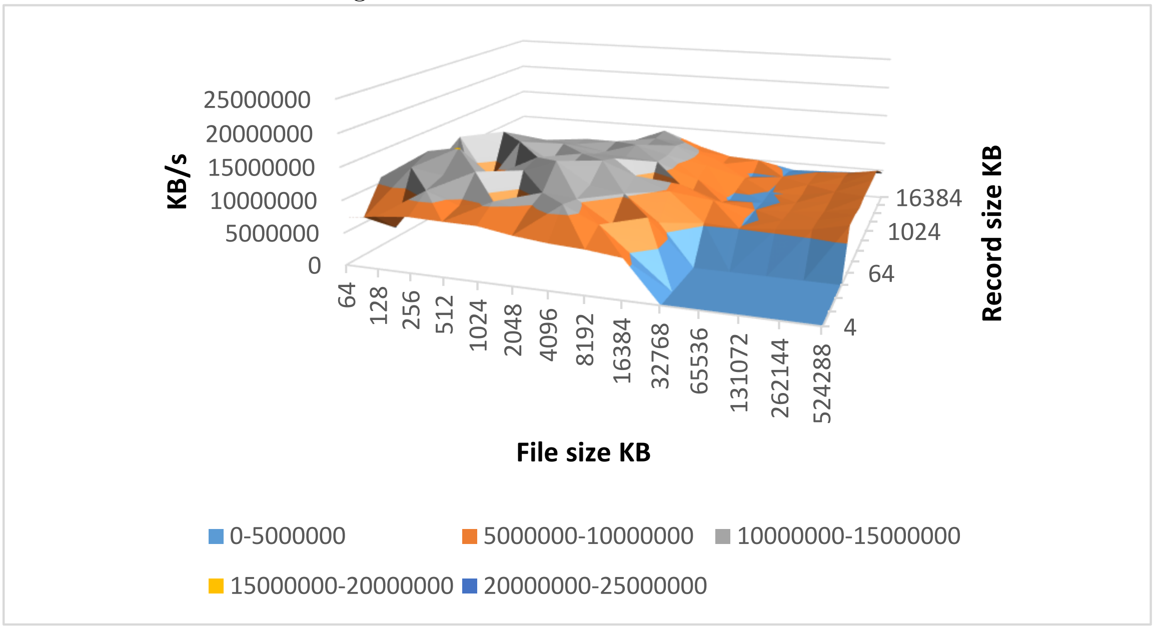


Figure B.8: XFS random read LVM cache

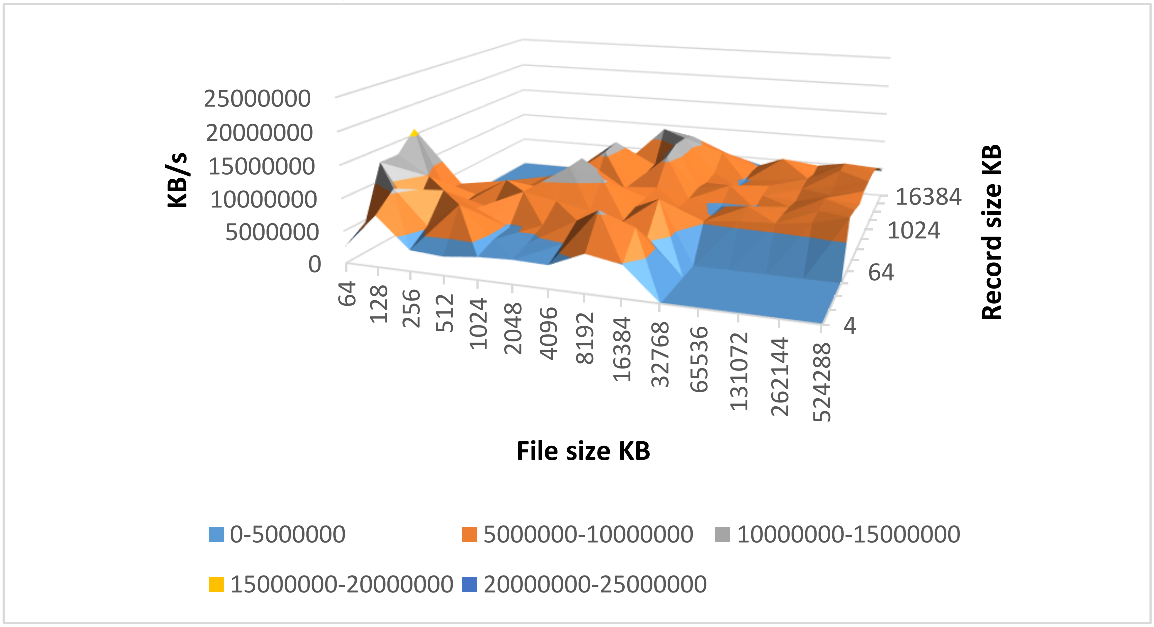


Figure B.9: XFS random read LVM cache writeback

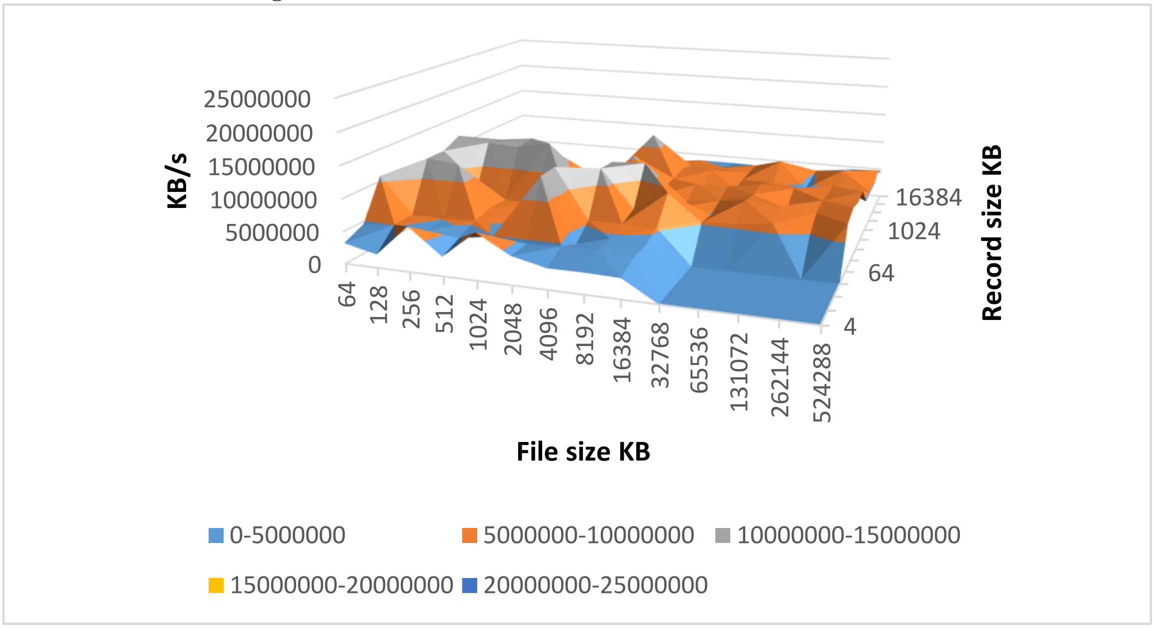


Figure B.10: XFS random read B-cache

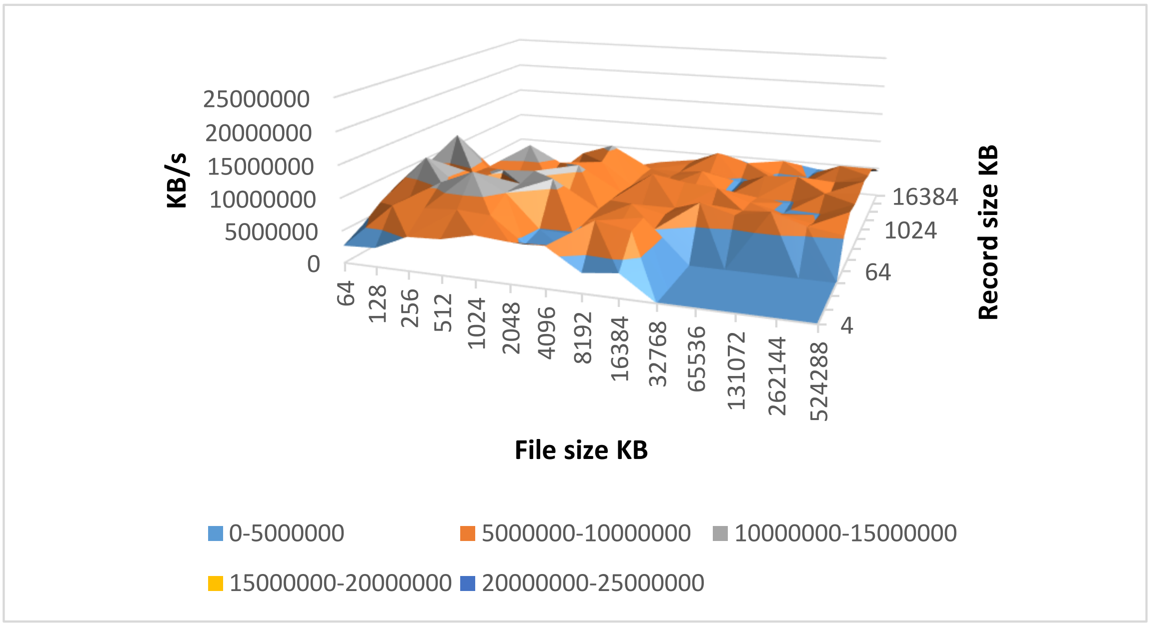


Figure B.11: XFS random read B-cache writeback

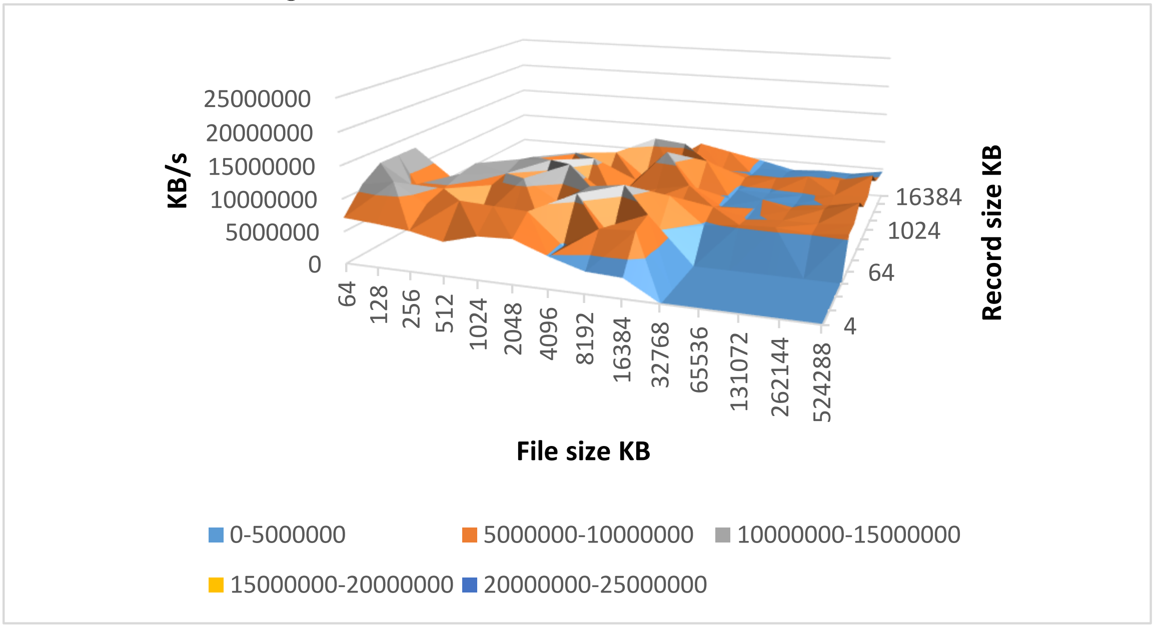


Figure B.12: Ext4 random write no cache

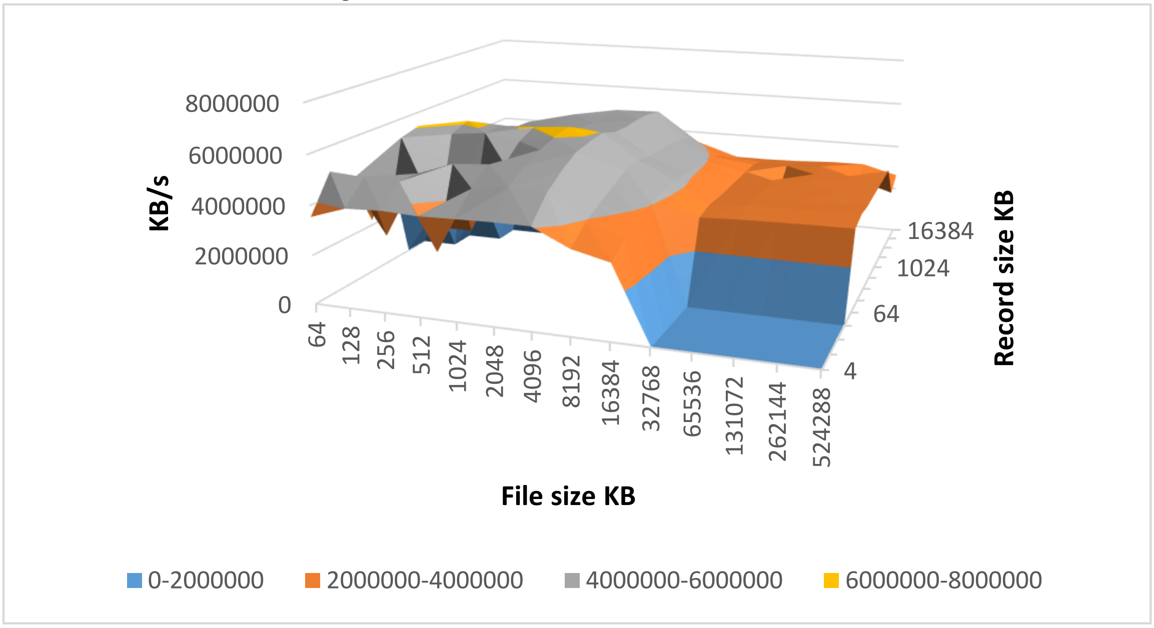


Figure B.13: Ext4 random write LVM cache

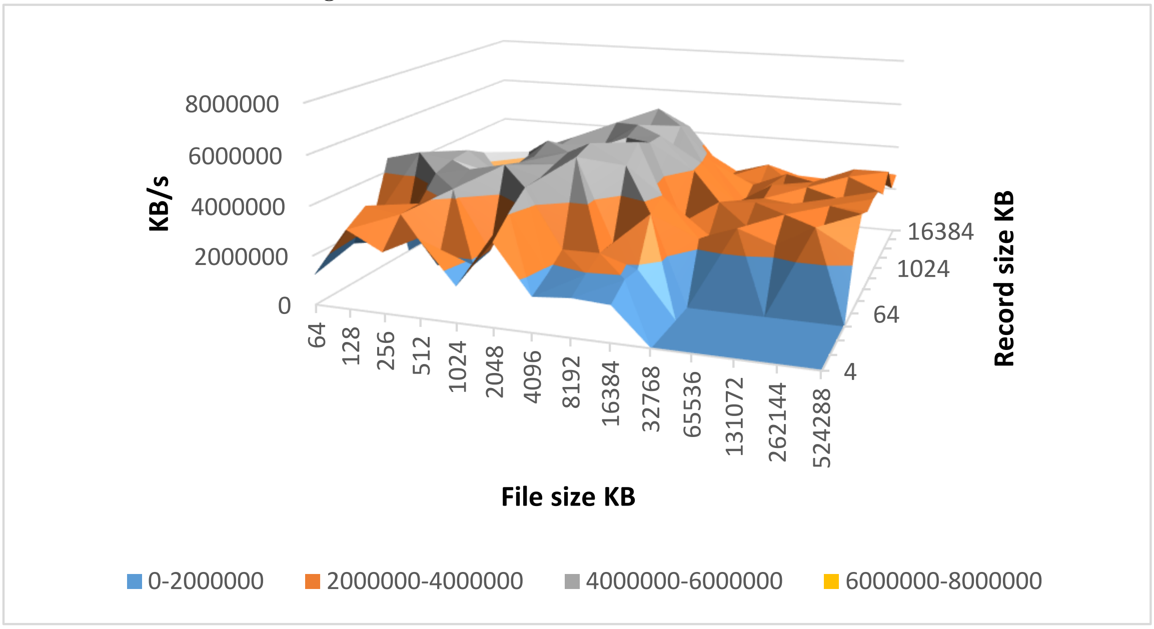


Figure B.14: Ext4 random write LVM cache writeback

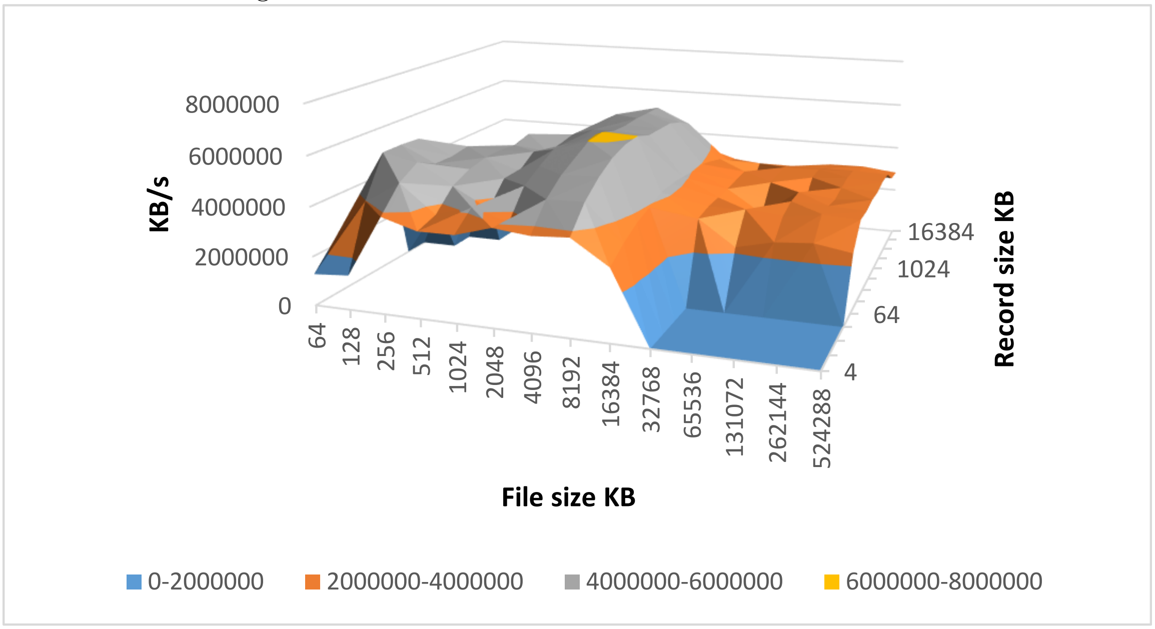


Figure B.15: Ext4 random write B-cache

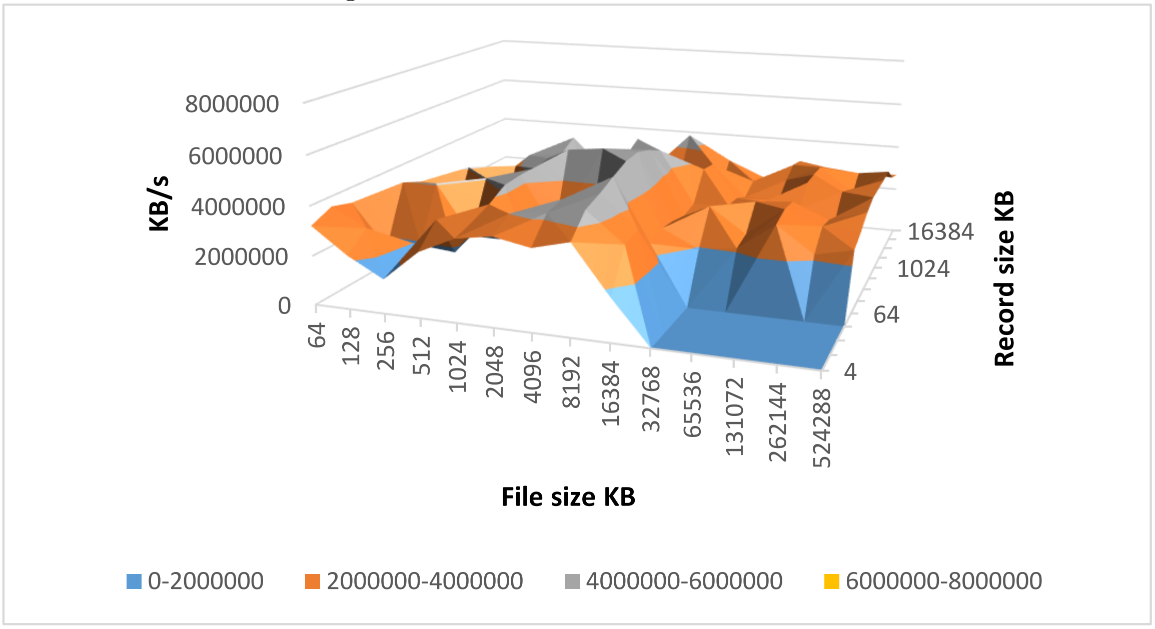


Figure B.16: Ext4 random write B-cache writeback

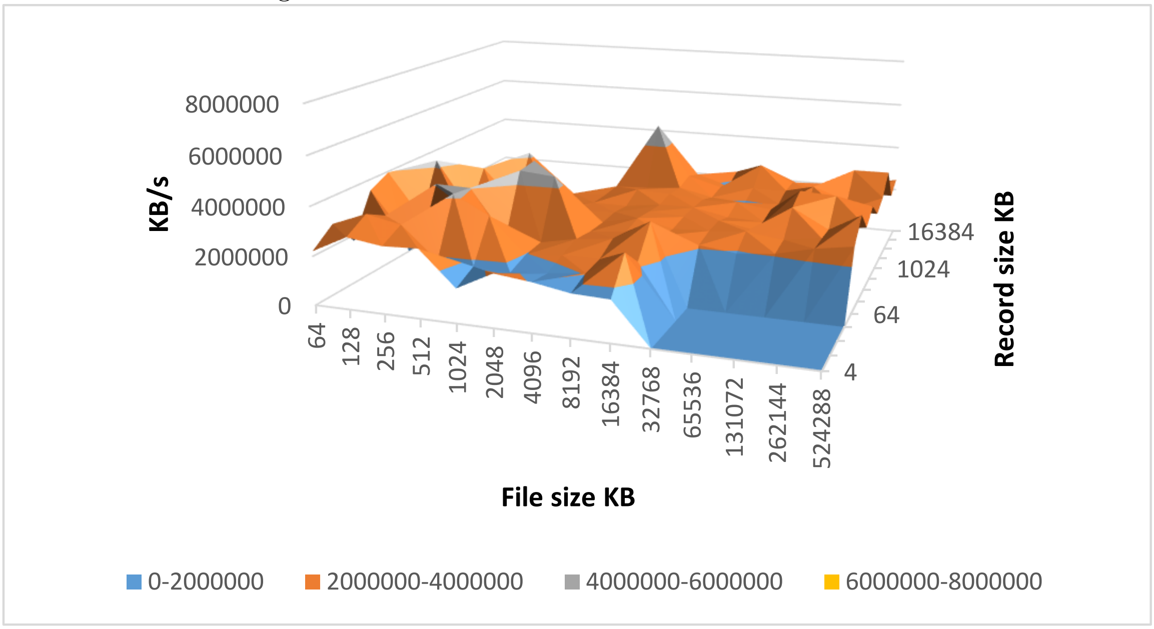


Figure B.17: XFS random write no cache

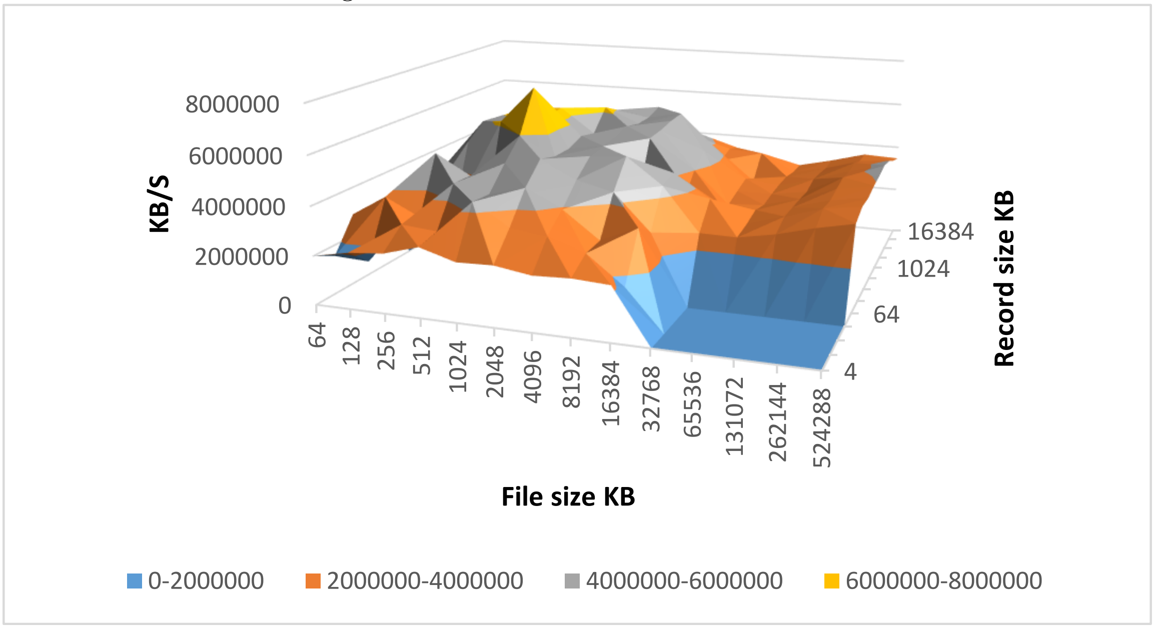


Figure B.18: XFS random write LVM cache

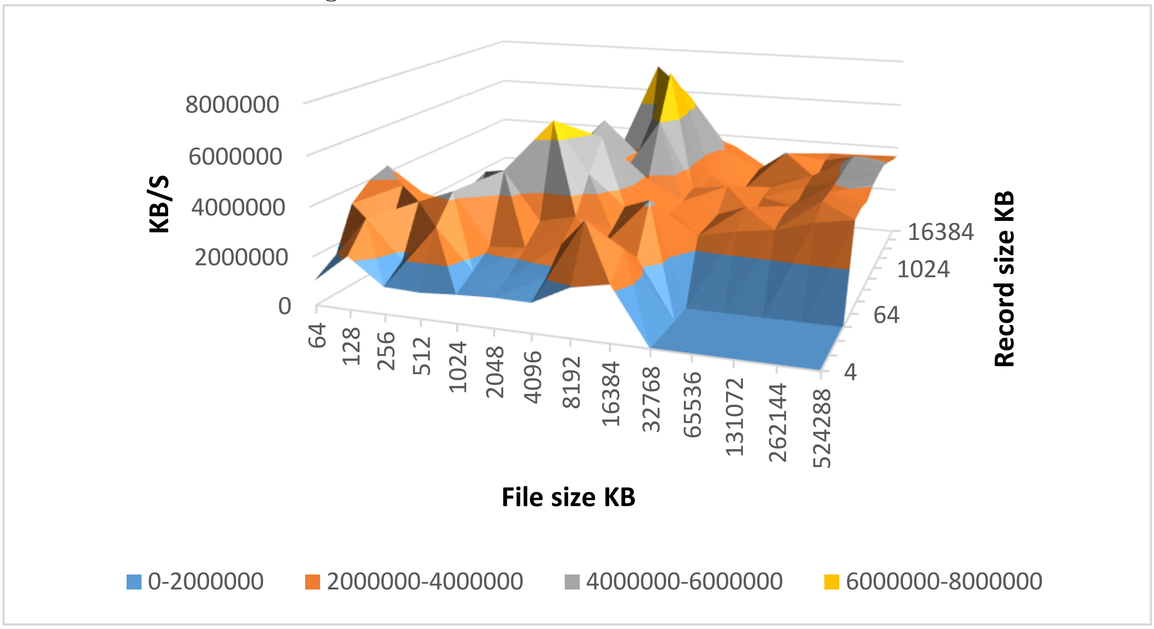


Figure B.19: XFS random write LVM cache writeback

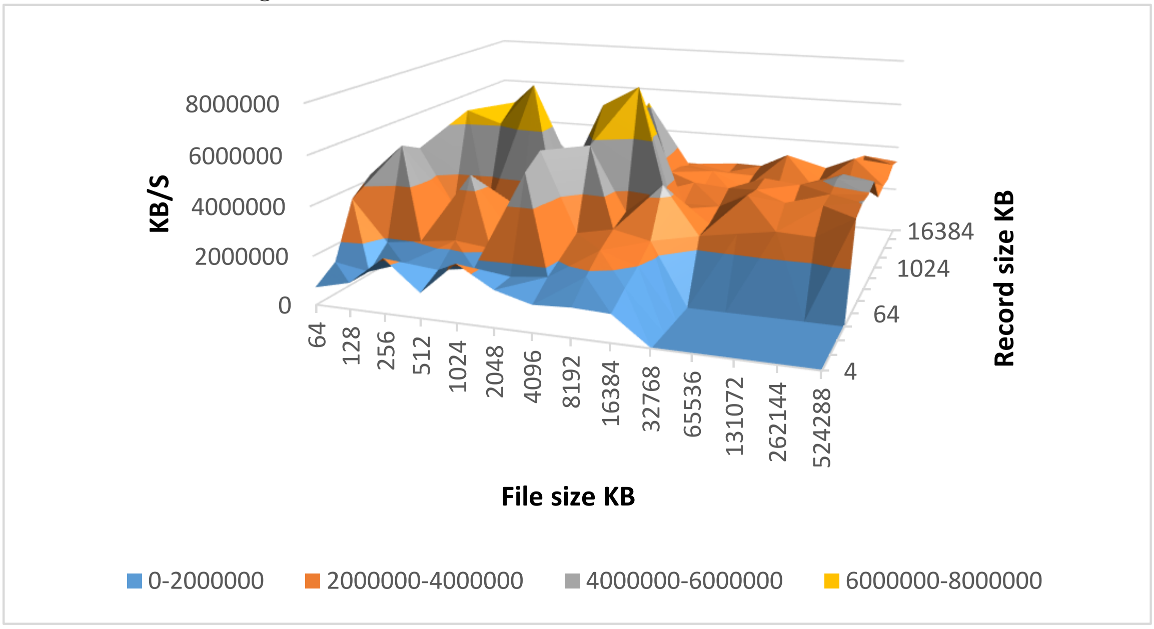


Figure B.20: XFS random write B-cache

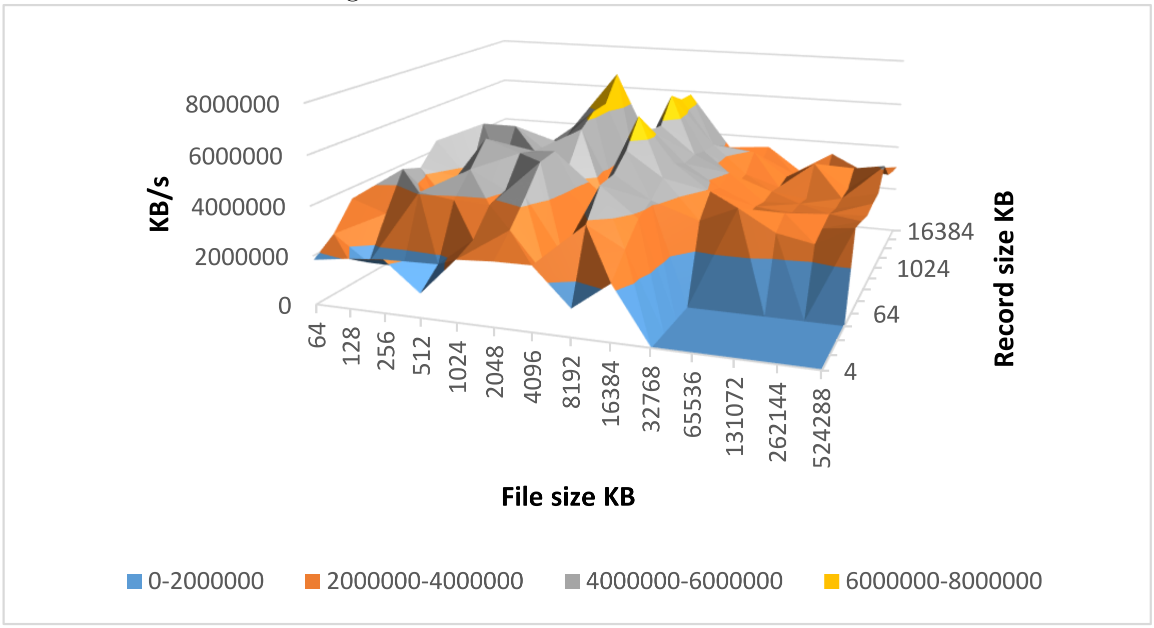
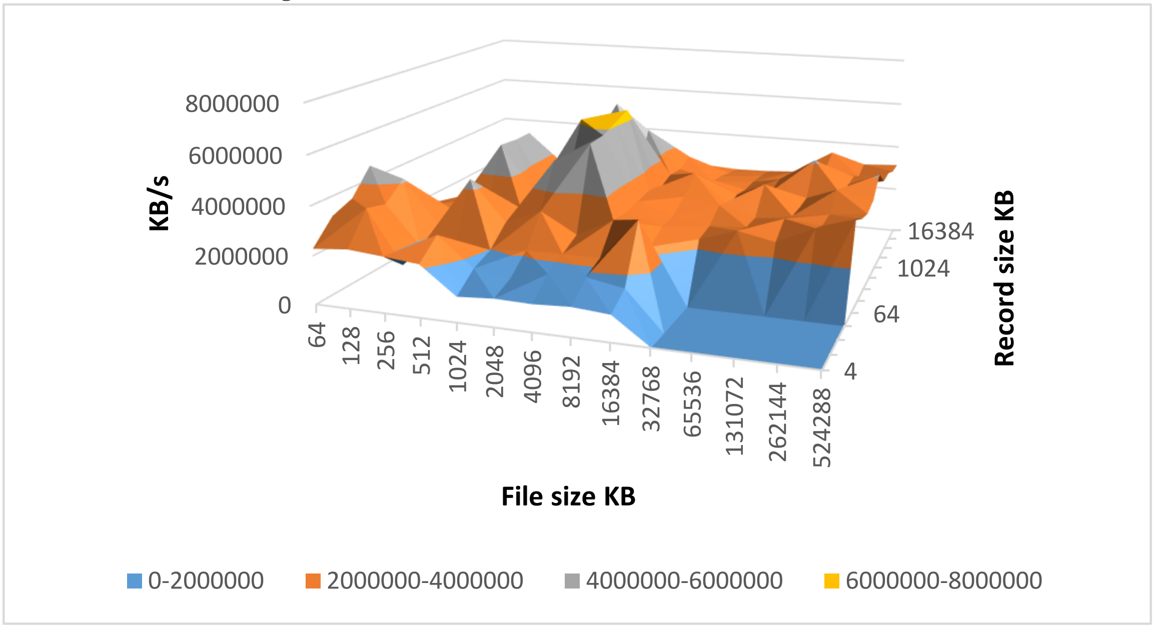


Figure B.21: XFS random write B-cache writeback


KB/s
File size KB
Record size KB