ENHANCING SYSTEM PERFORMANCE USING PERSISTENT RAM MODULES AS STORAGE CLASS MEMORY

TEBRA A MOUSSA JUMMAH

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR

2022

ENHANCING SYSTEM PERFORMANCE USING PERSISTENT RAM MODULES AS STORAGE CLASS MEMORY

TEBRA A MOUSSA JUMMAH

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITI MALAYA
KUALA LUMPUR

2022

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Tebra A Moussa Jummah**

Matric No: **17057630/1**

Name of Degree: **Master of Computer Science**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Enhancing System Performance Using Persistent RAM Modules as Storage Class Memory

Field of Study: **Computer System and Technology**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work.
(2) This Work is original.
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work.
(4) I do not have any actual knowledge, nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work.
(5) I hereby assign all and every right in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained.
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action, or any other action as may be determined by UM.

Candidate's Signature: Date: 20 Jan 2022

Subscribed and solemnly declared before,

Witness's Signature : Date: 20 Jan 2022

Name:

Designation:

ENHANCING SYSTEM PERFORMANCE USING PERSISTENT RAM MODULES AS STORAGE CLASS MEMORY

ABSTRACT

Throughput and time latency are critical performance metrics of most application systems; thus, any underlying storage technology must provide the best of both metrics. Since hard disk drive (HDD) and solid-state drive (SSD) became an I/O bottleneck performance for most intensive data applications, a new storage technology must be produced to address this issue. Storage class memory (SCM) emerged as the new promising technology with byte-addressable, high access time, and persistent features. In this research, an SCM emulator was implemented using a kernel module based on a RAM named ZRAM, which served as a general-purpose RAMDISK feature with persistence. Further, a test has been conducted on the implemented emulator with persistent RAMDISK and PMEM. The experiments conducted in this research were done through three stages: the first of which involved testing the workload within different data placement devices; the second stage involved running of tests upon a collection of disk filesystems, RAM filesystem, and persistent memory filesystem (PMFS). At the third stage, experiments were conducted to examine the effect of moving data files on performance. The implemented emulator persistent ZRAM (PZRAM) achieved superior performance as compared to HDD and SSD with a performance improvement of 14290% and 1167% respectively, a slide higher performance than PMEM with 2.3% improvement and almost similar performance of persistent RAMDISK. Additionally, the proposed PZRAM with TMPFS running on top of it has provided better performance with 11.83% over PZRAM with ext4. Further, this research provided comparative experiments on the effect of filesystem and moving data files on throughput and latency performance.

Keywords: Storage class memory (SCM), Hard disk drive (HDD), Solid-state drive (SSD), Filesystems and Performance.

MENINGKATKAN PRESTASI SISTEM MENGGUNAKAN MODUL RAM BERTERUSAN SEBAGAI STORAGE CLASS MEMORY

ABSTRAK

Truput dan latency masa adalah metrik prestasi kritikal bagi kebanyakan sistem aplikasi sehingga mana-mana teknologi penyimpanan mesti memberikan yang terbaik dari kedua metrik tersebut. Oleh kerana pemacu cakera keras (HDD) dan pemacu keadaan pepejal (SSD) menjadi prestasi penghambat I/O untuk kebanyakan aplikasi data intensif, teknologi penyimpanan baru mesti dihasilkan untuk mengatasi masalah ini. Memori kelas storan (SCM) adalah teknologi baru yang menjanjikan ini dengan pengalamatan byte, akses yang tinggi dan ciri-ciri berkekalan. Dalam penyelidikan ini, kami telah menerapkan emulator SCM menggunakan modul kernel bernama ZRAM sebagai fitur umum RAMDISK dengan ciri-ciri berkekalan. Selanjutnya, kami telah menguji emulator yang kami laksanakan dengan RAMDISK dan PMEM yang kekal. Eksperimen kami dilakukan melalui tiga peringkat; pertama: menguji beban kerja dalam peranti penempatan data yang berbeza, kedua: ujian dijalankan pada kumpulan sistem fail cakera, sistem fail RAM dan sistem fail memori berterusan (PMFS). Pada tahap ketiga, eksperimen adalah untuk mengkaji pengaruh memindahkan fail data terhadap prestasi. Emulator ZRAM yang kami laksanakan (PZRAM) telah menunjukkan prestasi yang unggul berbanding dengan HDD dan SSD dengan peningkatan prestasi masing-masing 14290% dan 1167%, prestasi yang lebih tinggi berbanding PMEM dengan peningkatan 2.3% dan prestasi RAMDISK berterusan yang hampir serupa. Selain itu, PZRAM yang kami cadangkan dengan TMPFS telah memberikan prestasi yang lebih baik dengan 11.83% berbanding PZRAM dengan ext4. Selanjutnya penyelidikan ini memberikan eksperimen perbandingan mengenai pengaruh sistem fail dan memindahkan fail data terhadap prestasi throughput dan latensi.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Almighty Allah for the help and in easing my journey throughout the study period to achieve this stage.

Secondly, I would like to express my sincere gratitude to both of my supervisors namely, Assoc. Prof. Dr. Rosli Salleh and Dr. Anjum Naveed for their guide, understanding, patience, help, and support throughout my thesis journey.

Next, I would like to thank my dearest family for their endless support, their kind prayers, and their wishes. Special thanks to my Brother Adam Abubaker Moussa for standing with me throughout this journey.

Furthermore, I owe so much thanks to my friends who were with me in this journey through their prayers and kind wishes. More importantly, special thanks to Mashahi Khalafalla, a friend of mine who helped me a lot during my stay in my home country with registration and other related matters.

Finally, I would like to thank all the people I have come across during this journey, who have helped or taught me in one way or the other. Also, I would like to appreciate the FSKTM staff for their kind and responsive attitudes during this period. In conclusion, I would express my sincere gratitude to Mrs. Norhazariah Binti Husin for her kind assistance throughout this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

Btrfs	:	B-tree filesystem
datadir	:	MYSQL data directory
DBMS	:	Database management system
DBW	:	InnoDB double-write buffer
DRAM	:	Dynamic random-access memory
Ext4	:	Fourth extended filesystem
F2FS	:	Flash-friendly filesystem
HDD	:	Hard drive disk
ibdata	:	Innodb system tablespace
MRAM	:	Magnetic random-access memory
NVDIMM	:	Non-volatile dual inline memory module
NVM	:	Non-volatile memory
NVRAM	:	Non-volatile random-access memory
PCM	:	Phase change memory
PM	:	Persistent memory
PMEM	:	Persistent memory emulator
PMFS	:	Persistent memory filesystem
PRAMDISK	:	Persistent random access-memory disk
RAM	:	Random access-memory
RERAM	:	Resistive random-access memory
SCM	:	Storage class memory
SRAM	:	Static random-access memory
SSD	:	Solid-state drive
STT_MRAM	:	Spin-transfer-torque magnetic random-access memory

TMPFS	:	Temporary storage filesystem
TPC-C	:	Online transaction processing benchmark
XFS	:	Extent's filesystem
ZRAM	:	A compressed random-access memory module

CHAPTER 1: INTRODUCTION

1.1 Background

For decades, hard disk drives (HDD) have been used as primary storage for most database management systems (DBMS) (Oukid & Lersch, 2019). However, the massive increase in present-day data volume alongside the need to access this data in a very short time has led the IT companies and database vendors the search for new storage media that can satisfy the requirements for all kinds of enterprises (Gaonkar, Bojewar, & Das, 2013).

To meet this ever-increasing number of data generated and processed by the present-day business companies; social media, banks, etc., as well as storage and memory industries, have experienced rapid growth in recent years to deliver numerous kinds of novel memory and storage to handle data and users/vendors' demands (Ouyang, Nellans, Wipfel, Flynn, & Panda, 2011).

This new revolution in the storage/memory industry has changed the traditional memory hierarchy which consisted of various tiers of memory technologies, sorted from the fastest and least capacity to the slowest and the highest capacity: CPU registers, cache memories (SRAM), main memory (DRAM), secondary storage (HDD), and the tertiary storage (Tape Drive) (Ma et al., 2016).

Solid-state drives (SSD) are the first storage technology leap introduced by semiconductor storage industries. It is a flash-based storage technology that functions as secondary storage. SSD offers an order of magnitude better performance than its predecessors: Hard disk drive (HDD) and magnetic tapes (Meza, Wu, Kumar, & Mutlu, 2015).

After the SSD leap, semiconductor storage and memory technologies showed a huge revolution in the market by delivering a collection of different storage technologies that

possessed the best metrics of both memory and storage technologies (Natarajan, 2004).

This collection of technologies is known as storage class memories (SCM). SCM is a non-volatile, byte-addressable memory that comes in different technologies introduced by different vendors. Some examples of SCM are phase-change memory (PCM), spin-transfer-torque magnetic random-access memory (STT-MRAM), and resistive random access memory (RRAM) (Natarajan, 2004). More details will be delivered in Chapter 2 of this dissertation.

SCM is also known as persistent memory (PM), non-volatile RAM (NVRAM), and non-volatile memory (NVM). It's a promising technology that bridges the performance gap between DRAM and HDD/SSD (Jackson, Johnson, & Parsons, 2018).

Consequently, all these new and old storage/memory technologies have changed the traditional memory hierarchy by adding new layers to the memory hierarchy. Such technologies will be placed above HDD and lower than DRAM. However, as expected, some SCM technologies can replace DRAM, and some can even replace SRAM. Interestingly, studies are still exploring the potential use of such technologies (Yu & Chen, 2016).

With the massive change in the storage area, generated data volume, and lagging of HDD to keep up with this change in terms of performance and market demands, applications had to explore new storage devices that provide them with their demands at a reasonable cost.

Database management systems (DBMS) are one of these applications that has a huge number of I/O operations. DBMS is used as a data host for most contemporary web applications, online transactions, etc., to store, manage and provide data integrity. Consequently, any improvement in DBMSs performance relatively leads to improve their

relying applications' performance (Ramez Elmasri, 2011).

Storage media and the filesystem (where DBMS store its data? and the type of running filesystem on top of the storage media?), play a hugely significant role in DBMS performance. Thus, choosing the right storage media and filesystem that suits the needs of business/users' is a critical task.

Because of the lack of real SCM technologies hardware, there are several emulators either based on hardware or software to emulate the real SCM device. Intel has introduced a PMEM emulator within Linux kernel 4 and later, additionally suggested using RAMDISK as an emulator (Corporation, 2016).

In this research, an examination has been carried out on how database management systems can benefit from SCM technologies by creating a persistent general-purpose RAMDISK using a kernel module based on RAM named ZRAM. This is in addition to testing the emulators with different types of filesystems and tuning the MYSQL InnoDB storage engine. Further details about the MYSQL InnoDB storage engine will be discussed in the next chapter.

1.2 Problem Statement

The performance of the most data-intensive applications is highly reliant on the performance of the used storage system and other additional factors (Van Renen, Vogel, Leis, Neumann, & Kemper, 2019). Such applications require high transaction rates at low latency times. Although the performance of I/O storage devices has been gradually improved, I/O latency and throughput remain one of the core systems' bottlenecks for disk-based database management systems (DBMS) (Bhimani et al., 2017).

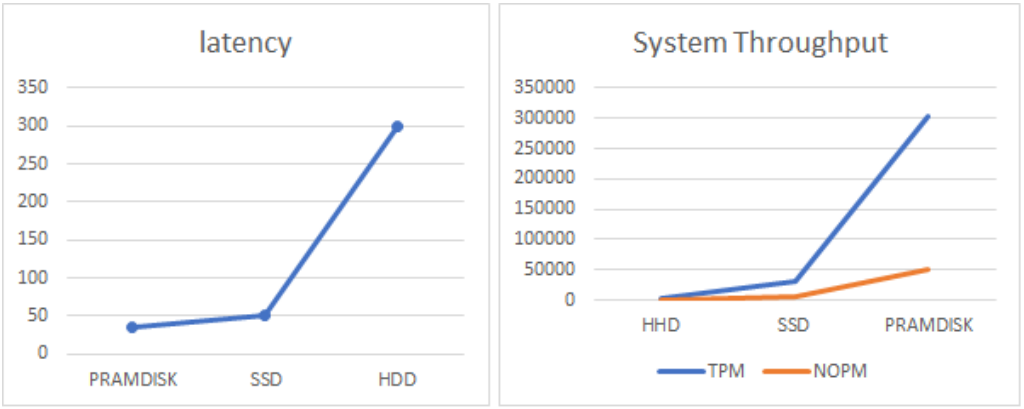

Figure 1.1: Performance of HDD, SSD, and PRAMDISK

From Figure 1.1, it's clear that there is a massive performance variation among hard disk drives, solid-state drives, and RAMDISK with regards to latency and throughput. Where the implemented emulator persistent RAMDISK (PZRAM) achieved superior performance as compared to HDD and SSD with a performance improvement of 14290% and 1167% respectively. This achievement has proved that both HDD and SSD drive performance lag far behind the dynamic random-access memory (DRAM). This huge performance gap is because DRAM latency is around 100,000 times lesser than hard disks (HDD) and the bandwidth of DRAM is 6000 times higher than HDD (Patterson & Hennessy, 2019, p. 856). Although SSD drives have bridged the high-performance gap between memory and storage disks, they are around three orders of magnitude slower than DRAM. Additionally, SSD-based-NAND flash is used as a block device which results in extra overhead (Dulloor, 2016).

This vast performance gap can be hidden and bridged using SCM technologies (Oukid & Lersch, 2019). However, SCM technologies are not broadly available yet. Thus, the Linux RAM-based drives and filesystems available in Linux kernels can be explored to build up a novel PM emulator (this research refers to the SCM emulator as a PM emulator in the remaining chapters) with no special hardware or any complicated code and complexity.

Although ZRAM is available on Linux kernel mainlines and can be used as a general-purpose RAMDISK, its use can only be found as a swap file in the research domain (Desireddy & Pathireddy, 2016; Ilyas, Ahmad, & Saleem, 2020). ZRAM is a compressed RAM-based block device that can save more memory, as compared to other RAMDISK implementation methods. Since ZRAM and RAMDISK are RAM-based, all the contents stored in ZRAM/RAMDISK will be lost upon system shutdown or system crash. Thus, making a ZRAM/RAMDISK persistent is highly necessary to act as an SCM device.

Moreover, apart from the performance being affected by the storage technology, there is an influence of the running filesystem from that storage device on the overall system performance. Thence, the choice as to which filesystem is more suitable for storage and application demand is critical.

For ZRAM, a disk-based filesystem is mostly used on top of it, such as ext4 (StuartIanNaylor, 2019; Gupta, 2014). However, it is argued that since ZRAM is a RAM-based block disk, a RAM-based filesystem will be a better choice than the disk-based one.

In a glance, to bridge the performance gap between dynamic random-access memory and storage device, this research proposes a persistent ZRAM as a PM emulator with TMPFS running on top of the ZRAM block-drive to store the MYSQL data directory.

1.3 Research Questions

In this section, the problem will be broken down into a set of questions for further guidance to design the proposed solution. These questions are identified as follows:

- What are the existing storage/memory technologies, their role, and limitations in database management systems?
- How to enhance system performance using a persistent RAM-based module (ZRAM) as an SCM?
- Is the proposed method PZRAM able to improve the system

performance?

1.4 Research Objectives

The main goal of this research is to improve system performance using SCM. Hence, the target is tailored towards speeding up the response time and increasing system throughput. The objectives of this research are listed thus:

- To explore and analyze the storage/memory technologies, their role, and limitations in database management systems.
- To propose and implement a method that enhances system performance using a persistent RAM-based module (ZRAM) as SCM.
- To empirically evaluate the proposed method (PZRAM) through an experimental analysis to prove its ability to improve the system performance.

1.5 Research Scope

The scope of this research addresses the following points:

- This study only focuses on the system throughput and latency performance.
- The effect of underlying storage on system performance in terms of throughput and latency performance.
- The effect of the running filesystem on system performance like Ext4, XFS, Btrfs, F2FS, RAMFS, TMPFS, and PMFS.
- The moving data files and their effect on performance when moving to another location.
- ZRAM a compressed random-access memory-based block drive is used to build up a PM emulator.
- TPMFS is a random-access memory filesystem, used to create a

RAMDISK to emulate the PM.

- PMEM an Intel emulator is enabled on Linux kernels 4 and later used to emulate PM in this research.

1.6 Research Contribution

This research contributes to MYSQL database management system performance improvement and SCM/NVM DBMS-related research. Since the real hardware of SCM is not yet widely available, the researcher implemented a software emulator to act as an SCM. This emulator is used to perform all experiments. Thus, to implement the software, the research employed the random-access memory-based module presented in mainline Linux kernels. Further, this research proposed a persistent RAM-block device using a ZRAM module with TMPFS on top of it. Additionally, the research delivered an empirical comparative experiment that covered three aspects of performance-related factors as follows:

- What is the best storage device to use as the main storage for data placement?
- What type of file system provides good performance when used on top of the storage device?
- What data files of an application have an impact on performance when it's moved?

The research highlighted the effect of the filesystem, with regards to the moving of data files on latency and throughput performance. Additionally, the study also provides users/researchers a comprehensive guideline to choose which storage type, which data files should move, and which filesystem can provide better performance.

Furthermore, this research provided the implementation of persistent RAMDISK and compared the different methods of its implementations with each other.

1.7 Thesis Organization

The structure of this thesis is organized thus:

Chapter 1-Introduction

The chapter discusses the background, research problem, research questions, objectives, and research contribution.

Chapter 2- literature review

In this chapter, the existing related studies and brief background are reviewed.

Chapter 3- Methodology

This chapter covers the proposed research methodology and its implementation in detail.

Chapter 4- Results and discussion

In this chapter, the obtained results are presented and analyzed to evaluate the proposed solution.

Chapter 5- Conclusion

Finally, this chapter concludes the dissertation and provides future suggestions.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter discusses the review of literature conducted for this research. It commences with a critical review of the thesis case study (MYSQL InnoDB storage engine), followed by discussing the DBMS data placement media, and concludes with a discussion of related works.

2.2 MYSQL

MYSQL is the most used open-source relational database management system (RDBMS) built upon the structured query language (SQL) (Manual, 2013, p. 4). The architecture of MYSQL differs from the other database servers' architecture, thus making it widely used for multi-purpose and demanding environments. Examples of applications/environments that make use of MYSQL include web applications, embedded applications, online transaction processing (OLTP), data warehouses, among others. The most significant merit of MYSQL is its storage-engine architecture. In which the design of its storage separates the query processing and other tasks from data storage and recovery. Different storage engines are offered by MYSQL, such as InnoDB, MyISAM, memory engine, archive engine, CSV engine, NDB cluster engine, etc. (Schwartz, Zaitsev, & Tkachenko, 2012).

2.2.1 InnoDB Storage Engine

InnoDB is the most popular designed storage engine for transactional and non-transactional storage. The vast popularity of Innodb comes as a result of its key provided features such as auto crash recovery, storing data in sequential files called system tablespace, using multi-version concurrency control (MVCC), row-level locking, implementing the four SQL standard isolation levels, producing faster primary key

lookups and foreign key constraints to protect data integrity and finally its support to ACID-transaction features (Schwartz et al., 2012; Manual, 2013, p.).

2.2.2 InnoDB Supported ACID

The ACID-compliant transaction featuring is a group of database concepts that supports data reliability, which is an important aspect of business data and critical data applications. The use of ACID along with the InnoDB storage engine can save stored data from corruption caused by crashes and bugs. These ACID features comprise atomicity, consistency, isolation, and durability (Manual, 2013, p.; MySQL, 2001,p. 242).

2.2.2.1 Atomicity

Atomicity means that every transaction should be accomplished, or nothing happens. Thus, if one transaction fails, then all other transactions will fail. MySQL's InnoDB ensures data atomicity by auto-commit setting, commit statement, rollback statement, and operational data from the information schema tables.

2.2.2.2 Consistency

Consistency is the part of the ACID model that deals with the internal InnoDB processing to save data from crashes and corruption. MYSQL's InnoDB supports data consistency by using features such as InnoDB DWB, and InnoDB crash recovery.

2.2.2.3 Isolation

The InnoDB provides an isolation feature by applying the auto-commit setting and SET ISOLATION LEVEL statement.

2.2.2.4 Durability

Finally, durability is a part of the ACID transaction model which ensures that the transaction that has been committed will be saved permanently. MYSQL InnoDB provides this feature through InnoDB DWB and other configuration options such as:

- Configuration option `innodb_flush_log_at_trx_commit`.
- Configuration option `sync_binlog`.
- Configuration option `innodb_file_per_table`.
- Write buffer in a storage device, such as a disk drive, SSD, or RAID array.

2.2.3 InnoDB Architecture

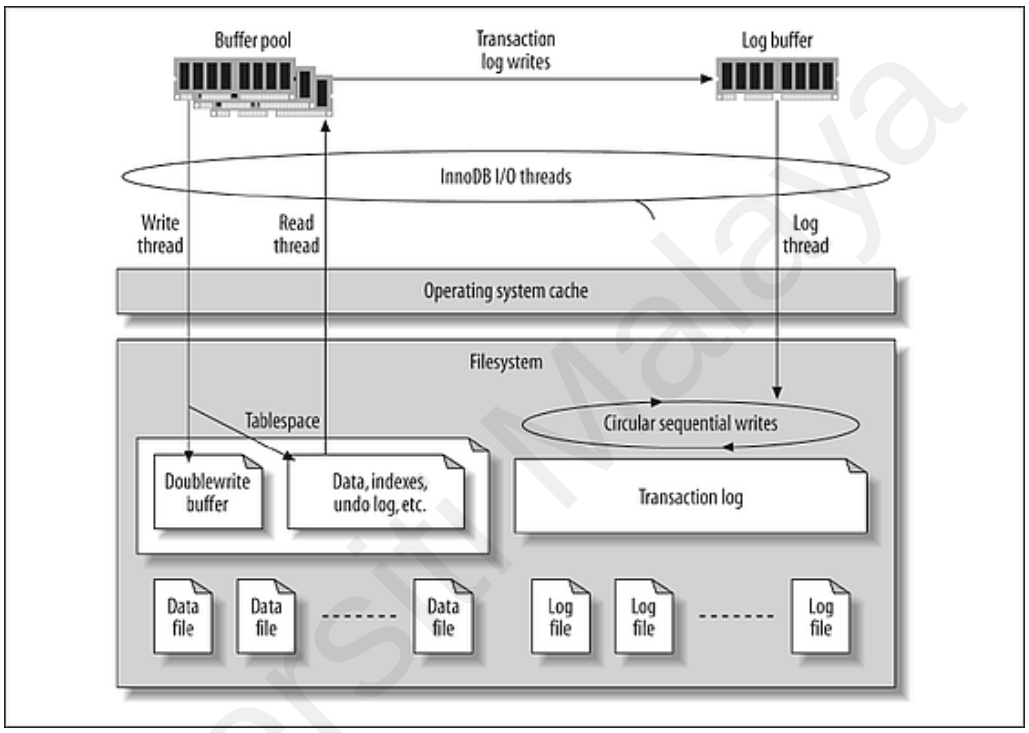

Figure 2.1: InnoDB Buffers and Logs (Schwartz et al., 2012)

As illustrated in Figure 2.1 above, the InnoDB consists of several buffers and logs.

- Buffer pool: a special reserved area in the main memory to cache InnoDB data table and indexes.
- Redo log buffer: a memory area that maintains data that needs to be written into redo logs.
- System tablespace: where the InnoDB stores the users' indexes and data. The created tablespace file, referred to as ibdata, is created in the data directory. The system tablespace includes:

– InnoDB data dictionary

– InnoDB double-write buffer

– Undo logs

– Change buffer

InnoDB distributes these buffers and logs into an in-memory and an on-disk storage area. Figure 2.2 illustrates the distribution.

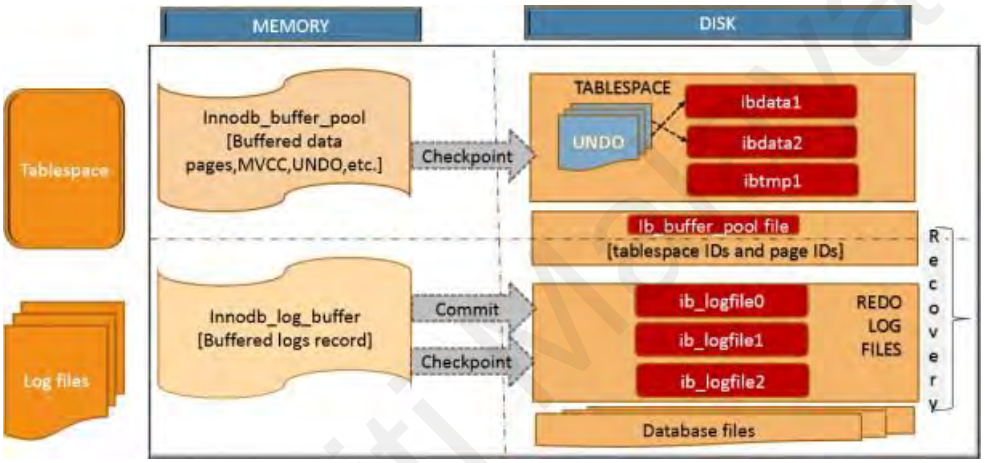

Figure 2.2: InnoDB Components (Lalit, 2016)

2.2.4 InnoDB Buffer Pool

The buffer pool is considered as a cache memory for InnoDB data tables and indexes. It is structured as a linked list of pages and constructed with the least recently used algorithm (LRU). The buffer bool blocks are divided into three blocklists:

- A free list that maintains the free page frames.
- LRU list which includes all pages that contain the file page.
- The flush list comprises all LRU list blocks that have been changed in memory but are not yet written to their data file in the disk.

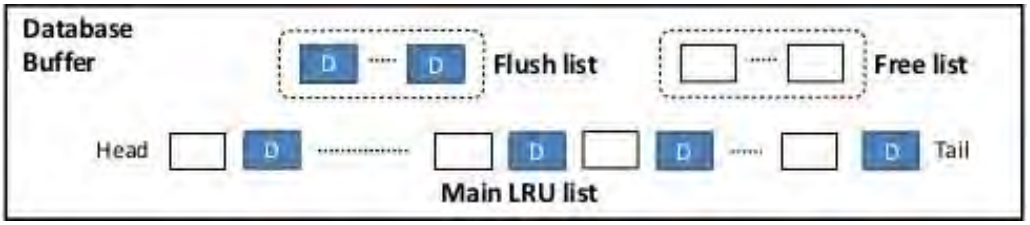

Figure 2.3: Buffer Pool Block Lists (Mijin, 2017)

2.2.4.1 Buffer Pool Mechanism

Since the buffer pool is managed as a list, the newly added pages to the pool are inserted in the middle of the list. The middle point of the list is split up into two sub-lists, the new and old sub list. The head of the list holds the new sub-list where the pages are recently accessed. However, the tail contains the old sub-list where pages are accessed for a short period. Accordingly, these lesser accessed pages are nominees to be evicted by the LRU. 3/8 of the buffer pool is dedicated to the old sub-list. Whenever a new read occurs, the new pages are written at the head of the old sublist and the less accessed ones are moved to the tail, and then evicted. Any reads occurring, inclusive of the old sublist, are moved up to the new sublist and will reside in the buffer pool for a longer time (Manual, 2010).

2.2.4.2 Multiple Buffer Pool Instances

To lower the contention among threads that concurrently read and write the cache pages, InnoDB enables users to split the buffer pool into multiple instances. Every page that is stored in or read from the buffer pool is randomly allocated to only one of the buffer instances, using the hashing function. The hashing function enables every buffer pool to control its free lists, flush lists, LRUs, and all other data structures linked to the buffer pool and ensures they are maintained by their buffer pool mutex. For enabling multiple buffer pool instances, there is a need to set the innodb_buffer_pool_instances configuration option to a value greater than 1 (Manual, 2010).

2.2.5 InnoDB Double-write buffer (DWB)

The DWB is a recovery technique used by InnoDB against partial-written pages issues during the process of writing and updating the pages. The DWB consist of two blocks, where each block has 64 pages. The DWB size is computed by 2 extents (2*64=128). Figure 2.4 below clarifies the DWB architecture:

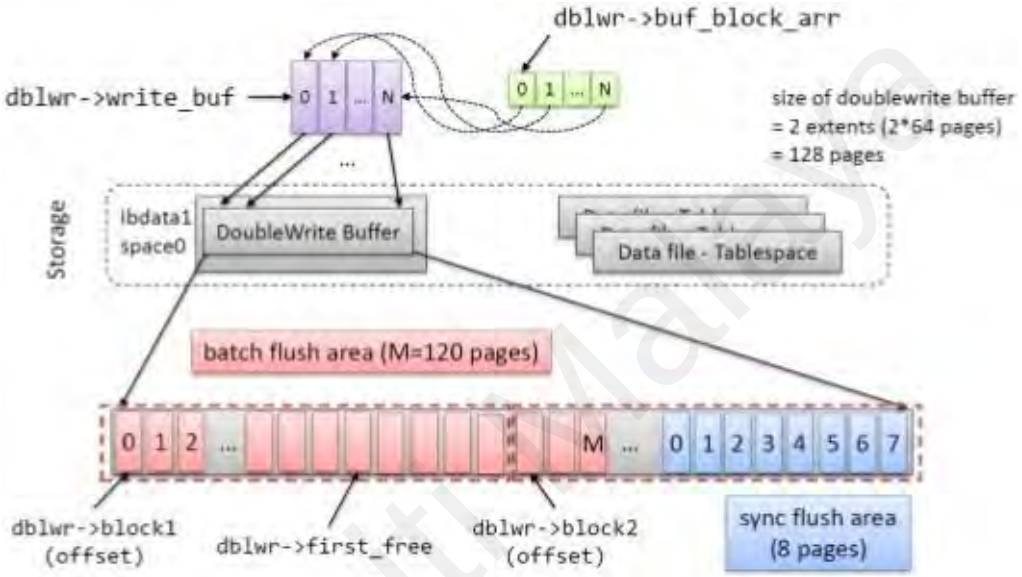

Figure 2.4: Double Write Buffer Architecture (Mijin, 2017)

InnoDB DWB is a redundant method that writes pages twice. This mechanism includes two steps: In the first phase, the InnoDB copies the dirty pages from the buffer pool instances to the DWB file. Then this contiguous set of memory pages is written synchronously and sequentially to the DWB file. In the second phase, InnoDB rewrites the same dirty pages to their proper locations in the tablespace file using the synchronous random writes. The double-write strategy ensures that a complete set of copy pages will always exist in the storage even in the case of crashes and system failures. For example, when a system crash occurs, InnoDB checks double-write files; if a page in the double-write (step I) is found to be partially written, it will be simply removed. If the page in the tablespace is inconsistent, it will be recovered by the copy of the page existing in the double-write file

(Son, Kang, Yeom,& Han, 2017; Manual, 2010; Schwartz et al., 2012).

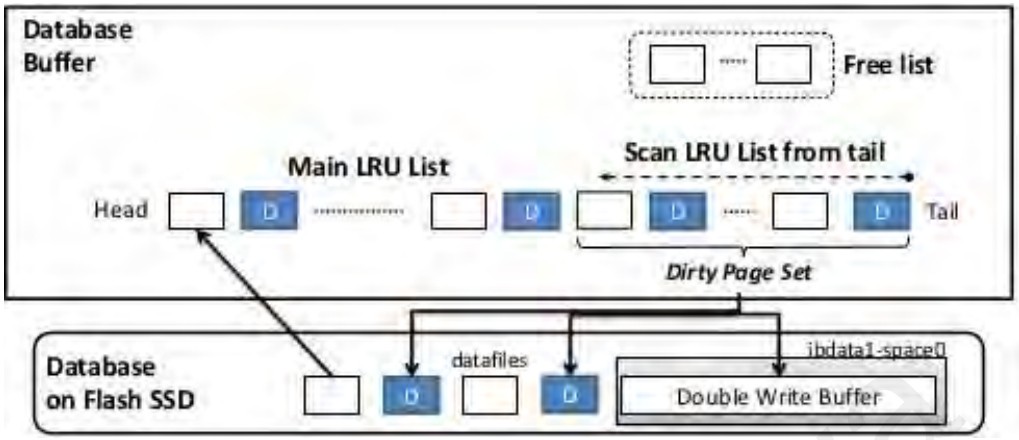

Figure 2.5: Double Write Buffer Phases (Mijin, 2017)

Besides MYSQL, there are other contributed works to the InnoDB DWB, targeted at optimizing it as well as addressing its issues. Percona server 5.7 presented a parallel DWB to address the IO contention issue and concurrency-related issues inherent in the legacy DWB due to its nature of being only one shared file (Schwartz et al., 2008, p. 293).

2.3 Data Placement's Media

Most of the contemporary RDBMS/applications are reliant on two different memory types (Schwartz et al., 2008). Typically, they make use of fast but volatile memory such as random-access memory (RAM) for storing the most frequently accessed data (hot data), whereas persistent and high dense devices are responsible for data storage like hard disk drive (HDD) or solid-state drive (SSD) (Petrov, Gottstein, & Hardock, 2015). Both memory types' pitfalls have led researchers and memory/storage vendors to investigate a new memory type that comprises the advantages of persistent, high bandwidth devices as well as fast, byte-addressable ones.

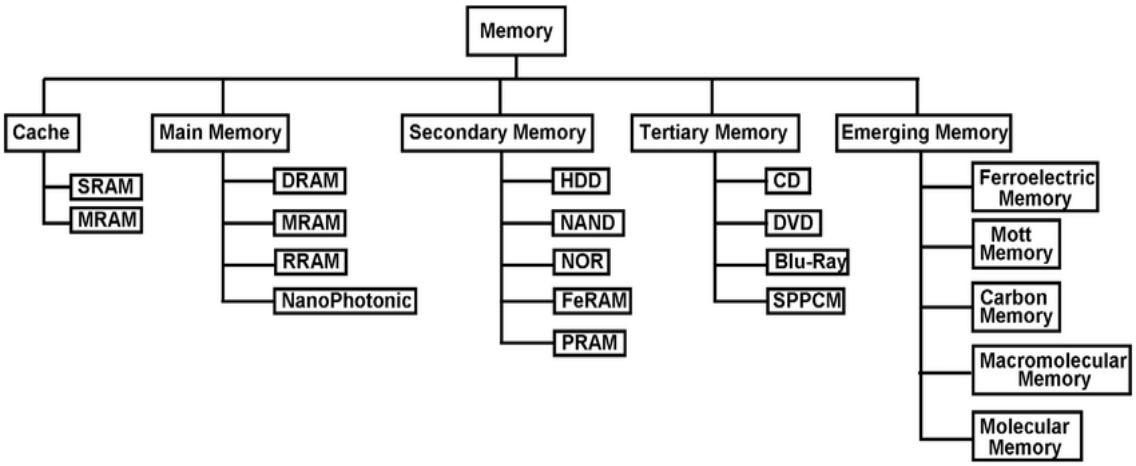

Figure 2.6: Memory Taxonomy (Meena, Sze, Chand, & Tseng, 2014)

2.3.1 Random Access Memory

Random-access memory (RAM) is a system volatile memory technology that stores machine instructions and temporal applications' data. It is shaped of integrated circuit (IC) chips with metal-oxide-semiconductor (MOS) memory cells. There are two commonly used types of RAM, which are static random access memory (SRAM) and dynamic random access memory (DRAM) (Ma et al., 2016; Oukid & Lersch, 2018).

2.3.1.1 Static Random-Access Memory

SRAM technology uses six transistors per bit to read and write data into memory cells. Additionally, it requires a very minimum amount of power to maintain the charge in the reserved mode, and it doesn't need to be refreshed. Therefore, its access time is very close to its cycle time, and it's considered as a very fast memory system, lower dense, and the most expensive one. SRAM is typically used as a cache memory for the CPU in all modern computer systems (Patterson & Hennessy, 2019, p. 85; Oukid & Lersch, 2018).

2.3.1.2 Dynamic Random-Access Memory

DRAM requires only one transistor to store a bit of data, and one capacitor to hold the charges. It requires a periodic refreshment and continuous charge to keep functioning. DRAM is less expensive and higher dense than SRAM, thus, it is used as the main

memory of computer memory systems. The role of DRAM in computer systems has given it higher attention from researchers and memory industry vendors (Oukid & Lersch, 2019). There are a variety of DRAM types in the market introduced by several companies. Currently, most of the DRAM market is controlled by Micron, Samsung, and SK Hynix; they possess more than 95% of the market share. The huge competition among such companies yields in producing numerous amount of DRAM technologies according to the specific application needs (Ma et al., 2016). Due to the reduction in the price of DRAM and its increased capacity, it is used as the main memory for some kind of database management systems (Kabakus & Kara, 2017). Following table 2.1 presents the role and limitations of DRAM:

Table 2.1: DRAM Role & Limitations

Role of DRAM	Systems	Most Popular Type	Limitation
Used as a buffer pool storage	DBMS	MYSQL, Microsoft SQL, Oracle, POSTGRESQL, etc.	Limited data capacity. Energy consumption. Scaleup issues.
Used as the main storage for data placement	In-memory database (IMDB)	Voltdb, SAP Hana, SQL Hekaton, Timesten, H-store, MemSQL etc.	
	Key-value stores (KVS)	Memcached, Redis, Aerospike, etc.	

Figure 2.7 presents the strengths, weaknesses, opportunities, and threats (SWOT) analysis of DRAM:

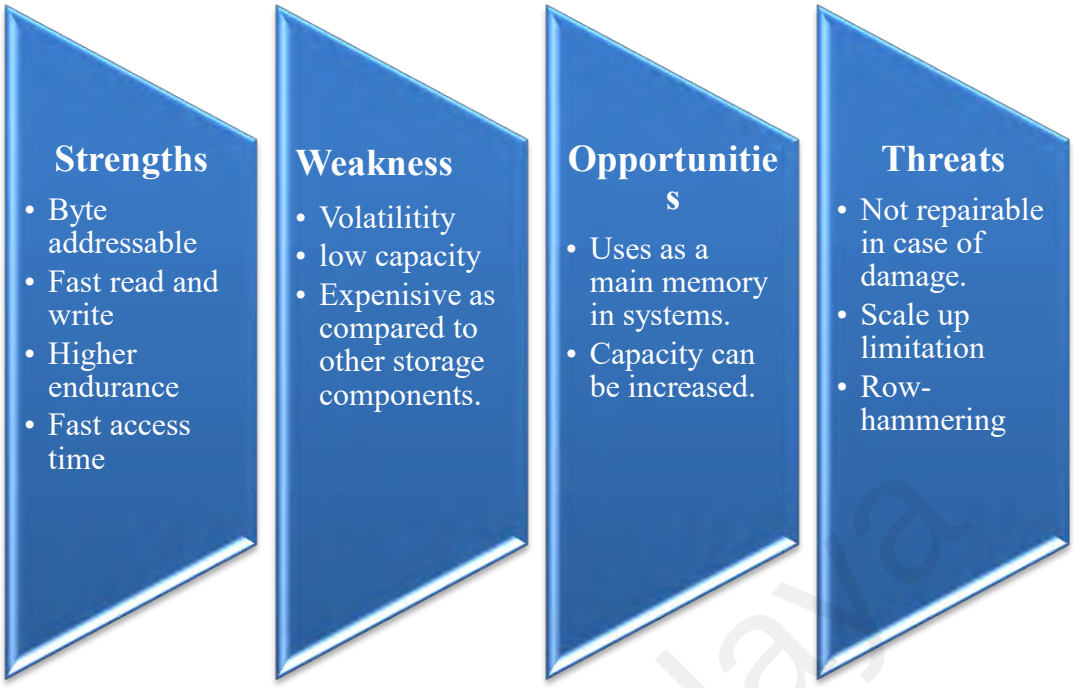

Figure 2.7: SWOT Analysis of RAM

2.3.2 Hard Disk Drive (HDD)

Hard disk drives (HDDs) are non-volatile memory that is commonly used as secondary storage devices in most current general-purpose computer systems. They make use of the magnetization concepts to store and retrieve data, hence they are called magnetic disks. Current HDDs are reliant on the same system design principle since their invention in the 1960s, of which they consist of several platters attached with a head surface, and a movable arm to read/write data to the platter surface (Patterson & Hennessy, 2019, p. M-85)

Although HDDs have been utilized as default storage for most systems and database management systems (DBMSs) due to their high density, lower prices, and high reliability, their nature of magnetic movable parts makes them slower than all mainstream NVM technologies such as SSD and class storage memory technologies (SCM).

Data on HDDs can be accessed randomly and sequentially. However, accessing reads and writes sequentially is far faster than accessing them randomly, because sequential

access does not require repositioning the movable arms. To improve the performance of hard disks, a Redundant Array of Independent Disks (RAID) was introduced to bridge the performance gap between disks and memory and the central processing unit (CPU) (Ma et al., 2016; Oukid & Lersch, 2019). Following table 2.2 presents the role and limitations of the hard disk drive (HDD) for the most popular database management systems (DBMSs).

Table 2.2: HDD Role & Limitations for DBMS

Role of HDD	Systems	Most Popular Type	Limitation
Used as the main storage for Data placement.	SQL-DBMS	MYSQL, Microsoft SQL, Oracle, POSTGRESQL, etc.	Slower data rate. Longer access time. Can become a bottleneck for IO data intensiveness.
Used either as a Backup location or as the main storage.	NOSQL	MongoDB, Redis, Memcached, CouchDB, Couchbase, Aerospike, etc.	

Figure 2.8 presents the strengths, weaknesses, opportunities, and threats (SWOT) analysis of the hard disk drives (HDD):

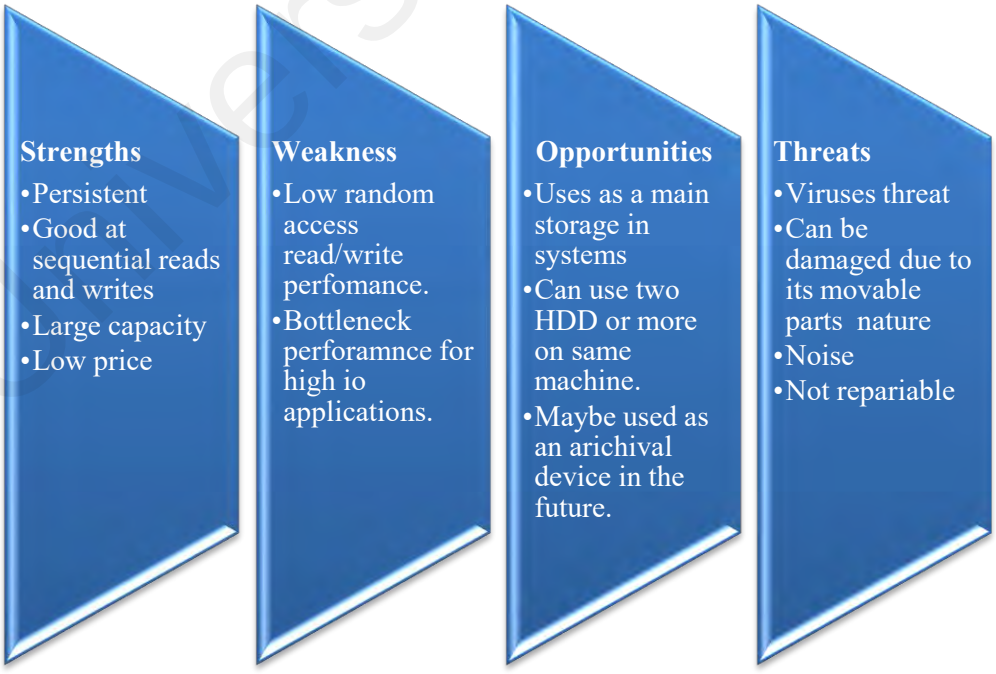

Figure 2.8: SWOT Analysis of HDD

2.3.3 Solid State Drive (SSD)

SSDs are electronic storage devices based on flash NAND/NOR or Random-access memory. SSDs vary from HDDs in their design, where they do not have any movable parts and stores data in semiconductor cells. SSDs are designed in such a way that they vary from each other according to the number of bits that can be stored in each semiconductor cell (Ma et al., 2016; Oukid & Lersch, 2019). However, most contemporary developed SSDs are functional either using a multi-level cell (MLC) or triple-level cell (TLC).

To scale more capacity, a 3D NAND Flash technology was introduced and exploited to manufacture SSDs. The 3D NAND technologies have been widely used in most present-day SSDs (Bhimani et al., 2017; Canim, Mihaila, Bhattacharjee, Ross, & Lang, 2010).

SSDs were the first storage technology leap that was placed between HDD and DRAM. Its access time is measured in microseconds which is far lower than HDD access time. Because of the promising performance of SSDs, a lot of studies investigated on making use of it with database management systems (Do et al., 2011; Kang, Lee, Moon, Kee, & Oh, 2014; Rizvi & Chung, 2010; Shahla Rizvi & Chung, 2010). Following table 2.3 presents the roles and limitations of the solid-state drive (SSD)for DBMS:

Table 2.3: SSD Role & Limitations

Role of SSD	Reference	Limitation
SSD-DBMS	(Shahla Rizvi & Chung, 2010), (Rizvi & Chung, 2010), (Schmidt, Ou, & Härder, 2009)	Lifespan & Reliability. Lower access time than DRAM. Not optimized for DBMS, it's simply replaced by HDD.
As a Buffer-pool extension	(Do et al., 2011), (Canim et al., 2010), (Zhuang, Zuk, Ramachandra, & Sridharan, 2016),	
Cache & Write optimization	(Kang et al., 2012), (He, Sun, & Feng, 2014), etc.	
Note*	SSD is simply used as main storage instead of HDD by different users, companies, etc. for their systems like DBMS or NoSQL-related apps to get better performance.	

Figure 2.9 shows the strengths, weaknesses, opportunities, and threats (SWOT) analysis of the solid-state drives (SSD):

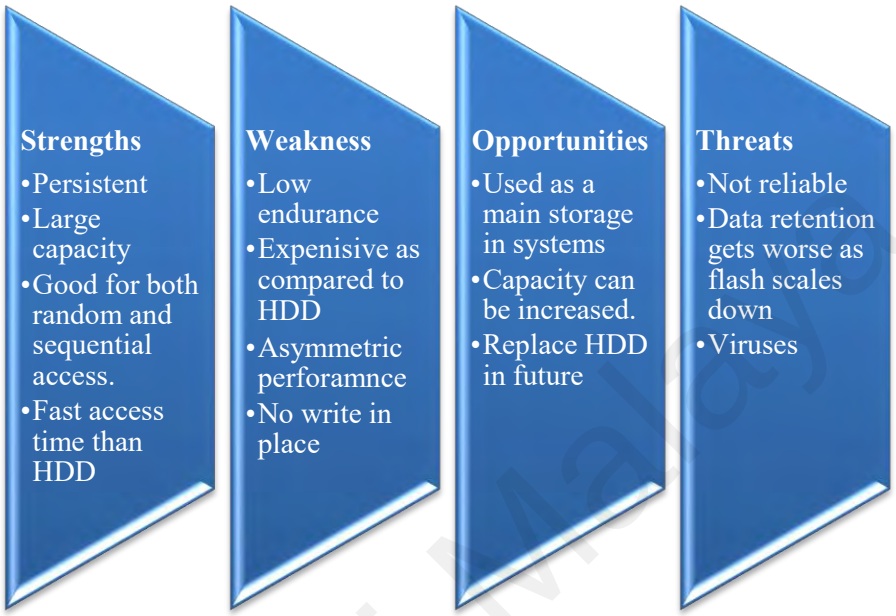

Figure 2.9: SWOT Analysis of SSD

2.3.4 Storage Class Memories (SCM)

SCM is a non-volatile memory (NVM) technology that provides non-volatility, high density as HDD/SSD, byte-addressable, lower latency, and higher throughput as DRAMs. Moreover, SCM can be directly connected to the CPU using the DRAM bus and addressed by usual load/store instructions. There are a variety of SCM technologies introduced by different companies like Micron, Intel, SK Hynix, SanDisk, IBM, Samsung, Crossbar, etc. some of these technologies are discussed in the next subsections (Oukid et al., 2017).

2.3.4.1 Non-Volatile Dual In-line Memory Module (NVDIMM)

NVDIMM is a non-volatile random access memory type that uses the dual in-line memory module package (DIMM). NVDIMM provides data persistence even in cases whereby there is a system failure as well as sudden system shutdowns. NVDIMM uses a

backup battery (BB) to supply power to volatile DRAMs for up to 72 hours. Due to the downside of this battery backup technology, most newly developed NVDIMMs use the onboard supercapacitors to save DRAMs' energy. This is because it is a non-volatile memory type and is well-matched with DRAM's interface; thus, it is also referred to as persistent memory (PMEM) (Bez & Pirovano, 2004; Nguyen & Lee, 2019). There are three types of NVDIMM introduced by the standard JEDEC organization and they are discussed next.

- **NVDIMM-N**

NVDIMM-N consists of both flash memory technology and legacy DRAM in a single module, in which the DRAM can be accessed directly by the computer systems. In case of any system failure or power loss, the data will be backed up from DRAM onto a non-volatile flash memory by the NVDIMM-N module and copied back into DRAM when the system power is restored. There are a variety of products based on the NVDIMM-N technology available in the market (Sainio, 2016; Gervasi, 2017).

- **NVDIMM-F**

NVDIMM-F is a kind of flash memory technology with the use of DRAM's bus. Although the use of DRAM's bus enhanced the NVDIMM-F performance over the SSD in terms of bandwidth and latency, it functions like an SSD (Sainio, 2016; Gervasi, 2017).

- **NVDIMM-P**

NVDIMM-P contains both features of DRAM and flash technologies, in which it can provide both block addressing and byte addressing. With regards to the size, NVDIMM-P can be manufactured with terabytes like any NAND flash technology. Furthermore, it can provide time delay within the level of 7-10 seconds. Like the previous types of NVDIMM, DRAM can be accessed directly using the memory bus, hence the data can be

accessed directly by the CPU without the need for extra latency of the PCIe or any other disk interfaces (Sainio, 2016; Gervasi, 2017).

2.3.4.2 Phase-Change Memory (PCM)

PCM is another form of non-volatile random access memory that is also referred to as a phase change RAM (PRAM/PCRAM), an ovonic unified memory (OUM), or a chalcogenide RAM (CRAM) (B. C. Lee, Ipek, Mutlu, & Burger, 2009). This memory technology is built upon a chalcogenide glass material that can be switched between two different states called amorphous and crystalline. The PCM cell structure is comprised of 1 transistor and one resistor (1T/1R) and the data is stored using resistivity instead of the electrical chargers (Lee, 2009; Oukid & Lersch, 2019). Intel and STMicroelectronics are now selling PRAM-based devices to consumers, under the names: 3D XPoint Optane, and QuantX.

2.3.4.3 3D Xpoint

This refers to a form of non-volatile memory (NVM) technologies introduced by Intel and Micron Manufacturing. Since April 2017, they have been available on the memory market under the names of Intel Optane and Micron QuantX. Their initial prices are lesser than the prices of dynamic random-access memory (DRAM) and more than the prices of flash-based memories (Jackson et al., 2018; Smith, 2015).

2.3.4.4 Resistive Random-Access Memory (RRAM/ReRAM)

ReRAM is another form of non-volatile random-access memory technology, similar to a phase-change memory (PCM). ReRAM functions by switching the resistance through a dielectric solid-state material usually referred to as a memristor (Calderoni, Sills, Cardon, Faraoni, & Ramaswamy, 2015; S. Kim, 2012).

2.3.4.5 Magneto- Resistive RAM (MRAM)

MRAM is also another form of non-volatile memory where data are stored by magnetic storage elements (Meena et al., 2014). These storage elements are developed by two ferromagnetic plates, each of which can hold a magnetic field, spilled by a thin layer. One of the two plates is a permanent magnet set to a polarity; the other's field can be changed to match that of an external field to store memory. STT-MRAM is a form of MRAM with better scalability and far higher densities over traditional MRAM generations. The following table presents the classifications of the research papers on storage class memory-database management systems (SCM-DBMS):

Table 2.4: Classifications of Research Papers on SCM-DBMS

Area of study	Reference
Design & architecture	(Arulraj & Pavlo, 2017a), (Van Renen et al., 2018a), (DeBrabant et al., 2014a), (Mustafa, Armejach, Ozturk, Cristal, & Unsal, 2017a), (Oukid, 2019), (Kuznetsov, 2019), etc.
Recovery & logs	(Huang, Schwan, & Qureshi, 2014a), (W. H. Kim, Kim, Baek, Nam, & Won, 2016), (Arulraj, Pavlo, & Dulloor, 2015), (Li, Liu, Xiao, Zeng, & Zhu, 2018), (Son, Kang, Yeom, & Han, 2017a), (Li et al., 2018), (Joshi, Nagarajan, Viglas, & Cintra, 2017), (Arulraj, Perron, & Pavlo, 2016), (Wang & Johnson, 2014), etc
Indexes & data structure	(Sha et al., 2018), (Oukid, Lasperas, Nica, Willhalm, & Lehner, 2016), (Lersch, Hao, Oukid, Wang, & Willhalm, 2019), (March & Clara, 2017), (Proctor, 2012), (Arulraj, Levandoski, Minhas, & Larson, 2018), (Lersch et al., 2019), etc
Storage engines & transactions	(Oukid, Booss, Lehner, Bumbulis, & Willhalm, 2014), (Eisenman et al., 2018), (Kimura, 2015), (Kolli, Pelley, Saidi, Chen, & Wenisch, 2016), (Liu et al., 2017), (Giles, Doshi, & Varman, 2013), (Sorin, 2017), etc.
Buffer & data management	(Leis, Haubenschild, Kemper, & Neumann, 2018),(Nguyen & Lee, 2019), (H. Kim, Agrawal, & Ungureanu, 2012), (Götze, van Renen, Lersch, Leis, & Oukid, 2018), (Arulraj, Pavlo, & Malladi, 2019), (Nguyen & Lee, 2019)etc.

Figure 2.10 shows the strengths, weaknesses, opportunities, and threats (SWOT) of the storage class memory devices (SCM):

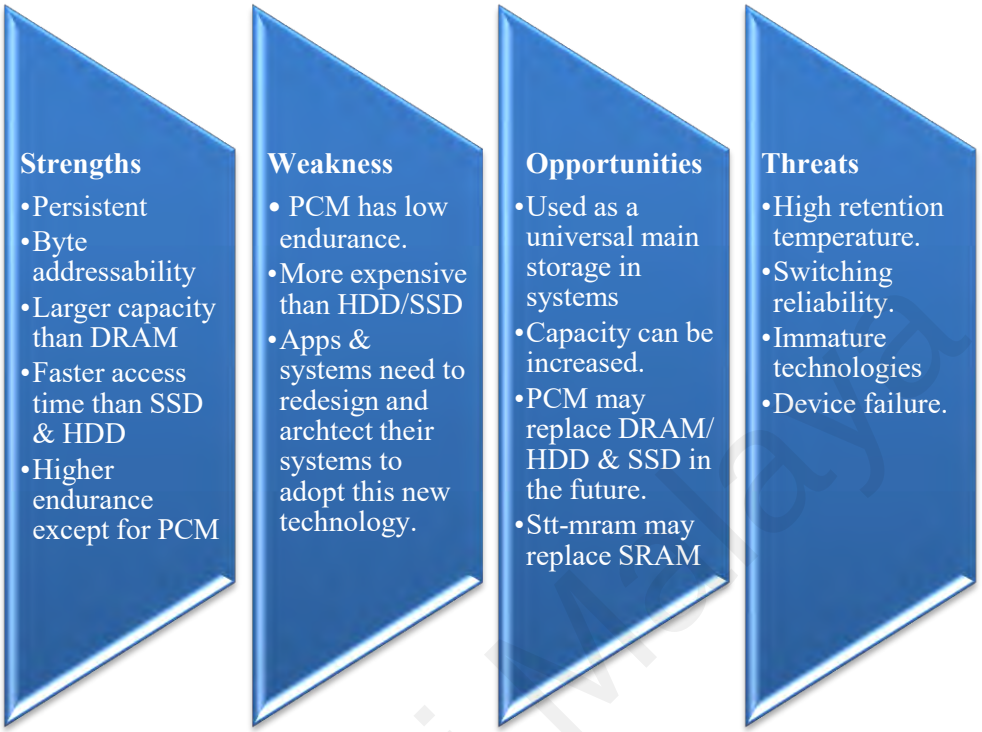

Figure 2.10: SWOT of SCM

2.4 Related Work

2.4.1 Main Memory Database System (MMDB)

Although MMDB was a subject of study since the mid-1980s, the rapid drop in the DRAMs' price and the increase of their capacity has again made MMDB a hot topic in recent years. Nowadays, most database vendors have an in-memory database solution to enhance their systems' performance. There is a vast number of MMDBs products that have dominated the DBMS enterprise and data-centers, varying with regards to them being either relational or NoSQL (Faerber et al., 2017, 2017).

1. **In-Memory Database (IMDB)**

One of the relational in-memory database examples is Oracle times-ten. Times-ten

(Lahiri, Neimat, & Folkman, 2013) stores all its related data into the memory at the runtime. In addition, it provides data durability by using an HDD to provide data persistence and recovery. HANA (Faerber et al., 2017), is another in-memory, column-based, relational DBMS that developed SAP SE. It is designed to handle real-time analytics and transactional processing. It differs from other MMDB by providing multi-level partitioning.

MEMSQL (Chen et al., 2015) is another type of relational database that carries out both transactions and real-time analytics. MEMSQL makes use of standard SQL to perform queries. Furthermore, it is compatible with MYSQL, where the application can connect to MEMSQL through MYSQL clients and servers. In like manner, Hekaton (Diaconu et al., 2013) is a Microsoft SQL integrated in-memory database. The main goal of Hekaton is to provide online transaction processing (OLTP). Furthermore, Voltdb is a relational in-memory database system based on ACID-compliant.

2. Key-Value Stores (KVS):

Memcached (Jose et al., 2011) is an example of open-source NoSQL key-value memory caching systems, which are widely used to store data and objects by the data centers, web applications, etc. Memcached uses large hash tables distributed among multiple machines/servers alongside the least recently used (LRU) mechanism, to evict data. It provides a simple data type and command. Query and data access are done in a multi-thread manner.

Redis (Anthony, 2016) is another open-source NoSQL key value in the memory database store. Redis can be used as a database or a cache system. When it is used as a cache, it provides six types of cache evection policies. Moreover, Redis supports a set of complex data types like maps, sets, strings, stored sets, lists, streams, indexes, etc. It is

considered one of the most common top KV stores globally. Aerospike is an open-source NoSQL flash-optimized in-memory.

Aerospike (Anthony, 2016) architecture consists of three layers, a client Layer that tracks node and figures out where the data is placed in the cluster; Data Distribution Layer that manages cluster communications and handles failover, replication, synchronization, rebalancing, and data migration; Data Storage Layer that stores data in memory and flash memory for fast retrieval. Although Data Storage Layer is optimized for flash memory (SSDs), it can also be configured to store data in memory (RAM).

2.4.2 Storage Class Memory Database Management Systems (SCM/NVM-DBMS)

There are several studies on the use of storage class memories/non-volatile memories for database management systems, which focus on either studying recovery methods, improving database logging, or designing a new DBMS architecture.

In a recent research work (Mustafa, Armejach, Ozturk, Cristal, & Unsal, 2017), the authors explored the implications of employing NVM as primary storage for DBMS by investigating the necessary modifications to be applied on a traditional relational DBMS to take advantage of NVM features.

This research (Van Renen et al., 2018) proposed a transparent integration of NVM into the memory hierarchy. While some systems use NVM mostly to achieve durability or to extend the main memory capacity, in this approach, NVM is an integral part: they leveraged not only the persistency but also the byte addressability by loading individual cache lines from NVM into DRAM. This way, variable-size pages can be deployed, which allows hot tuples to be kept in DRAM instead of hot pages.

Furthermore, the paper of (Lindström, Das, Mathiasen, Arteaga, & Talagala, 2015), implemented NVM Compression in the popular MariaDB database and used variants of

commonly available POSIX file system interfaces to provide the extended FTL capabilities to the user space application. The experimental results show that the hybrid approach of NVM Compression can improve compression performance by 2-7x, deliver compression performance for flash devices within 5% of uncompressed performance, improve storage efficiency by 19% over legacy Row-Compression, reduce data write up to 4x when combined with other flash aware techniques such as Atomic Writes, and deliver further advantages in power efficiency and CPU utilization.

In the work of (Lemke et al., 2017), the SAP HANA in-memory database system integrated NVM by utilizing its "delta" and "main" storage separation. The immutable and compressed bulk of the data ("main") was stored on NVM, while the updatable part ("delta"), which contains recent changes, remained in the main memory. This simple approach nicely fits HANA's architecture but does not apply to most database systems. Another specific way of exploiting NVM is to use it as a cache for LSM-based storage.

Furthermore, in the research conducted by (Fang, Hsiao, He, Mohan, & Wang, 2011), a detailed design of an SCM-based approach for DBMS logging was proposed, in which this approach achieved high performance by simplified system design and better concurrency support.

Another work conducted by (Arulraj & Pavlo, 2017a) provided an outline on how to build a new DBMS given the changes to the hardware landscape due to NVM. The authors surveyed recent developments in this area and discussed the lessons learned from prior research on designing NVM database systems. They further highlighted a set of open research problems and presented ideas for solving some of them.

Consequently in the study of (DeBrabant et al., 2014a), two possible architectures using non-volatile memory (i.e., NVM-only and NVM+DRAM architectures) were studied and evaluated. Their analysis shows that memory-oriented systems are better

suited to take advantage of NVM and outperform their disk-oriented counterparts. However, it was discovered that from both the NVM-only and NVM & DRAM architectures, the throughput of the memory-oriented systems decreases as workload skew is decreased, while the throughput of the disk-oriented architectures increases as workload skew is decreased.

More so, in another study by (DeBrabant et al., 2014b), a system software support to enable low-overhead PM access by new and legacy applications were explored. The authors implemented PMFS, a lightweight POSIX file system that exploits PM's byte-addressability to avoid overheads of block-oriented storage and enable direct PM access by applications (with memory-mapped I/O).

Similarly, to the above-mentioned works, this study also uses the non-volatile memory concepts to enhance and achieve better system performance. However, we differ from these works in terms of the used emulator and some other objectives. Further explanation will be carried in subsections 2.5.2 and 2.5.3.

2.4.3 Solid-state Drives Database Management Systems (SSD-DBMS)

The research community has taken note of this trend, and over the past year, there has been substantial research on redesigning various DBMS components for SSDs (Canim et al., 2010; He et al., 2014; Kang et al., 2014, 2012; Min, Kang, & Kim, 2015).

In a research by (Do et al., 2011), four alternative designs that use an SSD to extend SQL Server 2008 R2's buffer pool across a broad range of benchmarks were evaluated. In like manner, (Canim et al., 2010) presented an SSD-resident buffer-pool extension for database systems. The regular memory-resident buffer-pool functioned as the primary cache, whereas the SSD portion of the buffer-pool was used as a second-level cache.

Furthermore, in the work of Kang et al. (2015), SSD was used as the storage device for the DWB. This work addressed the DWB from a different aspect in which the storage devices are used to store the DWB per its size. DWB is used as a read cache for random reads, in addition, to supportingatomic writes. The proposed mechanism offered a 50% performance improvement compared to hard disk storage (HDD).

Another research about the use of SSD for DBMS was that of Rizvi & Chung (2010), where DBMS was implemented on flash memory SSD-based large enterprise applications. The authors presented the relevancy of SSD characteristics with storage features of data warehouses and data marts and proposed the architecture of data storage for variable-length records in and data retrieval, using virtual sequential access method by multilevel indexing from flash memory-based SSDs for such applications.

Moreover, in the study by Shahla Rizvi & Chung (2010), an advanced DBMS architecture was proposed using key sequenced data set, virtual sequential access method, and multilevel indexing for flash memory SSD based performance oriented embedded and multimedia applications. Basic database operations plus space reclamation and memory wear-leveling were achieved by taking flash characteristics into account carefully. Main memory-based buffer management was implemented to increase throughput and to ensure efficient media utilization.

2.4.4 USE of RAMDISK and ZRAM for Better Performance

RAMDISKS and ZRAM are dynamic random-access memory-based block drives that are provided within Linux kernels. Both modules can be used to enhance system performance.

2.4.4.1 RAMDISK

The major use of RAMDISK modules is their application as a virtual file system for Linux kernel; in which the Linux kernel utilizes more than one RAMDISK file system to

mount a kernel image in its root file system during the system boot time. Furthermore, Linux exploits space to keep system and hardware devices' information in a proc file system or sysfs of the RAM disk at the run time. However, the RAM-based file systems provided by Linux do not have durability, which means that if the systems shut down or crash, the data stored on the RAMDISK file systems would be lost. Thus, to ensure durability, moving stored data from the RAMDISK drive to the hard disk drive should be done synchronously.

Also, RAMDISK can be used to store temporary data of applications to gain better performance. Numerous studies have used RAMDISK to improve applications performance (H. Lee & Lee, 2016; Shu, Yu, & Yan, 2004; Wickberg & Carothers, 2012). This performance improvement comes with the use of DRAM capacity cost. Further, RAMDISKs can be used to emulate persistent memory or non-volatile memory (Bahn & Cho, 2020; Dulloor, 2016; Sehgal, Basu, Srinivasan, & Voruganti, 2015).

2.4.4.2 ZRAM

ZRAM is mostly used to optimize swap performance to improve the overall system performance (Desireddy & Pathireddy, 2016; Ilyas et al., 2020). Additionally, it can be used as a general-purpose RAMDISK. In an experiment (StuartIanNaylor, 2019), a ZRAM drive was implemented to enhance IoT projects by reducing write operations and memory footprints. However, this implemented ZRAM drive was performed using an overlay to ensure persistency and Ext4 as a filesystem running on top of the created ZRAM. The authors argued that since ZRAM is based on RAM, a RAM-based filesystem would be more suitable for ZRAM.

2.5 Discussion and Comparison

This section discusses and compares the difference between data placement media and the most related works.

2.5.1 Data Placement Media

Storage and memory markets provide us with a collection of technologies with different storage mechanisms and various performance characteristics. These differences among them give designers the ability to choose which storage or memory is needed for the system performance requirements. None of the aspects is best, thus whenever one is chosen above others, there's a need to clarify the accompanying demands. For performance improvement and access time, memories like SRAM, DRAM, PCM, STT_MRAM, and RERAM is the best choice. With regards to price and capacity, SSD and HDD are preferable. Nonetheless, the choices are not limited to the aspects alone, others exist such as reliability, lifetime, etc. Table 2.5 offers a comparison among these technologies with some characters.

Table 2.5: Comparison of Memory Technologies (Oukid & Lersch, 2019; Rizk, Rizk, Kumar, & Bayoumi, 2019; Zhao, Xu, Chi, & Xie, 2015)

Storage class	HDD	SSD	DRAM	SRAM	PCM	ReRAM	STT-RAM
Volatility	No	No	Yes	Yes	No	No	No
Endurance	$> 10^{i5}$	10^4	$> 10^{i5}$	$> 10^{i5}$	10^5-10^9	10^5-10^{ii}	$> 10^{i5}$
Scalability	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Mechanism	Magnetic	Flash					Magnetic
Cell element	N/A	1T	1T1C	6T	1T1R	1T1C	1T1R
Cell size	$2/3F^2$	$4-5F^2$	$6-8F^2$	$> 100F^2$	$8-16F^2$	$> 5F^2$	$37F^2$
Read latency	8.5ms	25 μs	10-60ns	<10ns	48ns	<10ns	<10ns
Write latency	9.5ms	200 μs	10-60ns	<10ns	40-150ns	10ns	12.5ns
Energy per bit access	100-100omJ	10nJ	2pJ	1pJ	100pJ	0.02pJ	2pJ
Cost	low	low	High	high	medium	medium	medium

2.5.2 Related Work

As surveyed in the related work section, numerous researchers have studied and investigated the impact of NVM on database design (Fang et al., 2011; Huang, Schwan, & Qureshi, 2014b; W. H. Kim et al., 2016; Son, Kang, Yeom, & Han, 2017b), where some specifically conducted studies on the database logging for NVM. Thus, this reduces the impact of disk I/O on transaction throughput and response times by directly writing

log records into an NVM component instead of flushing them to disk. However, the new experiments show that moving the whole data directory including logs, helped in gaining higher performance than moving only log files; thus, leading to the novelty of the present research as compared to that of past related works. This research also implements a general-purpose RAMDIK using ZRAM for emulation of PM.

The architecture of NVM DBMS is prevalent in literature (Arulraj & Pavlo, 2017a; Kuznetsov, 2019; Petrov et al., 2015) and was studied as storage for DBMS (Mustafa et al., 2017b). However, this present work examined the performance gains of MYSQL by using an NVM as a default storage for its data directory. This research has emulated the NVM using persistent RAMDISK (PZRAM), PMEM, and PRAMDISK. Additionally, the use of NVM has been tested as storage for log files, ibdata, and both ibdata and logs; also, the researchers examined the effect of filesystems on system performance. Further, the present research compared the performance by only moving logs into NVM, ibdata, logs & ibdata with moving the whole data directory. Moreover, no modification was made on the source code of MYSQL, rather it was only tuned to get better performance. Additionally, the emulator used in this research doesn't need any special hardware or any complexity. It's available on Linux kernel mainlines. Table 2.6 presents the most relevant studies to the current work.

Table 2.6: Comparison of Related Work

Author	Emulator	Case study	Focus	Contribution	Limitations
Mustafa, Naveed Ul [2016]	PMFS	PostgreSQL SE	NVM DBMS design	Investigated the impact of the use of NVM as the main storage for DBMS.	Increase cache miss latency where data is not close to the processing units when it is needed for processing
Kuznetsov, Sergey D [2019]	NVM emulator	SOFORT	NVM-DBMS architecture	Surveyed and discussed the existing works on NVM DBMS architecture.	The architecture sketch is not well technically elaborated.
Debrabant, Justin [2014]	NUMA interface & PMFS	H-store, Mysql(v5.5)	NVM DBMS design & architecture	Explored and analyzed the two possible uses of NVM for DBMS.	The reduction in performance is due to the overhead of fetching and evicting data from NVM
Renen, Alexander Van [2018]	SEP	table as a B+-tree using C++ templates	NVM DBMS design	Evaluated the three NVM DBMS design approaches and proposed a lightweight storage manager.	Writes are not immediately persistent because NVM is behind the same CPU cache hierarchy as DRAM and changes are initially written to the volatile CPU cache.
Zhang, Yiying [2015]	PMEP	File Bench, MongoDB, and MySQL, Memcached	NVM and its use as storage of application	Analyzed the performance of applications with the use of NVM.	A small performance drop over DRAM when using NVMM as a big memory for memory-intensive applications.

2.5.2 Identify the research gap:

Because of the rapid and massive amount of today's processed and generated online data, most of the data-intensive and real-time applications require higher performance metrics from their underlying storage media (Dulloor, 2016; Sha et al., 2018; Zhang & Swanson, 2015). However, most of the existing storage technologies are limited in their performance (Oukid & Lersch, 2019). Consequently, a new generation of storage technologies was explored to satisfy these applications' demands. Such technologies are called storage class memories (SCM), which are compromised with the best metrics of both disk/flash drives and DRAMs.

Since SCMs are still not widely available, many research papers offered and implemented PM emulators to experiment with the performance of SCM. However, such emulators either need special hardware or have a complexity (Kuznetsov, 2019; Mustafa et al., 2017b; Van Renen et al., 2018b; Volos, Magalhaes, Cherkasova, & Li, 2015).

Several pieces of research studied how database management systems (DBMS) can leverage the storage class memories (SCM). However, most of these works focused on database logging (Arulraj et al., 2015; Chatzistergiou, Cintra, & Viglas, 2015; Liu et al., 2017; March & Clara, 2017), the SCM-DBMS architectures (Arulraj & Pavlo, 2017b; DeBrabant et al., 2014a; Rizk et al., 2019), memory management(Arulraj et al., 2019; Van Renen et al., 2018b).

Although many research studies have been carried out on the SCM-DBMS, they did not focus on the type of data files to have resided in the SCM device, best filesystem for the underlying device, and exploiting system modules such as ZRAM to build up a persistent PM emulator.

The above shortcomings motivated the researcher to propose a method that uses a kernel module called ZRAM to enhance the system performance. In addition to

examining which type of data file has the highest effect on the system performance when resides within SCM, and the impact of the running filesystem on the top of the emulators on the overall system performance.

2.6 Summary

In summary, the performance of hard disk drives (HDD) lags far behind solid-state drives (SSD) and dynamic random-access memory (DRAM). Although SSD bridged the performance gap between HDD and DRAM, their performance is three orders magnitude lower than DRAM. Most contemporary systems and applications rely on HDD or SSD to store their data. Such storage technologies provide larger capacities with persistence at lower costs. However, due to their high access latency time, volatile main memories (DRAMs) were used as the main storage to store data by many systems. However, DRAM cannot be used further because of its limited capacity and scale issues.

As I/O latency and throughput are one of the major performance bottlenecks for disk-based database systems, new storage technologies must be introduced to bridge this performance gap. Storage class memories (SCM) are such promising technologies that can eliminate I/O bottlenecks and bridge the performance gap between HDD/SSD and DRAM. These technologies are not widely available yet, thus several SCM emulators were proposed by researchers. However, these emulators either require special hardware or have code complexity. Intel PMEM emulator and RAMDISK are available solutions within system resources. However, PMEM requires reconfiguring and recompiling the Linux kernel which takes more storage space. Finally, each RAMDISK implementation method has its advantages and disadvantages.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter starts with describing the research methodology used in this research, followed by discussing the proposed method and its implementation in detail. The methodology of this research consists of the following phases shown in Figure 3.1.

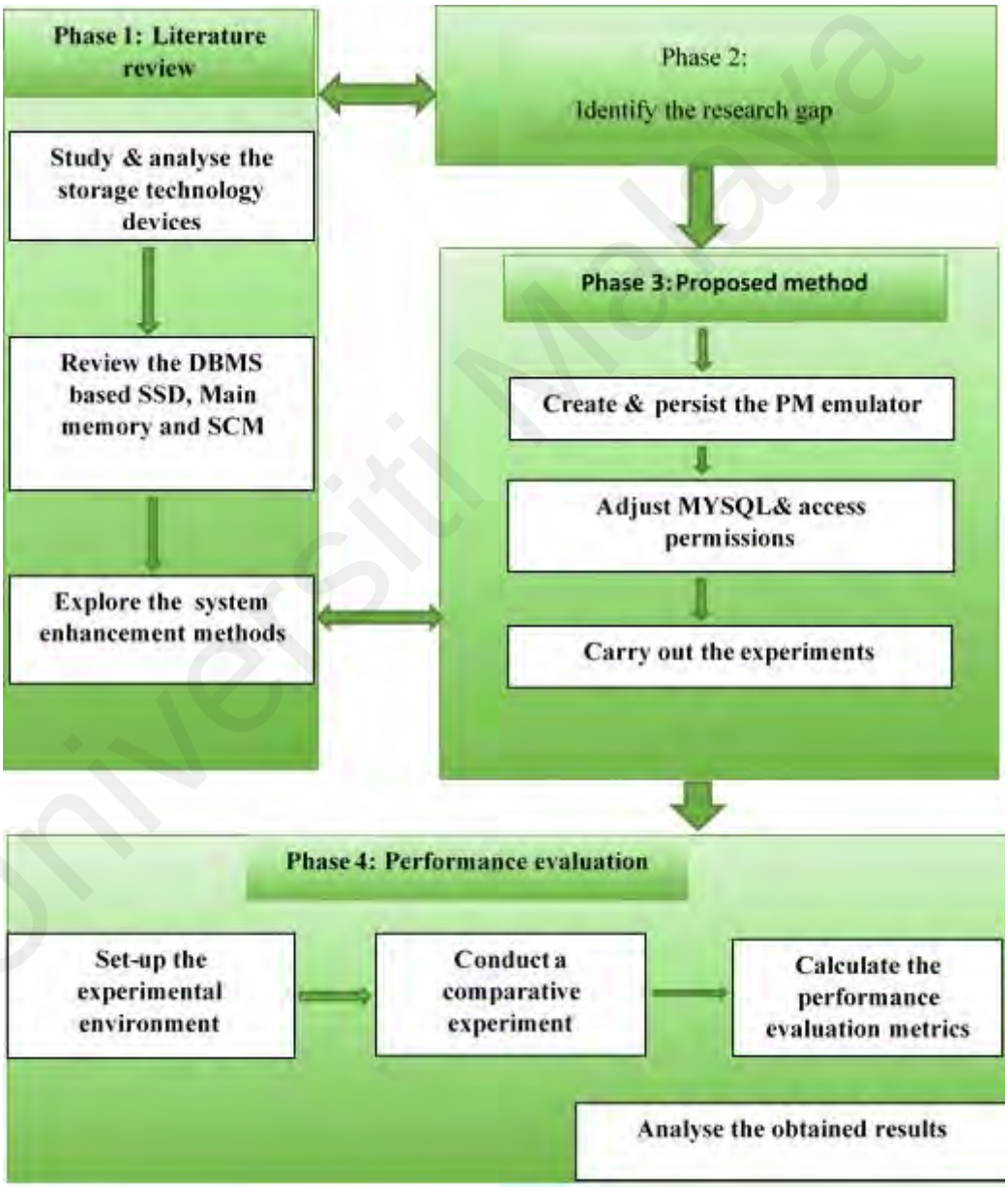

Figure 3.1: Research Methodology Phases

Phase 1: Literature review:

In this phase, the literature about storage devices and NVM-DBMS were reviewed and analyzed. Chapter 2 was devoted to covering this phase.

Phase 2: Defining the research problem and objectives:

From reviewing and analyzing the literature about NVM-DBMS, the research problem and objective were defined. Chapter 1 was devoted to covering this phase.

Phase 3: Implementing the proposed solution:

This phase addresses the proposed method to enhance system performance using persistent ZRAM and its implementation. Next sections 3.2 and 3.3 will cover this phase in detail.

Phase 4: Evaluation:

In the last phase, an evaluation of the proposed method is done to examine if the thesis objectives are fulfilled and if the proposed method offers better performance. This phase is covered in detail in Chapter 4.

3.2 Proposed Method

The steps undergone via the proposed method of this research are illustrated in Figure 3.2 below.

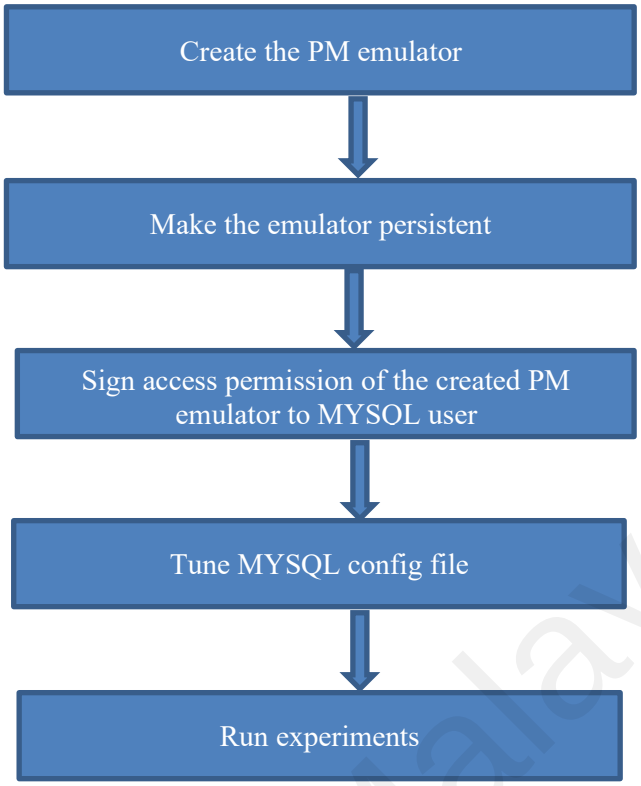

Figure 3.2: Proposed Method Steps

3.2.1 Create PM Emulators

For the creation of the persistent memory emulator, this research employed two types of available Linux kernel modules based on RAM namely: ZRAM and RAMDISK. This is in addition to a PMEM emulator that is based on dynamic random-access memory (DRAM).

3.2.1.1 ZRAM

ZRAM was earlier known as a compchache. It is a Linux kernel module available in Linux kernels since kernel version 3.14. ZRAM creates random access memory-based block drives with the compression on the fly feature. All pages written to such drives are squeezed and kept in the main memory. Thus, it offers faster I/O operations in addition to its compression feature that delivers a fine amount of memory savings. ZRAM can be used as a general-purpose RAMDISK or as a swap device. Figure 3.3 shows the steps undergone in creating the ZRAM.

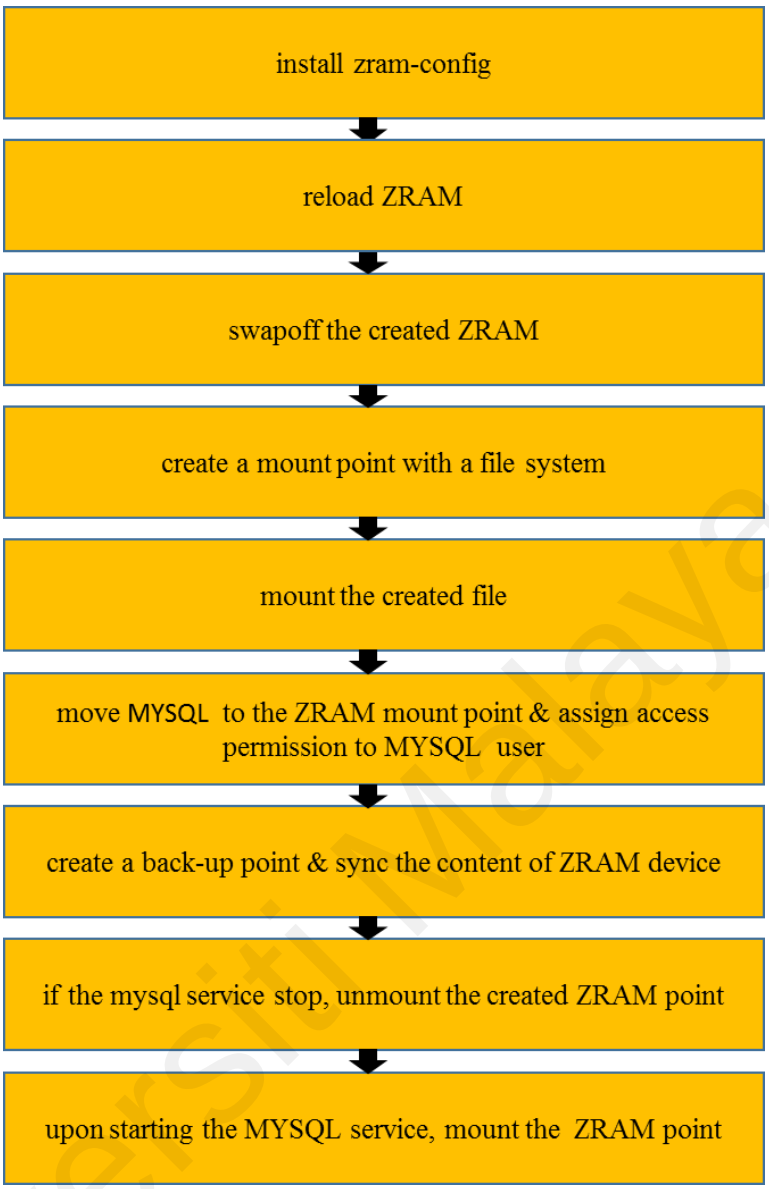

Figure 3.3: PZRAM Creation Steps

3.2.1.2 RAMDISK

RAMDISK is a chunk of available computer memory that is used by the operating system (OS) as a temporary disk drive for storing temporal data. The RAMDISK performs faster than the SSDs and hard drives.

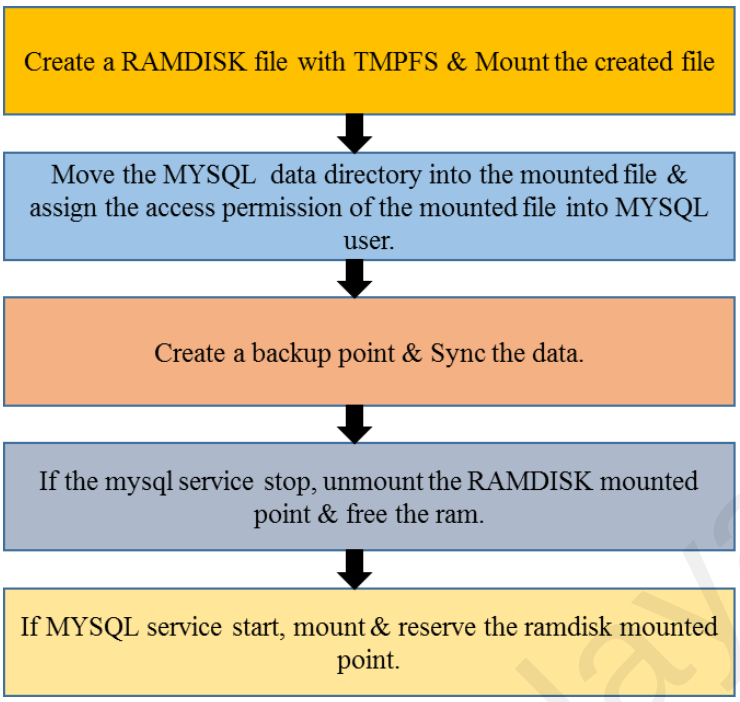

Figure 3.4: Pramdisk Creation Steps

3.2.1.3 PMEM

PMEM is a dynamic random-access memory-based emulation of the persistent memory which is a byte-addressable and non-volatile memory that can be accessed with common loads and stores via the same interface to a conventional DRAM. This emulation is provided in Linux kernel version 4.0 and above. Figure 3.5 shows the steps undergone in creating PMEM.

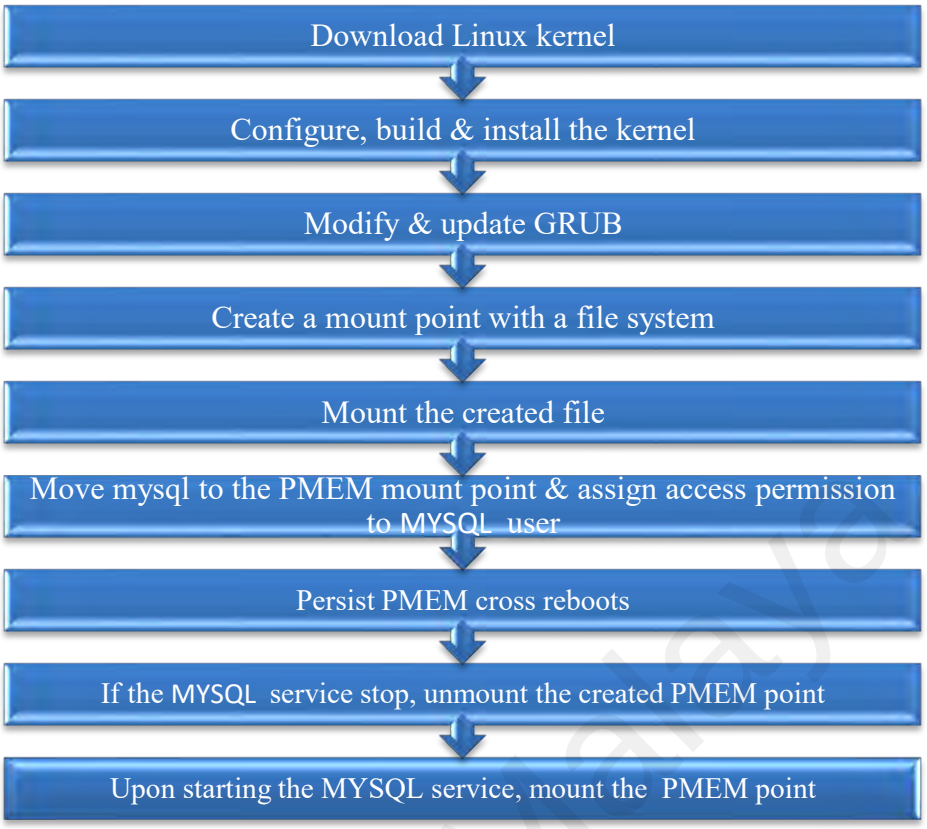

Figure 3.5: PMEM Creation Steps

3.2.2 Persisting PM Emulator

Since the emulators are based on dynamic random-access memory (DRAM) which is a volatile memory, any stored data will be lost by shutting down. As a result, in this research, three of the emulators were made persistent to appear as non-volatile memory (NVM). This can be done either by providing the NVM via a battery or by synchronizing it to non-volatile storage. This research employed the second approach to make a persistent ZRAM and ramdisk as follows:

- **First**, a backup directory was created to synchronize the MYSQL directory from and to ZRAM or the ramdisk mount point.
- **Second,** a script was created at /etc/init.rd/ramdisk to synchronize the contents of ZRAM or RAMDISK to a persistent backbone device.
- **Finally**, for a periodically synchronizing back from RAMDISK to HDD or SSD, the command was put into the crontab file

3.2.3 Sign Access Permission to PM Emulators

For security matters, MYSQL requires an MYSQL user that has its access permission to access its contents (Manual, 2013). To sign access permission to MYSQL users, this research employed two techniques.

Running Chown command: This is a Linux command that's responsible for changing users' ownership of a file, link, or directory in Linux (van Vugt, 2015, p. 172).

Modifying and reloading Apparmor: where an AppArmor is a security module supplied by Linux kernel to give the systems' administrators, ability to limit the program's resource and access (Gruenbacher & Arnold, 2007).

3.2.4 Tuning MYSQL Configuration File

To get an optimized MYSQL server performance, there was a need to tune some MYSQL variables. Hence the researcher tuned some of these variables to achieve the study's target.

```
basedir                 = /usr
datadir                 = /mnt/ramdisk/mysql #moving data dir to PRAMDISK
tmpdir                  = /tmp
lc-messages-dir         = /usr/share/mysql
skip-external-locking
#skip-innodb_doublewrite
Innodb_flush_method             =O_DIRECT
innodb_buffer_pool_size         =   2GB
innodb_buffer_pool_instances    =  8
innodb_file_per_table            =  1
```

- Increasing the size of the buffer pool had a great impact on the performance as a larger buffer pool produces better performance.
- The flush method also influences the performance, especially for I/O applications.
- Increase the number of buffer pool instances.

- Enabling the InnoDB file per file is a good option
- Changing the MYSQL data directory to a faster storage media result in a superior performance improvement.

3.2.5 Running Experiments

To run the experiments, there was a need to set and configure the system environment accordingly. The experiments comprised three stages: testing-based data placement devices, based types of filesystems, and the moved data files.

- When running experiments to test MYSQL performance based on utilizing the data placement device, the MYSQL data directory variable was tuned with the new location as follows:

```
datadir                    = /mnt/ramdisk/mysql #moving datadir to PRAMDISK
```

- When running experiments to test MYSQL performance based on the type of filesystem, the filesystem was made on a disk/block drive and mounted to a mount point. For instance, the researcher first run mkfs.ext4 -F /dev/zram0 and later ran the command mount /dev/zram0 /mnt/ramdisk.
- When running experiments to test mysql performance based on the moved file data, the specific file was moved and mysql tuned accordingly. For example, for moving logs, the researcher set the new location for the variable innodb_log_group_home_dir = /mnt/ramdisk in configuration file of MYSQL /etc/mysql/mysql.conf.d/mysqld.cnf. Consequently, for moving ibdata , the researcher set the innodb_data_file_path =“ibdata1:12M:autoextend” and innodb_data_home_dir = /mnt/ramdisk in MYSQL configuration file.

3.2.4.1 Running HammerDB TPC-C MYSQL Benchmark

After carrying out the proposed method, the experiment stage was performed. Initially, the server was rebooted; and the following command was run to start the

PZRAM/PRAMDISK/PMEM service as well as to start up MYSQL:

```
sudo etc/init.d/Pramdisk start

sudo service mysql start
```

Afterward, Hammerdb3.2 was run to do the experiments as illustrated in the following photos from Figure 3.6- 3.13 respectively.

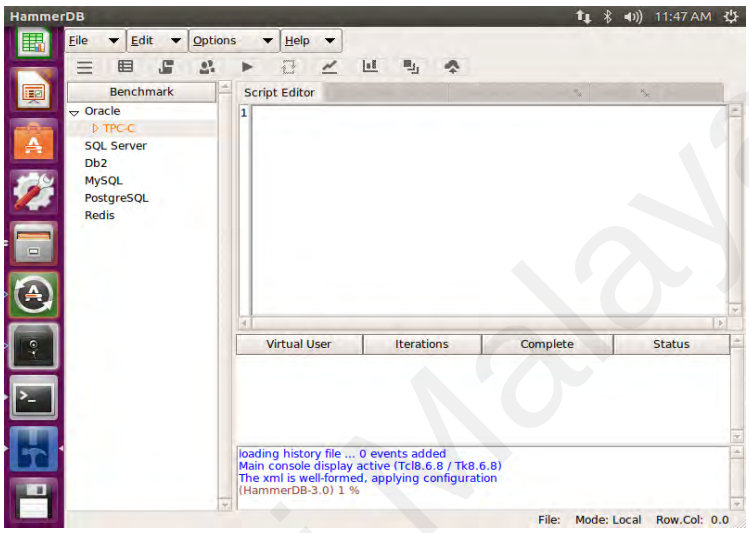

Figure 3.6: Running HammerDB

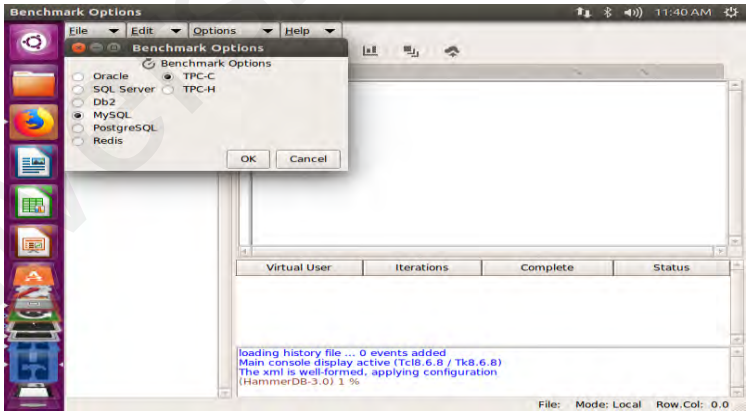

Figure 3.7: Selecting MYSQL TPC-C Benchmark

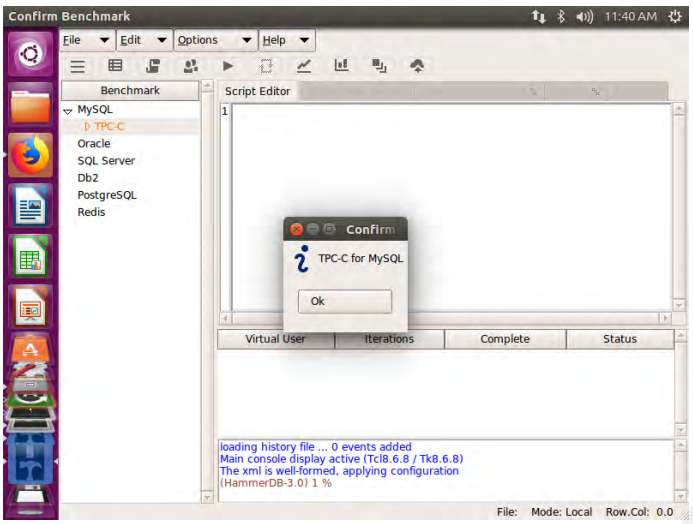

Figure 3.8: Confirming TPC-C for MYSQL

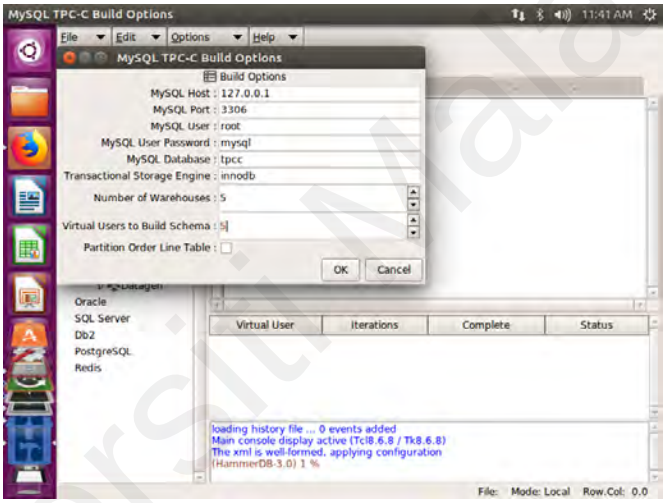

Figure 3.9: Building Schema Options

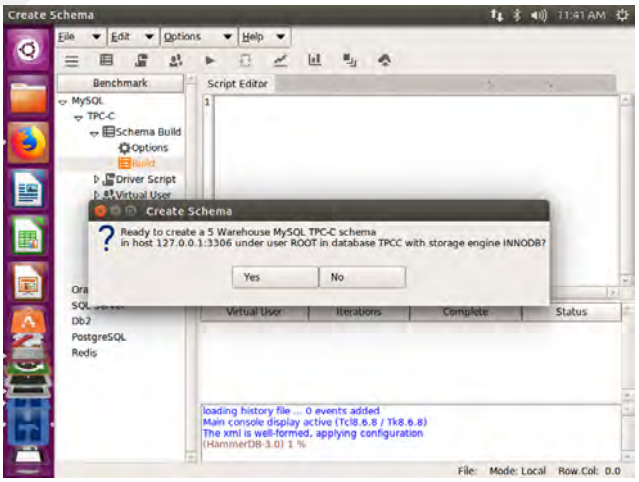

Figure 3.10: Building Schema

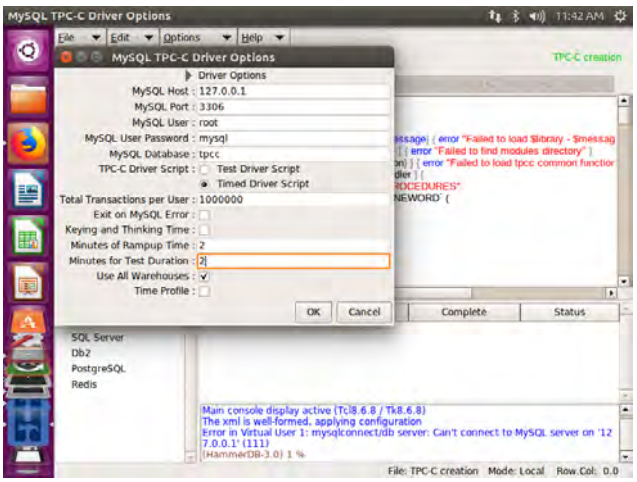

Figure 3.11: Selecting the Driver Options

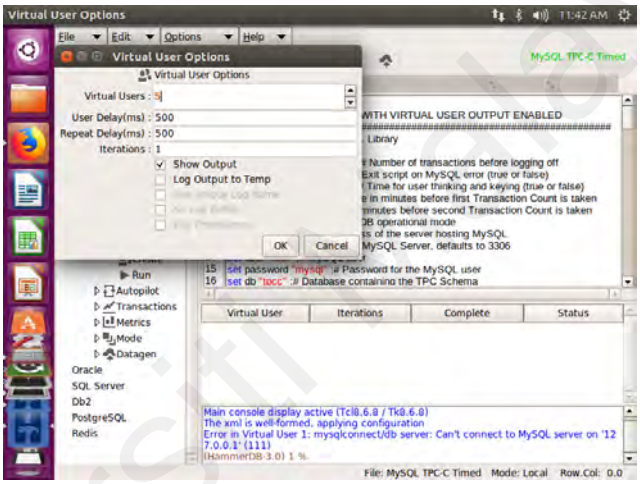

Figure 3.12: Defining the Number of Users

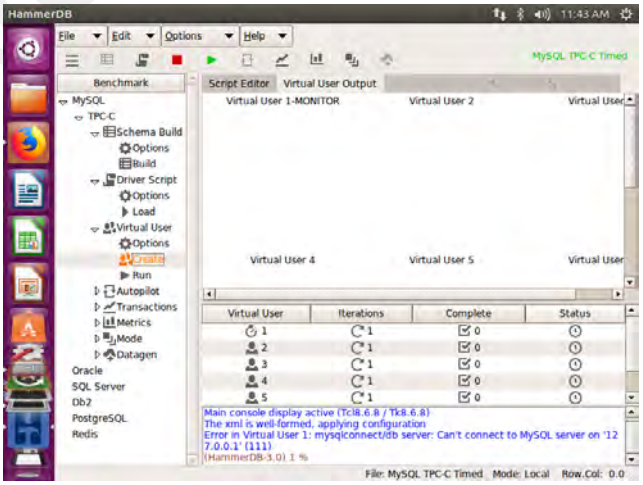

Figure 3.13: Creating the Virtual Users

After creating this step, the program was run and it led to the collection of data, of which the results are presented as illustrated in Figure 3.14.

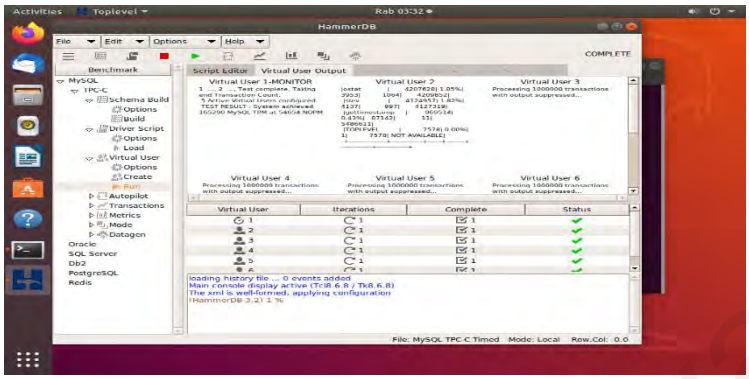

Figure 3.14: TPC-C collects the TMP & NOPM

3.3 Implementation

To implement the proposed method, some scripts were created alongside modification of some system files.

3.3.1 Source Code Files' Tree

In this section, the scripted files were structured, and the system modified files to implement the study's proposed method. Figure 3.15 demonstrates the created and modified files tree accordingly.

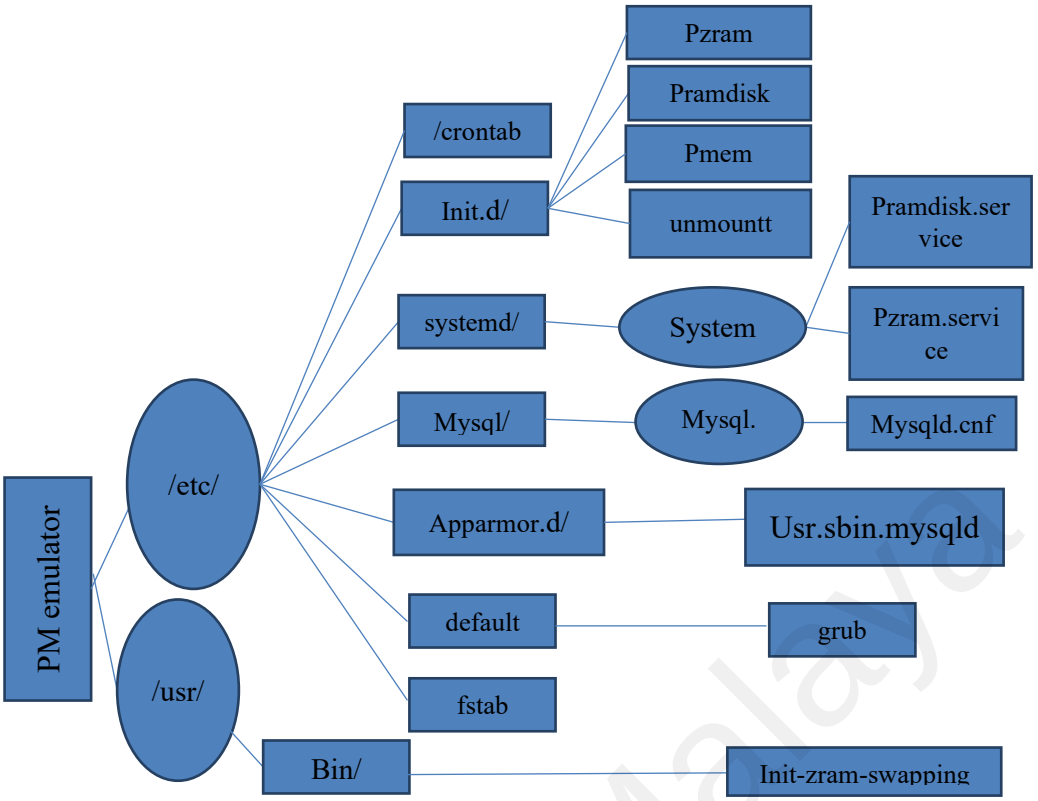

Figure 3.15: Source Code Files' Tree

3.3.1.1 Created Source Code Files

- /etc/init.d/Pzram: this file contains the creation script of persistent ZRAM.
- /etc/init.d/Pramdisk: this file contains the creation script of persistent RAMDISK.
- /etc/init.d/pmem: this file contains the creation script of PMEM.
- /etc/init.d/umounting: to mount the created ZRAM/PMEM/PRAMDISK and their mount point.
- /etc/systemd/system/Pzram: to run the PZRAM script automatically.
- /etc/systemd/system/Pramdisk: to run the PRAMDISK script automatically.
- /etc/crontab: to periodically sync data into PZRAM/PRAMDISK and HDD/SSD.

3.3.1.2 Modified Source Code Files

- /etc/mysql/mysql.conf.d/mysqld.cnf: this is the file configuration of MYSQL.
- /etc/apparmor.d/usr.sbin.mysql: this contains the configuration of apparmor for MYSQL access.

- /etc/default/grub: this file contains a GNU and boot system info. This file was used to reserve a memory region for PMEM.
- /etc/usr/bin/init_zram_swapping: this file creates and initializes the ZRAM.

3.3.2 Source Code

Here, the codes and commands used in carrying out the proposed method are presented.

3.3.2.1 PZRAM

❖ **/etc/init.d/Pzram script:**

```
#!/bin/bash
# This script to create a persistent zram block drive
# Creation of ramdisk/zram mount point
if [  -d /mnt/ramdisk]; then
            echo “ramdisk mount point already created"
else
            sudo mkdir /mnt/ramdisk
fi
#Loading zram
modprobe num_devices = 1
#Deactivate swap on zram0
swapoff /dev/zram0
#Making a filesystem on zram0
#mkfs.ext4 -F /dev/zram0 #making ext4 on zram0 disk
#mkfs.xfs -f /dev/zram0    #making xfs on zram0 disk
#mkfs.f2fs -f /dev/zram0   #making f2fs on zram0 disk
#mkfs.btrfs -f /dev/zram0  #making btrfs on zram0 disk
#Mounting the selected disk-based filesystem on zram
# mount /dev/zram0 /mnt/ramdisk
#Mounting tmpfs on zram0
if mountpoint -q /dev/zram0 /mnt/ramdisk; then
echo “/dev/zram /mnt/ramdisk is mounted”
else
sudo mount -t Tmpfs /dev/zram0 /mnt/ramdisk
fi
#Moving mysql directory to /mnt/ramdisk
#Move mysql data directory to /mnt/ramdisk
mv /var/lib/mysql /mnt/ramdisk
#Sign access permission to mysql user
chown -R mysql:mysql /mnt/ramdisk/mysql
#Restarting mysql service after tuning its file configuration before running this script
sudo service mysql restart
# Create a ramdisk backup directory
if [  -d /var/ramdisk-backup]; then
            echo “ramdisk backup directory already created"
```

```
else
        sudo mkdir /var/ramdisk-backup
fi
#Synching the ramdisk contents to hdd and verse vice
case "$1" in
  start)
    echo "Copying files to ramdisk"
    rsync -av /var/ramdisk-backup/ /mnt/ramdisk/
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched from HD >>
/var/log/ramdisk_sync.log
    ;;
  sync)
    echo "Synching files from ramdisk to Harddisk"
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched to HD >>
/var/log/ramdisk_sync.log
    rsync -av --delete --recursive --force /mnt/ramdisk/ /var/ramdisk-backup/
    ;;
  stop)
    echo "Synching logfiles from ramdisk to Harddisk"
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched to HD >>
/var/log/ramdisk_sync.log
    rsync -av --delete --recursive --force /mnt/ramdisk/ /var/ramdisk-backup/
    ;;
  *)
    echo "Usage: /etc/init.d/ramdisk {start|stop|sync}"
    exit 1
    ;;
esac

exit 0
```

After creating the script, the /etc/init.d/Pzram file was enabled for execution by running the following command in the terminal:

```
sudo chmod u+x /etc/init.d/Pzram
```

Then, the script file was set up to run at startup by running the following command in the terminal:

```
update-rc.d Pzram defaults 00 99
```

Finally, the file was synchronized to its backup directory by running the following command:

```
/etc/init.d/Pzram sync
```

And for periodically synchronizing back from RAMDISK to HDD or SSD, the following command was inputted into the crontab file:

```
2 * * * * root        /etc/init.d/Pzram sync >> /dev/null 2>&1
```

- **PZRAM Unmount script /etc/init.d/unmount**

```
#! /bin/bash
 sudo umount /dev/zrm0 /mnt/ramdisk
```

- **/etc/systemd/system/Pzram. Service:**

```
[Unit]
Description=Create persistent zram as a PM emulator

[user]
User=root
Group=root

[Service]
Type=simple
WorkingDirectory=/usr/bin
ExecStart=/bin/bash Pzram start
KillMode=process
ExecStop=/bin/bash unmount stop
[Install]
WantedBy=default.target
```

- **/etc/usr/bin/init_zram_swapping:**

The size of ZRAM was modified from the following line to set 2G from:

```
mem=$(((totalmem / 2 / ${NRDEVICES}) * 1024))
```

to

```
mem=$(((totalmem * 3 / ${NRDEVICES}) * 1024))
```

- **/etc/fstab**

```
dev/zram0 /mnt/zram     tmpfs rw,relatime,noatime 0 0
```

- /etc/apparmor.d/usr.sbin.mysql:

The **/etc/apparmor.d/usr.sbin.mysql/** file was modified by adding these two lines under **# Allow data dir. access** line:

```
/mnt/ramdisk/mysql/ r,
/mnt/ramdisk/mysql/** rwk,
```

3.3.2.2 PRAMDISK

/etc/init.d/Pramdisk script:

```
#!/bin/bash
# This script to create a persistent ramdisk using tmpfs
# Creation of ramdisk/zram mount point
if [  -d /mnt/ramdisk]; then
            echo “ramdisk mount point already created"
else
            sudo mkdir /mnt/ramdisk
fi
#Mounting the ramdisk
If !mountpoint -q /mnt/ramdisk; then
sudo mount -t Tmpfs -o size=1G  tmpfs /mnt/ramdisk
fi
#Move mysql data directory to /mnt/ramdisk
mv /var/lib/mysql /mnt/ramdisk
#Sign access permission to mysql user
chown -R mysql:mysql /mnt/ramdisk/mysql
#Restarting mysql service after tuning its file configuration before running this script
sudo service mysql restart
# Create a ramdisk backup directory
if [  -d /var/ramdisk-backup]; then
echo “ramdisk backup directory already created"
else
            sudo mkdir /var/ramdisk-backup
fi
#Synching the ramdisk contents to hdd and vice versa
case "$1" in
  start)
    echo "Copying files to ramdisk"
    rsync -av /var/ramdisk-backup/ /mnt/ramdisk/
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched from HD >>
/var/log/ramdisk_sync.log
    ;;
  sync)
    echo "Synching files from ramdisk to Harddisk"
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched to HD >>
/var/log/ramdisk_sync.log
    rsync -av --delete --recursive --force /mnt/ramdisk/ /var/ramdisk-backup/
    ;;
  stop)
    echo "Synching logfiles from ramdisk to Harddisk"
    echo [`date +"%Y-%m-%d %H:%M"`] Ramdisk Synched to HD >>
/var/log/ramdisk_sync.log
    rsync -av --delete --recursive --force /mnt/ramdisk/ /var/ramdisk-backup/
    ;;
  *)
    echo "Usage: /etc/init.d/ramdisk {start|stop|sync}"
```

```
    exit 1
    ;;
esac

exit 0
```

After creating the script, the /etc/init.d/Pramdisk file was enabled for execution by running the following command in the terminal:

```
sudo chmod u+x /etc/init.d/Pramdisk
```

Then, the script file was set up to run at start-up by running the following command in the terminal:

```
update-rc.d Pramdisk defaults 00 99
```

Finally, the file was synchronized to its backup directory by running the following command:

```
/etc/init.d/Pramdisk sync
```

And for periodically synchronizing back from RAMDISK to HDD or SSD, the following command was inputted into the crontab file:

```
2 * * * * root        /etc/init.d/Pramdisk sync >> /dev/null 2>&1
```

- **PRAMDISK Unmount script /etc/init.d/unmount**

```
#! /bin/bash
 sudo umount /mnt/ramdisk
```

- **/etc/systemd/system/Pramdisk. Service:**

```
[Unit]
Description=Create persistent ramdisk as a PM emulator

[user]
User=root
Group=root

[Service]
Type=simple
WorkingDirectory=/usr/bin
ExecStart=/bin/bash Pramdisk start
KillMode=process
ExecStop=/bin/bash unmount stop
[Install]
WantedBy=default.target
```

❖ **/etc/fstab**

This file was modified by mounting the created RAMDISK file at the boot time by adding the following command into the /etc/fstab:

```
tmpfs           /mnt/ramdisk tmpfs      defaults, size=4096M 0 0
```

❖ /etc/apparmor.d/usr.sbin.mysql:

At this point, the **/etc/apparmor.d/usr.sbin.mysql**/ file is modified by adding these two lines under **# Allow data dir. access** line:

```
/mnt/ramdisk/mysql/ r,
/mnt/ramdisk/mysql/** rwk,
```

3.3.2.3 PMEM

❖ **/etc/init.d/Pmem script:**

```
#!/bin/bash
# This script to create a mount point for PMEM
# Creation of PMEM mount point
if [  -d /mnt/ramdisk]; then
            echo “ramdisk mount point already created"
else
            sudo mkdir /mnt/ramdisk
fi
#Making a filesystem on top of pmem
sudo mkfs.ext4  /dev/pmem0
#Mounting the pmem on a mount point
If !mountpoint -q /dev/pmem0 /mnt/ramdisk; then
sudo mount /dev/pmem0 /mnt/ramdisk
fi
#Move mysql data directory to /mnt/ramdisk
mv /var/lib/mysql /mnt/ramdisk
#Sign access permission to mysql user
chown -R mysql:mysql /mnt/ramdisk/mysql
#Restarting mysql service after tuning its file configuration before running this script
sudo service mysql restart
```

- **/etc/default/grub:**

This file was modified to reserve a memory region of 2G for this study's created PMEM block device.

```
GRUB_CMDLINE_LINUX="memmap=2G!4G"
```

- /etc/apparmor.d/usr.sbin.mysql:

The **/etc/apparmor.d/usr.sbin.mysql**/ file was modified by adding these two lines under **# Allow data dir. access** line:

```
/mnt/ramdisk/mysql/ r,
/mnt/ramdisk/mysql/** rwk,
```

- **PMEM Unmount script /etc/init.d/unmount**

```
#! /bin/bash
sudo umount /dev/pmem0 /mnt/ramdisk
```

For PMEM, there was no need to do a synchronization or set the script to run at start-up, the only thing needed was to add the following line into /etc/fstab to persist the image of PMEM on reboots, after which PMEM appears as a disk partition with persistent contents:

```
/dev/pmem0 /mnt/mem    ext4 rw,relatime,dax  data ordered 0 0
```

3.4 Summary

In summary, this chapter described the proposed method and its implementation from creating the PM emulator, persisting it, signing its access permission, to the tuning of MySQL configuration file and presenting the experimental steps in detail.

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, obtained results from the experimental stage are presented. Further, the chapter reviews the experiment setup and tools and discusses the difference between using three data placement media for the MYSQL data directory: NVM emulator, SSD, and HDD. Furthermore, deliberations are provided on the influence of file system type on the performance.

4.2 Experimental Setup

MYSQL version 5.7.31 was used as the case study application for this research. The benchmark workload for the MYSQL server was generated using HammerDB version 3.2. The next sections outline the details of the experimental setup.

4.2.1 Server Specification

The configuration of the server is summarized in Table 4.1.

Table 4.1: Server Information

Specification	Description
Processor	processor Intel® Court i3-8100 CPU @ 3.60GHz × 4
Memory	4G
Storage	HDD 1TB, Samsung SSD 128 GB
Cores	4
Operating System	Ubuntu 18.04.4/Linux kernel-4.15.9
Application	MYSQL -5.7.31 server
Benchmark	Hammerdb-3.2

4.2.2 MYSQL Specification

MYSQL version 5.7.31 was run on the server with a 1Gb buffer pool and 8 instances. For this study, the experiment was run on MYSQL with the tuning of specific MYSQL variables. Additionally, different MYSQL InnoDB files such as log files, system-tablespace, and the whole data directory, were moved. MYSQL was configured with tuning the following variables:

- **Innodb_buffer_ pool_size:** the size of the buffer pool was set to 1G.
- **Innodb_buffer_ pool_ instances:** this option was configured with 8 instances.
- **Innodb_file_per_table:** this variable was enabled by setting to 1.
- **Innodb_flush_method:** the flush method option was set to O_direct.

4.2.3 Data Placement

The experiment was divided into three parts based on the placement of the MYSQL Data directory.

- **PM emulator:** in the first set of experiments, the MYSQL data directory was placed in a PRAMDISK/PMEM/PZRAM.
- **SSD:** In the second set of experiments, the MYSQL data directory was placed on an SSD drive.
- **HDD:** In the third set of experiments, the MYSQL data directory was placed on an HDD (results of this experiment are not mentioned in this chapter, it was used in the problem statement section).

4.2.4 Filesystem Running on PM Emulator

In addition to the type of data placement media, the experiments were run based on the type of running file system on top of the PM emulators. The used file systems in this experiment are listed as follows:

- Ext4 (Djordjevic, 2012): a disk-based filesystem developed to support large

volumes of stored data (1 exabyte).

- XFS (XFS Overview, 2013): a disk-based filesystem that offers a high-performance using B+ trees for directories and file allocation.
- Btrfs (Btrfs Contributors at kernel.org, 2016): an Oracle-designed file system for Linux based on the copy-on-write (COW) concepts.
- F2FS (Lee, 2015): a flash-based filesystem designed for flash SSD devices.
- TMPFS (Snyder, 1990): a RAM filesystem that overcomes the limitations of RAMFS.
- RAMFS (Li, 2011): a random-access memory file system that grows dynamically.
- PMFS (Dulloor, 2016): a persistent file system developed by Intel for persistent memory (Shaw, 2012).

4.2.5 MYSQL Data Files

The experiments were also run according to the stored MYSQL files on the PM emulator to find out their effect on performance. These files are:

- MYSQL data directory (datadir).
- MYSQL log files (ib_log_buffer0/ib_log_buffer1).
- MYSQL system tablespace (ibdata1).

4.2.6 Benchmark Application

HammerDB version 3.2 (Shaw, 2012) was used as a benchmark application. HammerDB implements the TPC-C standard, which is a standard test suite for database performance testing. The tests simulate a warehousing system accessed through online users. Users place online orders for the items distributed among multiple warehouses. The size of the test database can be varied by changing the number of warehouses (Shaw, 2012). This research tested the load by setting the number of warehouses to 5 and the

number of users to 5 as well, where the total of transactions is set to 1000000 users' transactions with enabling the use of all warehouses. Every database transaction consists of order placement and order fulfilment. HammerDB tests are suitable for transactional workload where a mix of reading and write operations are generated by the transactions. The performance of a database server system is tested in terms of transactions per minute (TPM) and Number of new orders (NOPM) and latency/response time.

4.2.7 Performance Parameter

Since the throughput and latency are the most significant database management system (DBMS) performance metrics (Bhimani et al., 2017), This study utilizes these two performance metrics to evaluate the proposed method performance. The transaction per minute (TPM) (Shaw, 2012) and the number of new orders per minute (NOPM) (Shaw, 2012) were used to measure and collect the number of transactions completed per unit (throughput); The latency time where the time taken by system to response is collected by noting the time taken from running the build schema to its completion.

4.3 Results

This section presents the obtained results from the performed experiments on the server with three types of data placement media, moved MYSQL data files, and type of filesystem running on the PM emulators in terms of TPM, NOPM, and latency time.

4.3.1 Data Placement Media

MYSQL by default stores its data into the HDD unless the user/admin moves its data directory to another storage device. However, in this research, the MYSQL data directory option was tuned to set in SSD, PRAMDISK, PMEM, and PZRAM as prime storage for the MYSQL data directory in each experiment stage. This experimental stage is the most important one because it is concerned about the core objective of this study (enhancing system performance by moving MYSQL data directory into an SCM emulator). To

distinguish from the other stages, we have made the number of iterates 3 and the rest of experiments has only two iterates. Table 4.2 shows the results obtained from testing MYSQL on all data placement media listed above.

Table 4.2: Results based on Data Placement Media

Iterates	SSD			PMEM			PRAMDISK			PZRAM		
	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time
1	18512	6193	51sc	229304	75528	34sc	233216	77283	34sc	234567	77234	33sc
2	18509	6103	49sc	229400	75663	35sc	234390	76932	33sc	234127	76961	34sc
3	18389	6037	49sc	230018	75757	35sc	234681	76953	34sc	236931	78282	33sc

With an SSD, the transaction per minute (TPM) reached a peak of 18512 and the number of new orders (NOPM) got 6193 with a latency of 51sc-49sc. PMEM reached a peak of 229304 for transaction per minute (TPM) and 75757 new orders per minute (NOPM) with a latency of 34sc -33sc. PRAMDISK achieved the highest transaction per minute (TPM) at 234681 and 77283 for new order per minute (NOPM) with a latency of 34sc-34sc. Finally, PZRAM reached its highest transaction per minute (TPM) at 236931 and 78282 for its new order per minute at a time latency of 34sc-33sc.

4.3.2 Filesystem Running on PM Emulator

In this section, the results of experiments run are presented based on the type of running file system on top of PM emulators namely, PMEM, PRAMDISK, and PZRAM. For each emulator, the suitable filesystems that match have been used.

4.3.2.1 PZRAM and Filesystems

For PZRAM, a collection of disk-based filesystems and RAM filesystems were used

on top of the created disk ZRAM. Tables 4.3 and 4.4 demonstrates the results of each used filesystem on top of ZRAM.

Table 4.3: Results of ZRAM based on Disk Filesystems

Iterates	Ext4			XFS			Btrfs			F2FS		
	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time
1	211865	69984	34sc	182641	60221	34sc	158806	52588	35sc	213265	70410	34sc
2	211696	69944	35sc	182403	60166	35sc	159318	52726	35sc	214529	71275	34sc

ZRAM with Ext4 reached a peak of 211856 for the transaction per minute (TPM) and 69984 number of new orders (NOPM) with a latency of 35sc-34sc. With XFS, it reached a peak of 182641 for transaction per minute (TPM) and 60221 new orders per minute (NOPM) with a latency of 35sc -34sc. With Btrfs, the highest transaction per minute (TPM) reached were at 159318 and 52726 for new order per minute (NOPM) with a latency of 35sc. Consequently, F2FS reached its highest transaction per minute (TPM) at 214529 and 71275 for its new order per minute at a time latency of 34sc-33sc. TMPFS and RAMFS reached the peak of their transaction per minute (TPM) at 236931, 235652 respectively, and new order per minute (NOPM) peak at 78282, 77669 respectively with 33sc-34sc latency time.

Table 4.4: Results of ZRAM based on RAM Filesystem

Iterates	TMPFS			RAMFS		
	TPM	NOPM	Time	TPM	NOPM	Time
1	234127	76961	34sc	234172	77349	34sc
2	236931	78282	33sc	235652	77669	34sc

4.3.2.2 PRAMDISK and Filesystems

The results of creating the RAMDISK with different methods like creating it only through mounting filesystem like TMPFS/RAMFS or using /dev/shm are shown in Table 4.5 below.

Table 4.5: Results of PRAMDISK with Different Implementation Methods Criteria

Iterates	RAMFS			RAMFS /dev/shm			TMPFS			TMPFS /dev/shm		
	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time
1	234433	77348	33sc	232173	76940	34sc	233216	77283	34sc	234470	77343	34sc
2	233734	77677	34sc	232646	76846	33sc	234681	76953	33sc	234501	77501	35sc

Creating the RAMDISK with mounting RAMFS helped it in reaching its highest transaction per minute (TPM) at 234443 and the new order per minute (NOPM) at 77677 at a latency of 34sc-33sc. When creating using /dev/shm with RAMFS on top of it, it was able to reach a 232646 transaction per minute (TPM) and a 76940 new order per minute (NOPM) at a latency of 34sc-33sc. For TMPFS, the peak of transaction per minute (TPM) was at 234681 and the new order per minute (NOPM) was attained at 77283 at a time latency of 33sc-34sc. Finally, the /dev/shm based TMPFS reached the highest point of transaction per minute (TPM) at 234501 and 77501 for new orders per minute (NOPM) at a latency of 35sc-34sc.

4.3.2.3 PMEM and Filesystems

The findings of testing PMEM with types of disk filesystems that support DAX and persistent memory file system (PMFS) are listed in Table 4.6 below.

Table 4.6: Results of PMEM with Filesystems

Iterates	Ext4			XFS			PMFS		
	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time
1	229304	75528	34sc	203467	67430	34sc	228952	75652	34sc
2	229400	75663	35sc	204625	67121	35sc	229481	75989	34sc

PMEM with Ext4 was able to attain a 229400 transaction per minute (TPM) and 75663 new orders per minute (NOPM) at a time latency of 35sc-34sc. With XFS, it was able to reach a peak of transaction per minute (TPM) at 204625 and 67430 new orders per minute (NOPM) at a latency time of 35sc-34sc. With PMFS on top of PMEM, the highest transaction per minute (TPM) at 229481 and 75989 new orders per minute (NOPM) were achieved at a latency of 34sc.

4.3.3 MYSQL Data Files

The obtained findings from moving the three types of MYSQL data files are demonstrated in Table 4.7 below.

Table 4.7: Results of Moving Different Data Files

Iterates	Logs			ibdata			Logs & ibdata			datadir		
	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time	TPM	NOPM	Time
1	165606	54644	42s	19701	6510	46s	184909	62203	41s	233216	77283	34sc
2	165290	54654	42s	19540	6558	44s	184060	62629	44s	234390	76932	33sc

Moving only log files gets its peak transaction per minute (TPM) at 165606 and new orders per minute at 54654 at a time latency of 42sc. An only moved ibdata reached a 19701 transaction per minute (TPM) and 6558 at a latency of 46sc-44sc. The moved logs & ibdata together provide about 184909 transactions per minute and attained a new order

per minute (NOPM) of 62629 at a latency of 44sc-41sc. Moving the whole data directory shows its highest transaction per minute at 234390 and 77282 new orders per minute at a latency of 34sc-33sc.

4.4 Discussion

This section is devoted to a discussion of the research findings in the following subsections.

4.4.1 Data Placement Media

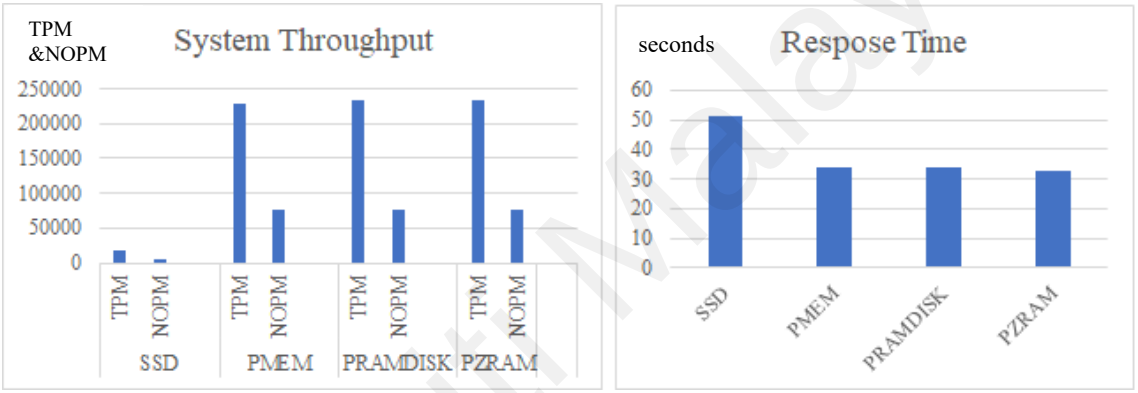

Figure 4.1: System Performance-based on Type of Data Placement Devices

According to results from Figure 4.1, it is revealed that an SSD performance falls behind the three types of PM emulators in both performance parameters: throughput and response time. This is due to the fact that the SSD is orders of magnitude slower than DRAM in which SSD latency times are counted in microseconds whereas the DRAM is in a nanosecond. Further, DRAM outperforms the SSD in the bandwidth and endurance, etc. (Dulloor, 2016; Ma et al., 2016; Oukid & Lersch, 2019). Since the three proposed emulators are based on the DRAM, it is proved that the study's proposed PZRAM with TMPFS has better performance over SSD with 1167% performance improvement. PZRAM outperforms PMEM with 3.2% performance improvement and almost the same performance with PRAMDISK. This slide improvement over PMEM, because PMEM requires kernel reconfiguration and compilation which results in an extra overhead and

more occupied storage space. However, ZRAM is better than RAMDISK in which that ZRAM saves more memory capacity and squeezes the data due to its compression in fly feature.

4.4.2 Filesystem Running on PM Emulator

4.4.2.1 PZRAM and Filesystems

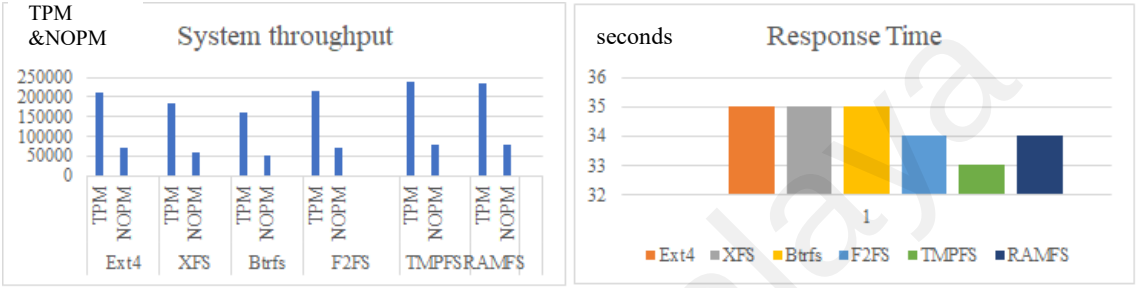

Figure 4.2: PZRAM Performance based on Filesystems

From Figure 4.2, it can be interpreted that using RAM filesystems on top of ZRAM disk outperforms disk/flash-based filesystems like Ext4 with 11.83%, XFS with 29.72%, F2FS with 10.44%, and Btrfs with 48.7% performance improvement. This is because the whole disk-based filesystems are programmed, oriented, and built to optimize disk devices. Additionally, as RAM technologies have the characters that differ them from other devices, they require a RAM/Memory-based filesystem to optimize its peak performance. Also noted from the above figure is that F2FS has a better performance than Ext4, XFS, and BTRFS. This result is because F2FS is a flash-based filesystem. After F2FS, Ext4 boasts of a better performance than XFS and Btrfs. Consequently, Btrfs has the lowest throughput.

4.4.2.2 PRAMDISK and Filesystems

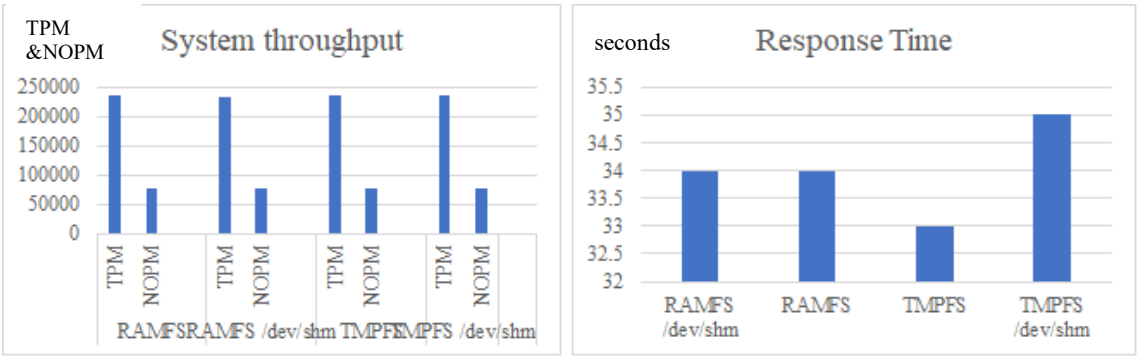

Figure 4.3: PRAMDISK Performance based on Implementation Methods

To create a RAMDISK on Linux operating system, different methods can be employed. Implementation of RAMDISK can be done through TMPFS, RAMFS, or /dev/shm with RAMFS or TMPFS on top of shared memory /dev/shm. Figure 4.3 above, shows that all RAMDISK implementation methods provided by Linux almost have the same performance with just a tiny slide of variation. Also, it can be noticed that mounting RAMFS/TMPFS as RAMDISK outperforms using /dev/shm since the /dev/shm is built as a block device and use the shared memory.

4.4.2.3 PMEM and Filesystems

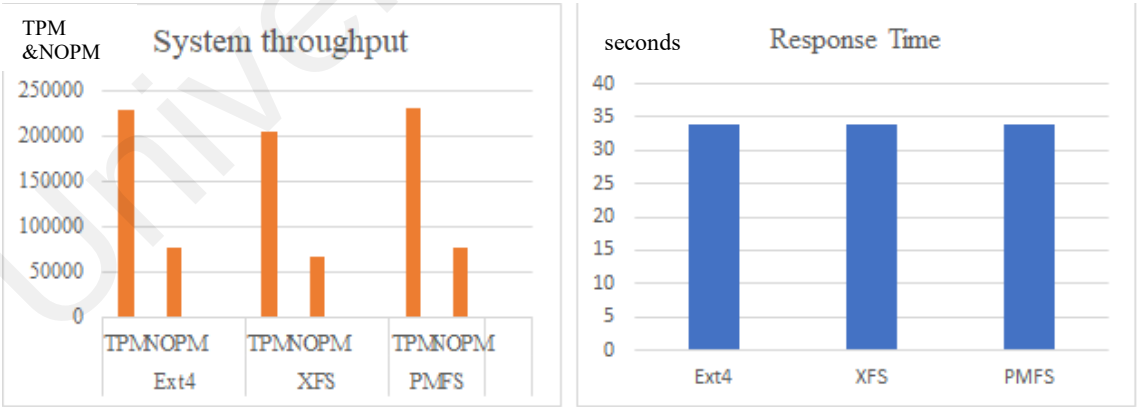

Figure 4.4: PMEM Performance based on Filesystems

From Figure 4.4, it is evident that Ext4 with DAX and PMFS has the almost same performance, where an XFS supporting DAX, falls behind both files in terms of throughput.

4.4.3 MYSQL Data Files

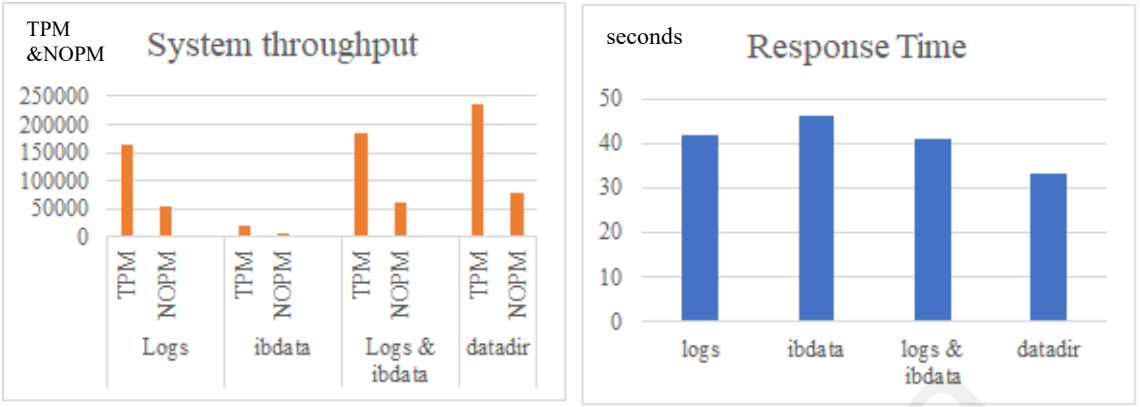

Figure 4.5: System Performance-based on Moving Data Files

It is obvious from Figure 4.5 that moving the whole data directory has the best performance with an 1167% improvement over SSD. This is because moving the data directory into the main memory has lower access time and greater bandwidth than SSD (Oukid & Lersch, 2019). Where moving the log files also showed a good influence on performance with 792.88% over the ibdata. This performance improvement is because moving logs to SCM will reduce writings to the disk. Moving system-table space (ibdata) has a slight performance improvement with 6.42% which doesn't have a noticeable significant impact on performance like the other moved files. However, it was found that moving both logs and ibdata has a higher performance with 894.27% than moving only logs. Further, it was also discovered that moving the whole data directory offers a better performance improvement with 1089.74%, 41.81%, and 27.34% performance improvement over moved ibdata, only moved logs and moved ibdata & logs respectively. This improvement comes as a result that moving the whole data directory into a PM emulator eliminates reads and writes from disks. Thus a superb performance can be gained when systems' data can be stored and accessed from the higher memory hierarchy levels like caches and main memory rather than IO reads and writes which are considered as slower memory levels (Patterson & Hennessy, 2019, p. 85).

4.5 Summary

This chapter has addressed and discussed the obtained results and made comparisons among each of the components to prove that the proposed method provides a superior MYSQL performance in terms of throughput and response time.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

Storage class memory (SCM) is a promising non-volatile, byte-addressable memory that is introduced by different memory landscape makers to bridge the performance gap between the main memory and storage drives. In this thesis, a PM emulator using PZRAM was proposed. Further, the performance of the MYSQL InnoDB engine was tested on three PM emulators in terms of throughput and response time. More so, a persistent ZRAM was built to host the MYSQL data directory as a PM. This was done to make and use a PMEM emulator and PRAMDISK as data placement media for the MYSQL data directory besides HDD and SSD. The experiment in this study contains the test of moving different data files separately, such as log files, system tablespace, and whole data directory, to a PM emulator. Also, experiments were run to examine the effect of filesystems on system performance. The study's implemented emulator persistent ZRAM (PZRAM) has shown a massive performance improvement as compared to HDD and SSD with an improvement of 14290% and 1167% respectively, a tiny higher performance than PMEM with 2.3% improvement and almost similar performance of persistent RAMDISK. Additionally, the study's proposed PZRAM with TMPFS running on top of it has provided better performance with 11.83% over PZRAM with Ext4.

5.2 Fulfilment of Research Objectives

The following are the objectives of the research and their associated research questions:

1) To explore and analyze the storage/memory technologies, their role, and limitations in database management systems.

RQ1: What is the existing storage/memory technologies, their role, and limitations in database management systems?

2) To propose and implement a method that enhances system performance using a

persistent RAM-based module (ZRAM) as SCM.

RQ2: How to enhance system performance using a persistent RAM-based module (ZRAM) as SCM?

3) To empirically evaluate the proposed method (PZRAM) through an experimental analysis to prove its ability to improve the system performance

RQ3: Is the proposed method PZRAM able to improve the system performance?

To achieve the first research objective and its associated research question; the literature review was performed. Different types of storage and memory: HDD, SSD, DRAM, SRAM, and SCM technologies were studied and compared. The strengths, weaknesses, opportunities, and threats of these technologies were identified.

To achieve the second research objective and its associated research question; the PZRAM a general-purpose RAMDISK was proposed. The proposed PZRAM is based on ZRAM and TMPFS.

To achieve the third research objective and its associated research question; the proposed method was evaluated by running several experiments with different environmental criteria.

5.3 Research Significance

The most important part of this research is that all requirements are available in the systems. The only thing needed is to exploit and optimize them in order to achieve superior performance improvements. A PM emulator was created using a ZRAM compressed RAM-block drive with a RAM filesystem on top of it. Further, comparisons were made on the PMEM emulator with Ext4, XFS, and PMFS filesystems to check which filesystem with PMEM can reach the best performance. The same scenarios were done for the creation of ZRAM, which was tested with Ext4, XFS, F2FS, Btrfs, RAMFS,

and TMPFS.

For persistent RAMDISK, tests were conducted on the RAMDISK with TMPFS, RAMFS, RAMFS /dev/shm, and TMPFS /dev/shm. Another significance of this research besides storage device and filesystem, the data file of an application, is that tests were run according to moving data file separately to other locations such as log files, system tablespace, and whole data directory.

This research provides a good assessment to users, researchers, and administrators as it can help them to understand the role of filesystems and moved data files on system throughput performance. Further, it also helped in reviewing and analyzing the different data placement devices and their role in database management systems.

5.4 Research Limitations

- As this research's-built PM emulator is based on dynamic random-access memory (DRAM), thus more memory will be consumed.
- The size of created PM emulator is limited by the memory capacity.
- Another limitation and strength at the same time are that there were no modifications done on the MYSQL source code, which could have helped in creating awareness of SCM/PM byte addressability features.
- This research only focussed on two performance metrics namely: throughput and response time.

5.5 Future Work

There is no perfect work, especially in the research domain. Each research completes the other. Thus, this dissertation focuses on only two metrics of system performance namely the throughput and response time. In the upcoming works, the researcher aims at measuring query execution performance and recovery performance. For the setup environment, future research would probably run experiments on a real SCM device to

check if its performance is gained by emulators. Furthermore, in the proposed method of the current study, there were no modifications to the source code of MYSQL. Thus, future work might consider modifying the source code to make an MYSQL InnoDB aware of persisting memory by creating an SCM- MYSQL storage engine and comparing its performance with the non-modified version.

For improving the application performance, future work plans to move forward to a cache memory level since it's faster than DRAM and the other lower memory hierarchy levels. Regarding this, future work is aimed at creating a partition of cache memory and flushing the hottest data of the desired application into that partition, and finally locking the created partition.

REFERENCES

Anthony, A. (2016). Memcached, Redis, and Aerospike Key-Value Stores Empirical Comparison, 1–8.

Arulraj, J., Levandoski, J., Minhas, U. F., & Larson, P.-A. (2018). Bztree: a high-performance latch-free range index for non-volatile memory. *Proceedings of the VLDB Endowment*, *11*(5), 553–565. https://doi.org/10.1145/3187009.3164147

Arulraj, J., & Pavlo, A. (2017a). How to build a non-volatile memory database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vol. Part F1277, pp. 1753–1758). https://doi.org/10.1145/3035918.3054780

Arulraj, J., & Pavlo, A. (2017b). How to build a non-volatile memory database management system. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, *Part F1277*, 1753–1758. https://doi.org/10.1145/3035918.3054780

Arulraj, J., Pavlo, A., & Dulloor, S. R. (2015). Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vol. 2015-May, pp. 707–722). https://doi.org/10.1145/2723372.2749441

Arulraj, J., Pavlo, A., & Malladi, K. T. (2019). Multi-tier buffer management and storage system design for non-volatile memory. *ArXiv*.

Arulraj, J., Perron, M., & Pavlo, A. (2016). Write-Behind Logging. *Proceedings of the VLDB Endowment*, *10*(4), 337–348. https://doi.org/10.14778/3025111.3025116

Bahn, H., & Cho, K. (2020). Implications of NVM based storage on memory subsystem management. *Applied Sciences (Switzerland)*, *10*(3), 1–18. https://doi.org/10.3390/app10030999

Bez, R., & Pirovano, A. (2004). Non-volatile memory technologies: Emerging concepts and new materials. In *Materials Science in Semiconductor Processing* (Vol. 7, pp. 349–355). https://doi.org/10.1016/j.mssp.2004.09.127

Bhimani, J., Yang, J., Yang, Z., Mi, N., Xu, Q., Awasthi, M., … Balakrishnan, V. (2017). Understanding performance of I/O intensive containerized applications for NVMe SSDs. In *2016 IEEE 35th International Performance Computing and Communications Conference, IPCCC 2016*. https://doi.org/10.1109/PCCC.2016.7820650

Btrfs Contributors at kernel.org. (2016). kernel.org. Retrieved from https://btrfs.wiki.kernel.org/index.php/Contributors

Calderoni, A., Sills, S., Cardon, C., Faraoni, E., & Ramaswamy, N. (2015). Microelectronic Engineering Engineering ReRAM for high-density applications. *MICROELECTRONIC ENGINEERING*, (April), 1–6. https://doi.org/10.1016/j.mee.2015.04.044

Canim, M., Mihaila, G. A., Bhattacharjee, B., Ross, K. A., & Lang, C. A. (2010). SSD bufferpool extensions for database systems. *Proceedings of the VLDB Endowment*, *3*(2), 1435–1446. https://doi.org/10.14778/1920841.1921017

Chatzistergiou, A., Cintra, M., & Viglas, S. D. (2015). REWIND: Recovery Write-Ahead System for In-Memory Non-Volatile Data-Structures. *Vldb '15*, 497–508. https://doi.org/10.14778/2735479.2735483

Chen, J., Jindel, S., Walzer, R., Sen, R., Jimsheleishvilli, N., & Andrews, M. (2015). The MemSQL query optimizer: A modern optimizer for real-time analytics in a distributed database. In *Proceedings of the VLDB Endowment* (Vol. 9, pp. 1401–1412). https://doi.org/10.14778/3007263.3007277

Corporation, I. (2016). How to emulate Persistent Memory. Retrieved from https://pmem.io/2016/02/22/pm-emulation.html

DeBrabant, J., Arulraj, J., Pavlo, A., Stonebraker, M., Zdonik, S. B., & Dulloor, S. (2014a). A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. *Amds@Vldb*, 57–63. Retrieved from http://dblp.uni-trier.de/db/conf/vldb/adms2014.html#DeBrabantAPSZD14

DeBrabant, J., Arulraj, J., Pavlo, A., Stonebraker, M., Zdonik, S. B., & Dulloor, S. (2014b). A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. *Amds@Vldb*, 57–63. Retrieved from http://dblp.uni-trier.de/db/conf/vldb/adms2014.html#DeBrabantAPSZD14

Desireddy, S., & Pathireddy, D. R. (2016). Optimize In-kernel swap memory by avoiding duplicate swap out pages. In *International Conference on Microelectronics, Computing and Communication, MicroCom 2016* (pp. 2–5). https://doi.org/10.1109/MicroCom.2016.7522551

Diaconu, C., Freedman, C., Ismert, E., Larson, P.-åke, Mittal, P., Stonecipher, R., … Zwilling, M. (2013). Hekaton : SQL Server ' s Memory -Optimized OLTP Engine.

Do, J., Zhang, D., Patel, J. M., DeWitt, D. J., Naughton, J. F., & Halverson, A. (2011). Turbocharging DBMS buffer pool using SSDs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 1113–1124). https://doi.org/10.1145/1989323.1989442

Dulloor, S. R. (2016). Systems and applications for persistent memory. *Thesis*, (December 2015). Retrieved from https://smartech.gatech.edu/handle/1853/54396%5Cnpapers3://publication/uuid/9A636FA2-795A-4C6C-AE4D-66246BA778EE

Dulloor, S. R., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R., & Jackson, J. (2014). System software for persistent memory. *Proceedings of the 9th European Conference on Computer Systems, EuroSys 2014*. https://doi.org/10.1145/2592798.2592814

Eisenman, A., Naumov, M., Gardner, D., Smelyanskiy, M., Pupyrev, S., Hazelwood, K., … Katti, S. (2018). Bandana: Using non-volatile memory for storing deep learning models. *ArXiv*.

Faerber, F., Kemper, A., Larson, P. Å., Levandoski, J., Neumann, T., & Pavlo, A. (2017). Main memory database systems. *Foundations and Trends in Databases*. https://doi.org/10.1561/1900000058

Fang, R., Hsiao, H. I., He, B., Mohan, C., & Wang, Y. (2011). High performance database logging using storage class memory. *Proceedings - International Conference on Data Engineering*, 1221–1231. https://doi.org/10.1109/ICDE.2011.5767918

Gaonkar, P. E., Bojewar, S., & Das, J. A. (2013). A Survey : Data Storage Technologies, *2*(2), 547–554.

Giles, E., Doshi, K., & Varman, P. (2013). Bridging the programming gap between persistent and volatile memory using WrAP. *Proceedings of the ACM International Conference on Computing Frontiers, CF 2013*. https://doi.org/10.1145/2482767.2482806

Götze, P., van Renen, A., Lersch, L., Leis, V., & Oukid, I. (2018). Data Management on Non-Volatile Memory: A Perspective. *Datenbank-Spektrum*, *18*(3), 171–182. https://doi.org/10.1007/s13222-018-0301-1

Gruenbacher, A., & Arnold, S. (2007). AppArmor Technical Documentation.

He, S., Sun, X. H., & Feng, B. (2014). S4D-cache: Smart selective SSD cache for Parallel I/O systems. *Proceedings - International Conference on Distributed Computing Systems*, 514–523. https://doi.org/10.1109/ICDCS.2014.59

Huang, J., Schwan, K., & Qureshi, M. K. (2014a). NVRAM-aware logging in transaction systems. *Proceedings of the VLDB Endowment*, *8*(4), 389–400. https://doi.org/10.14778/2735496.2735502

Huang, J., Schwan, K., & Qureshi, M. K. (2014b). NVRAM-aware logging in transaction systems. *Proceedings of the VLDB Endowment*, *8*(4), 389–400. https://doi.org/10.14778/2735496.2735502

Ilyas, M. U., Ahmad, M., & Saleem, S. (2020). Internet-of-Things-Infrastructure-as-a-Service: The democratization of access to public Internet-of-Things Infrastructure. *International Journal of Communication Systems*, *33*(16), 1–15. https://doi.org/10.1002/dac.4562

Imamura, S. (n.d.). Evaluating a Trade-Off between DRAM and Persistent Memory for Persistent-Data Placement on Hybrid Main Memory.

Jackson, A., Johnson, N., & Parsons, M. (2018). Exploiting the Performance Benefits of Storage Class Memory for HPC and HPDA Workflows. *Supercomputing Frontiers and Innovations*, *5*(1), 79–94. https://doi.org/10.14529/jsfi180105

Jose, J., Subramoni, H., Luo, M., Zhang, M., Huang, J., Wasi-Ur-Rahman, M., … Panda, D. K. (2011). Memcached design on high performance RDMA capable interconnects. In *Proceedings of the International Conference on Parallel Processing* (pp. 743–752). https://doi.org/10.1109/ICPP.2011.37

Joshi, A., Nagarajan, V., Viglas, S., & Cintra, M. (2017). ATOM: Atomic Durability in Non-volatile Memory through Hardware Logging. *Proceedings - International Symposium on High-Performance Computer Architecture*, 361–372. https://doi.org/10.1109/HPCA.2017.50

Kabakus, A. T., & Kara, R. (2017). A performance evaluation of in-memory databases. *Journal of King Saud University - Computer and Information Sciences*, *29*(4), 520–525. https://doi.org/10.1016/j.jksuci.2016.06.007

Kang, W.-H., Lee, S.-W., Moon, B., Kee, Y.-S., & Oh, M. (2014). Durable write cache in flash memory SSD for relational and NoSQL databases. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data - SIGMOD '14*, 529–540. https://doi.org/10.1145/2588555.2595632

Kang, W.-H., Yun, G.-T., Lim, S.-P., Shin, D.-I., Park, Y.-H., Lee, S.-W., & Moon, B. (2012). InnoDB DoubleWrite Buffer as Read Cache using SSDs. *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, 1–2.

Kim, H., Agrawal, N., & Ungureanu, C. (2012). Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory. *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, *8*(4), 1–25. https://doi.org/10.1145/2385603.2385607

Kim, S. (2012). Resistive RAM (ReRAM) Technology for High Density Memory

Applications. *4th Workshop Innovative Memory Technol MINATEC 2012; June 21-24 2012*, 4.

Kim, W. H., Kim, J., Baek, W., Nam, B., & Won, Y. (2016). NVWAL: Exploiting NVRAM in write-ahead logging. In *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* (Vol. 02-06-Apri, pp. 385–398). https://doi.org/10.1145/2872362.2872392

Kimura, H. (2015). Foedus: Oltp engine for a thousand cores and NVRAM. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vol. 2015-May, pp. 691–706). https://doi.org/10.1145/2723372.2746480

Kolli, A., Pelley, S., Saidi, A., Chen, P. M., & Wenisch, T. F. (2016). High-performance transactions for persistent memories. *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, *02-06-Apri*, 399–411. https://doi.org/10.1145/2872362.2872381

Kuznetsov, S. (2019). Towards a native architecture of In-NVM DBMS. In *Proceedings - 2019 Actual Problems of Systems and Software Engineering, APSSE 2019* (pp. 77–89). IEEE. https://doi.org/10.1109/APSSE47353.2019.00017

Lahiri, T., Neimat, M.-A., & Folkman, S. (2013). Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE Data Eng. Bull*, *36*(2), 6–13.

Lee, B. C., Ipek, E., Mutlu, O., & Burger, D. (2009). Architecting phase change memory as a scalable dram alternative. *ACM SIGARCH Computer Architecture News*, *37*(3), 2–13. https://doi.org/10.1145/1555815.1555758

Lee, H., & Lee, S.-W. (2016). Performance Improvement Plan for MySQL Insert Buffer. https://doi.org/10.1145/3007818.3007833

Leis, V., Haubenschild, M., Kemper, A., & Neumann, T. (2018). Leanstore: in-memory data management beyond main memory. *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*, 185–196. https://doi.org/10.1109/ICDE.2018.00026

Lemke, C., Radestock, G., Schulze, R., Thiel, C., Meghlan, A., Sharique, M., … Willhalm, T. (2017). SAP HANA adoption of non-volatile memory. In *Proceedings of the VLDB Endowment* (Vol. 10, pp. 1754–1765). https://doi.org/10.14778/3137765.3137780

Lersch, L., Hao, X., Oukid, I., Wang, T., & Willhalm, T. (2019). Evaluating persistent memory range indexes. *Proceedings of the VLDB Endowment*, *13*(4), 574–587. https://doi.org/10.14778/3372716.3372728

Li, Y., Liu, F., Xiao, N., Zeng, J., & Zhu, L. (2018). SNFS: Small Writes Optimization

for Log-Structured File System Based-on Non-Volatile Main Memory. In *Proceedings - 2017 IEEE 19th Intl Conference on High Performance Computing and Communications, HPCC 2017, 2017 IEEE 15th Intl Conference on Smart City, SmartCity 2017 and 2017 IEEE 3rd Intl Conference on Data Science and Systems, DSS 2017* (Vol. 2018-Janua, pp. 89–97). https://doi.org/10.1109/HPCC-SmartCity-DSS.2017.12

Lindström, J., Das, D., Mathiasen, T., Arteaga, D., & Talagala, N. (2015). NVM aware MariaDB database system. In *2015 IEEE Non-Volatile Memory Systems and Applications Symposium, NVMSA 2015*. https://doi.org/10.1109/NVMSA.2015.7304362

Liu, M., Zhang, M., Chen, K., Qian, X., Wu, Y., Zheng, W., & Ren, J. (2017). DudeTM: Building durable transactions with decoupling for persistent memory. *ACM SIGPLAN Notices*, *52*(4), 329–343. https://doi.org/10.1145/3037697.3037714

Ma, L., Arulraj, J., Zhao, S., Pavlo, A., Dulloor, S. R., Giardino, M. J., … Zdonik, S. (2016). Larger-than-memory data management on modern storage hardware for in-memory OLTP database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/2933349.2933358

Manual, M. R. (2013). MySQL 5 . 0 Reference Manual. *MySQL 5.0 Reference Manual*, *1*, 1692. Retrieved from dev.mysql.com

March, F., & Clara, S. (2017). WORT : Write Optimal Radix Tree for Persistent Memory Storage Systems This paper is included in the Proceedings of the 15th USENIX Conference on. *Fast*.

Meena, J. S., Sze, S. M., Chand, U., & Tseng, T. Y. (2014). Overview of emerging nonvolatile memory technologies. *Nanoscale Research Letters*, *9*(1), 1–33. https://doi.org/10.1186/1556-276X-9-526

Meza, J., Wu, Q., Kumar, S., & Mutlu, O. (2015). A Large-Scale Study of Flash Memory Failures in the Field (pp. 177–190). Association for Computing Machinery (ACM). https://doi.org/10.1145/2745844.2745848

Min, C., Kang, W.-H., & Kim, T. (2015). Lightweight Application-Level Crash Consistency on Transactional Flash Storage. *Usenix Atc'15*, 221–234. Retrieved from https://www.usenix.org/conference/atc15/technical-session/presentation/min

Moving MySQL databases to ramdisk on Ubuntu · GitHub. (n.d.). Retrieved March 20, 2020, from https://gist.github.com/kurisuchan/1262135/d7c2db7258b3cda96144027b346beb4f44fb8205

Mustafa, N. U., Armejach, A., Ozturk, O., Cristal, A., & Unsal, O. S. (2017a). Implications of non-volatile memory as primary storage for database management systems. *Proceedings - 2016 16th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2016*, 164–171. https://doi.org/10.1109/SAMOS.2016.7818344

Mustafa, N. U., Armejach, A., Ozturk, O., Cristal, A., & Unsal, O. S. (2017b). Implications of non-volatile memory as primary storage for database management systems. In *Proceedings - 2016 16th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2016* (pp. 164–171). https://doi.org/10.1109/SAMOS.2016.7818344

Natarajan, S. (2004). Emerging memory technologies. In *Microelectronics: Design, Technology, and Packaging* (Vol. 5274, p. 7). https://doi.org/10.1117/12.530385

Nguyen, T. D., & Lee, S. W. (2019). PB-NVM: A high performance partitioned buffer on NVDIMM. *Journal of Systems Architecture*, *97*(August 2018), 20–33. https://doi.org/10.1016/j.sysarc.2019.03.007

Oukid, I. (2019). Architectural principles for database systems on storage-class memory. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft Fur Informatik (GI)*, *P-289*, 477–486. https://doi.org/10.18420/btw2019-29

Oukid, I., Booss, D., Lehner, W., Bumbulis, P., & Willhalm, T. (2014). SOFORT: A hybrid SCM-DRAM storage engine for fast data recovery. *10th International Workshop on Data Management on New Hardware, DaMoN 2014 - In Conjunction with the ACM SIGMOD/PODS Conference*. https://doi.org/10.1145/2619228.2619236

Oukid, I., Booss, D., Lespinasse, A., Lehner, W., Willhalm, T., & Gomes, G. (2017). Memory management techniques for large-scale persistent-main-memory systems. *Proceedings of the VLDB Endowment*, *10*(11), 1166–1177. https://doi.org/10.14778/3137628.3137629

Oukid, I., Lasperas, J., Nica, A., Willhalm, T., & Lehner, W. (2016). FPTree: A hybrid SCM-DRAM persistent and concurrent B-Tree for Storage Class Memory. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, *26-June-20*, 371–386. https://doi.org/10.1145/2882903.2915251

Oukid, I., & Lersch, L. (2019). On the diversity of memory and storage technologies. *ArXiv*. https://doi.org/10.1007/s13222-018-0287-8

Ouyang, X., Nellans, D., Wipfel, R., Flynn, D., & Panda, D. K. (2011). Beyond block

I/O: Rethinking traditional storage primitives. *Proceedings - International Symposium on High-Performance Computer Architecture*, 301–311. https://doi.org/10.1109/HPCA.2011.5749738

Patterson, D. A., & Hennessy, J. L. (2019). Computer architecture: A Quantative Approach, 1–1527.

Peglar, R. (n.d.). What You can Do with NVDIMMs President , Advanced Computation and Storage LLC.

Petrov, I., Gottstein, R., & Hardock, S. (2015). DBMS on modern storage hardware. In *Proceedings - International Conference on Data Engineering* (Vol. 2015-May, pp. 1545–1548). https://doi.org/10.1109/ICDE.2015.7113423

Proctor, A. (2012). Non-volatile memory & its use in enterprise applications. *Viking Technology Understanding Non-Volatile Memory Technology Whitepaper*, (January), 1–8. Retrieved from https://scholar.google.fr/scholar?hl=en&q=Non-volatile+memory+%26+its+use+in+enterprise+applications&btnG=&as_sdt=1%2C5&as_sdtp=#0

Ramez Elmasri, S. B. N. (2011). *Fundamentals of Database Systems - 6th Edition*. *Addison Wesley* (Vol. 49). https://doi.org/10.1007/978-1-4842-0877-9_10

Rizk, R., Rizk, D., Kumar, A., & Bayoumi, M. (2019). Demystifying emerging nonvolatile memory technologies: Understanding advantages, challenges, trends, and novel applications. In *Proceedings - IEEE International Symposium on Circuits and Systems* (Vol. 2019-May). https://doi.org/10.1109/ISCAS.2019.8702390

Rizvi, S. S., & Chung, T. S. (2010). Flash memory SSD based DBMS for data warehouses and data marts. In *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010* (Vol. 3, pp. 557–559). https://doi.org/10.1109/ICCAE.2010.5451825

Schmidt, K., Ou, Y., & Härder, T. (2009). The promise of solid state disks: Increasing efficiency and reducing cost of DBMS processing. In *ACM International Conference Proceeding Series* (pp. 35–41). https://doi.org/10.1145/1557626.1557633

Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, J. D., Lentz, A., & Balling, D. J. (2008). *High Performance MySQL*. https://doi.org/10.1017/CBO9781107415324.004

Sehgal, P., Basu, S., Srinivasan, K., & Voruganti, K. (2015). An empirical study of file systems on NVM. In *IEEE Symposium on Mass Storage Systems and Technologies* (Vol. 2015-Augus). https://doi.org/10.1109/MSST.2015.7208283

Sha, E. H. M., Jiang, W., Dong, H., Ma, Z., Zhang, R., Chen, X., & Zhuge, Q. (2018). Towards the Design of Efficient and Consistent Index Structure with Minimal Write Activities for Non-Volatile Memory. *IEEE Transactions on Computers*, *67*(3), 432–448. https://doi.org/10.1109/TC.2017.2754381

Shahla Rizvi, S., & Chung, T. S. (2010). Flash memory SSD based DBMS for high performance computing embedded and multimedia systems. In *Proceedings, ICCES'2010 - 2010 International Conference on Computer Engineering and Systems* (pp. 183–188). https://doi.org/10.1109/ICCES.2010.5674850

Shu, J. W., Yu, B., & Yan, R. (2004). Design and Implementation of a non-volatile RAM disk in the SAN environment. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3252*, 203–212. https://doi.org/10.1007/978-3-540-30207-0_26

Smith, R. (2015). Intel Announces Optane Storage Brand For 3D XPoint Products. *AnandTech*. Retrieved from http://www.anandtech.com/show/9541/intel-announces-optane-storage-brand-for-3d-xpoint-products

Snyder, P. (1990). tmpfs: A virtual memory file system. *Proceedings of the Autumn 1990 EUUG Conference*, 241–248. Retrieved from http://wiki.deimos.fr/images/1/1e/Solaris_tmpfs.pdf

Son, Y., Kang, H., Yeom, H. Y., & Han, H. (2017a). A log-structured buffer for database systems using non-volatile memory. *Proceedings of the ACM Symposium on Applied Computing*, *Part F1280*, 880–886. https://doi.org/10.1145/3019612.3019675

Son, Y., Kang, H., Yeom, H. Y., & Han, H. (2017b). A log-structured buffer for database systems using non-volatile memory. *Proceedings of the Symposium on Applied Computing - SAC '17*, 880–886. https://doi.org/10.1145/3019612.3019675

Sorin, D. J. (2017). Persistent Memory. *Computer*, *50*(3), 12. https://doi.org/10.1109/MC.2017.67

Van Renen, A., Leis, V., Kemper, A., Neumann, T., Hashida, T., Oe, K., … Sato, M. (2018a). Managing non-volatile memory in database systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1541–1555. https://doi.org/10.1145/3183713.3196897

Van Renen, A., Leis, V., Kemper, A., Neumann, T., Hashida, T., Oe, K., … Sato, M. (2018b). Managing non-volatile memory in database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 1541–1555). https://doi.org/10.1145/3183713.3196897

Van Renen, A., Vogel, L., Leis, V., Neumann, T., & Kemper, A. (2019). Persistent memory I/O primitives. *ArXiv*.

van Vugt, S. (2015). *Beginning the Linux Command Line*. *Beginning the Linux Command Line*. https://doi.org/10.1007/978-1-4302-6829-1

Volos, H., Magalhaes, G., Cherkasova, L., & Li, J. (2015). Quartz: A lightweight performance emulator for persistent memory software. In *Middleware 2015 - Proceedings of the 16th Annual Middleware Conference* (pp. 37–49). https://doi.org/10.1145/2814576.2814806

Wang, T., & Johnson, R. (2014). Scalable logging through emerging nonvolatile memory. *Proceedings of the VLDB Endowment*, *7*(10), 865–876. https://doi.org/10.14778/2732951.2732960

Wickberg, T., & Carothers, C. (2012). The RAMDISK storage accelerator - A method of accelerating I/O performance on HPC systems using RAMDISKs. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers, ROSS 2012 - In Conjunction with: ICS 2012*. https://doi.org/10.1145/2318916.2318922

XFS Overview. (2013). Silicon Graphics International Corp. Retrieved from http://oss.sgi.com/projects/xfs/index.html

Yu, S., & Chen, P. Y. (2016). Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Magazine*, *8*(2), 43–56. https://doi.org/10.1109/MSSC.2016.2546199

Zhang, Y., & Swanson, S. (2015). A study of application performance with non-volatile main memory. *IEEE Symposium on Mass Storage Systems and Technologies*, *2015-Augus*. https://doi.org/10.1109/MSST.2015.7208275

Zhao, J., Xu, C., Chi, P., & Xie, Y. (2015). Memory and storage system design with nonvolatile memory technologies. *IPSJ Transactions on System LSI Design Methodology*, *8*(February), 2–11. https://doi.org/10.2197/ipsjtsldm.8.2

Zhuang, Z., Zuk, S., Ramachandra, H., & Sridharan, B. (2016). Designing SSD-Friendly Applications for Better Application Performance and Higher IO Efficiency. *Proceedings - International Computer Software and Applications Conference*, *1*, 369–378. https://doi.org/10.1109/COMPSAC.2016.94