An in-depth Investigation Into the Application of Flash Memory In a Business Intelligence Database Environment

Cheikh Salmi[1,2], Nour El-Houda Senoussi[1,2], Djoumana Chaal[1] and Mohamed Boudjadi[1]

[1]Computer Science Department, M'hamed Bougara University, Boumèrdes, Algeria
[2]LIMOSE Laboratory

Abstract

Recent developments in solid-state drive (SSD) technology have significantly enhanced the access speed of these storage devices, surpassing traditional magnetic hard drives by orders of magnitude. This remarkable capability for swift data retrieval aligns seamlessly with On-Line Analytical Processing (OLAP) techniques, which heavily depend on read speeds rather than write speeds. In this paper, we analyze the performance of OLAP techniques on an SSD as compared with a normal hard drive using a trace-based approach. The advantage of this approach is that it (1) allows simulating any HDD, SSD and DBMS (2) allows the DBMS to treat the SSD as though it was a regular hard drive and (3) allows to observe the differences in execution times without altering any data structure or algorithms of the target DBMS. Although SSDs do not substantially improve write speeds, our experiments empirically indicate that their exceptional read speed makes them the prime candidate for data storage in read-oriented database environment.

Keywords

Data warehouse, flash sim, IO simulation, olap.

1. Introduction

Online Analytical Processing OLAP [1] is a set of techniques that often involves scanning large datasets applying aggregations at certain granularity levels and writing summary information back to the database. In addition, OLAP can also involve the retrieval of specific groups from a large precomputed repository of data. For an OLAP user, performance is often the most important and sought-after quality. Like many other techniques in database management systems, the effectiveness of OLAP relies fundamentally on the speed of memory, CPU capabilities, and hard drive speeds[2]. In nearly all instances, it's the hard drives that constitute the limiting factor in terms of performance. In the ever-changing landscape of data storage, hard disk drives (HDDs) have long been the workhorses, serving faithfully as the primary storage medium for all types of data and applications and particularly for DBMSs and their operational data . As data volume grows annually and new data types (semi-structured, multimedia, graph, etc.) consistently emerge, the need for cutting-edge technologies capable of efficiently managing this extensive and diverse data becomes paramount[3, 4, 5, 6, 7, 8]. Addressing the evolving challenges posed by the expanding and varied nature of contemporary data mandates a shift beyond the limitations of HDDs. Additionally, as storage technology advances, it becomes imperative to carefully examine the limitations of hard drives and, therefore, advocate for the adoption of new media such as Solid State Drives (SSDs)[9, 10].

One of the most significant drawbacks of HDDs is their mechanical nature. Unlike SSDs, which use flash memory for data storage, HDDs rely on spinning disks and a moving read/write head. This mechanical structure introduces latency, leading to slower data access times and increased susceptibility to wear and tear[11, 12, 13]. The constant movement of parts within an HDD not only hinders overall performance but also makes them more prone to failure. Another notable disadvantage is the fragility of HDDs. The delicate nature of the internal components means that HDDs are susceptible to damage from physical shocks and vibrations[14, 15, 16]. This vulnerability can result in data loss or, in severe cases, render the entire drive inoperable. In contrast, SSDs, being devoid of moving parts, are inherently more robust and better equipped to withstand shocks, making them a more reliable choice for data storage. Energy efficiency is also a significant concern when comparing HDDs to SSDs. HDDs consume more power due to the continuous spinning of disks and the movement of mechanical parts. In environments where energy efficiency is a priority, such as in laptops or data centers, SSDs stand out as a more energy-efficient alternative. Their lower power consumption not only contributes to a greener computing environment but also translates to longer battery life in portable devices. SSDs are based on flash memory which is a widely used medium in embedded systems (e.g. PDAs,

SYSTEM 2025: 11th Sapienza Yearly Symposium of Technology, Engineering and Mathematics. Rome, June 4-6, 2025
c.salmi@univ-boumerdes.dz (C. Salmi);
n.senoussi@univ-boumerdes.dz (N. E. Senoussi);
djoumana.chaal@gmail.com (D. Chaal);
boudjadi.mlohammed@gmail.com (M. Boudjadi)
0000-0001-7131-6158 (C. Salmi); 0009-0009-4906-9686 (N. E. Senoussi)

mobile phones, etc.) and promises to replace the magnetic hard drive for secondary storage system in all IT devices recent and data centers.

More specifically, the Solid State Drive (SSD) is a data storage device that uses NAND flash memory and can be used in the same way as an HDD disk via IDE, SATA or SCSI interfaces or more recently with NVMe interfaces that are four times faster than SATA.

OLAP is a technique for multidimensional analysis, which allows decision-makers to have quick and interactive access to relevant information presented from various and multiple angles, according to their particular needs. OLAP is therefore a technique whose functionalities are used to facilitate multidimensional analysis: operations that can be carried out on the hypercube to extract data. Moreover, OLAP processing necessitates fewer write operations compared to other database applications like operational OLTP databases (online transaction processing) . This characteristic makes OLAP a suitable candidate for execution on SSD-type media, the speed of which has consistently improved since their inception. The aim of this work is to study the suitability of flash memories and SSD disks to data warehouse technology. The paper is organized as follows. In Section 2, we review some related works related to the integration of flash memory in Database Management Systems (DBMS). In Section 3 we present the fundamental concepts of flash memory and SSD drive. In section 4 we present the data warehouses and the OLAP technique which constitutes the context of our work. In Section 5, we present our proposal, emphasizing the hybridization of HDD and SSD media. Section 6 is dedicated to implementation and experimentation. Within this section, we introduce the flash memory simulator utilized, provide details about the environments, and present the results of all conducted simulations. Section 7 concludes our paper.

2. Related Work

Ever since the authors of [17] announced that 'tape is dead, disk is tape, and flash is disk', there has been a lot of research that has attempted to study the use of flash memory in conjunction with databases. Many techniques have been studied such as query processing [18, 19, 20] and page layout [21]. intense work has been done to extend the buffer with flash memory. The work of [22] propose to use SSDs as an extension to database buffer pool and exploit them to manage the dirty pages. Before being evicted from the buffer, the authors propose to first store the dirty pages on the SSD to avoid the communication with the HDD[23, 24]. Three eviction strategies were proposed. Clean-write: never write dirty pages to SSD, dual-write: write an evicted dirty page to both HDD and SSD and finally, lazy-cleaning in which dirty pages are first written to SSD and later copied from the SSD to HDD (lazy updates). Authors in [25] focused on flash memory as a write cache for databases stored on conventional hard disk. In this case, when dirty pages are evicted from the memory buffer pool, they are first written on the flash-based-cache and later propagated to hard disk, applying some replacement strategies to reduce the amount of data written to flash. The issue of SDD buffer in the OLTP environment has been examined in [26], with a significant focus on enhancing recovery time and ensuring data integrity following a crash or a routine database restart[27, 28, 29]. Metadata about the contents of the SSD buffer are stored on the so called: SSD buffer table. This table can be reconstructed using transactional log files and is periodically flushed in an asynchronous mode. A caching algorithm at the granularity of subtuples is proposed in [30]. The algorithm partitions vertically a database table and then the subtuples are cached. Updates are gathered and when a page eviction occurs, subtuples are flushed to flash memory in the same area (a page or more) which reduces the amount of data written to flash memory. In the paper [31], authors introduced a caching system that utilizes flash memory as an intermediary layer positioned between the main memory buffer pool and the magnetic disk. A theoretical cost model and a strategy that decides which data to be cached were also proposed. Despite their importance in rapid decision-making and their differences from classical relational databases, data warehouses and OLAP queries have not been studied much, especially in the context of recent storage media such as flash memory. The authors in [32] have proposed an approach whose principle is to generate small lattices by reducing the redundancy of prefixes and suffixes to manage condensed cubes. The authors of [5] have conducted an experimental investigation aimed at enhancing the performance of OLAP cube processing on SSDs. This study utilized specific types of storage disks, specifically a Seagate 160 GB magnetic disk and a Patriote 64 GB Solid State Drive[33].

3. Solid State Drive

A solid-state drive (SSD) is a permanent storage medium that uses flash memory-based chips for data storage. Unlike the traditional hard disk drive (HDD), an SSD lacks moving mechanical parts and consists of an array of flash memory cells that depend on MOS transistors (Metal Oxide Semiconductor, e.g., USB flash drive). These cells can trap electrical current to encode the binary digits '0' or '1'. Consequently, this memory type features three primary input/output operations: read, write, and erase. As previously mentioned, SSDs offer numerous advantages over HDDs, including greater efficiency, enhanced relia-

bility, lower energy consumption, and silent operation. While their prices have historically been higher, they are gradually decreasing.

3.1. Flash Memory

There are two types of flash memory (1) NOR flash: this type of memory allows fast data addressing. Conversely, write and erase times are long. These memories are used to store code intended to be executed in place without copying the code into the main memory (XIP, for Execute In Place). They are generally used to host the operating systems (OS) of various embedded systems such as smartphones, etc. (2) NAND flash : constitutes the foundation of most external mass storage devices due to their high density and affordable price.

3.2. NAND Flash Memory

Internally, NAND flash memory comprises two primary components: (1) the flash controller, a hardware component responsible for supervising the interface between the flash device and the host system, as well as handling received I/O requests; (2) the flash chips, which serve as the actual data storage units and are interconnected by a bus. Figure 1 illustrates the architecture of a NAND type flash memory. In the controller there is an SRAM, used in particular to store information relating to the flash translation layer (FTL) whose main role is to maximize the lifetime of the memory which depends on the number of write/erase cycles. There are two major classes of NAND memories: Single Level Cell (SLC) and Multi Level Cell (MLC). SLC memory cells are capable of storing 1 bit, and MLCs can store 2 or more (triple, quad LC store three and four bits per cell respectively) [34, 35]. SLCs have a lifetime 10 times longer than MLCs [36, 37, 38]. From a logical point of view, the data is organized in logical units (LU), erase blocks and pages. The LUs called data banks are matrices of blocks. Erase blocks are arrays of pages. Finally, pages represent the smallest addressable units for read and write commands. A distinction is made in a page between the area that contains the data stored on the flash memory (user data) and an Out Of Band Area (OOB), used to store meta-data on the page itself and the data contained therein. It should be noted that the OOB data is used by the FTL for its own operation[39]. As mentioned before, flash memory supports three operations: read (read), write (or program), and erase (data deletion). Reads and writes apply at page level, erasing applies to an entire block. Similar to other EEPROM devices, clearing individual bits can only be achieved by erasing a significant memory block. Consequently, a block can only endure a finite number of erasures, beyond which its capacity to reliably store data diminishes[40]. It is important to note that modifying the

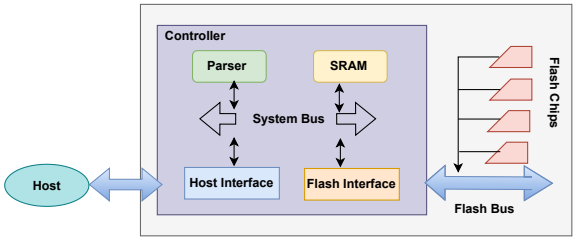

Figure 1: NAND flash memory architecture.

content of a page (update) requires: (1) copy the entire block that contains the page to a different location (free block, previously erased), while taking into account the modifications requested by the write request; (2) invalidate the old obsolete block to be recycled soon; (3) finally update the addressing tables to indicate that the new block contains the latest version of the data. The lifetime also called endurance (number of write/erase cycles) is one of the main constraints of flash memories. Currently, a 100,000 P/E Cycles is reached [1] but the problem of block wear still remains.

3.3. Flash Translation Layer

A straightforward method to connect flash memory to a host system is by treating it as a hard disk and assigning its management to the standard file system. However, as these file systems are not specifically optimized for flash memory, one may notice a swift deterioration of the memory blocks. This problem is compounded by the erase-before-write constraint mentioned earlier [41]. It is therefore essential to minimize and distribute block erasing in a way that maximizes the average lifetime of each cell constituting the flash memory. This is the purpose of a technique called wear leveling. Wear leveling is integrated, among other features, into flash memory in two different ways: (1) by using a dedicated file system: Flash File System (FFS), such as UBIFS (Unsorted Block Images File System) [42], YAFFS, YAFFS2 (Yet Another Flash File System) [43], or JFFS (Journaling Flash File System) [44], etc.; (2) by implementing a translation layer between the system and the flash memory: the FTL. The main advantage of FTL is that it works with standard file systems. The FTL roles comprises an address mapping (translate host system requests to physical addresses), bad block management, power-off recovery, wear Levelling and a garbage collection functions [45, 46].

[1] https://www.apacer.com/en/News/Detail/2022-apacer-3d-slc-lite-x-en

3.3.1. Address translation

The FTL's address translation process is based on mapping tables stored in the SRAM. Each entry of the table maps a logical address to a unique physical address. There are mainly three basic address translation schemes, each has its advantages and drawbacks: page mapping, block mapping, and hybrid mapping [47, 41, 45]. Page mapping is an intuitive technique that consists of directly associating a physical page with each logical page. It requires a large mapping table since an entry is needed for each page of flash memory. During a write request, if the page is not empty, the FTL chooses a free physical page, and updates the mapping table. Upon receipt of an I/O request from the host system addressing a page, the FTL first calculates the corresponding logical block number (LBN) according to the following formulas:

$$LBN = \frac{LPN}{NPPB} \tag{1}$$

$$OFFSET = LPN\%NPPB \tag{2}$$

The first equation determines the address of the physical block by dividing the logical page address (LPN) by the number of pages contained in a block (NPPB). Then, using the mapping table, the FTL determines the corresponding physical block. The second equation determines the offset between the first page of the physical block (page 0) and the page addressed for reading or writing. The disadvantage of block mapping is the generation of extra operations in case of write requests compared to page mapping. Indeed, if a write operation requires an update of only a few pages of the block, the entire block must be re-mapped to another free physical block. Hybrid-mapping is an approach designed to overcome the shortcoming of the two previous techniques. Other works have been based on this method to improve mapping performance. In their work, J. Kim et al. [48] introduce a technique that employs a hybrid approach, combining block-level and page-level (coarse-grain and fine-grain) methods. This log block scheme optimizes the SRAM map size and enhances performance for both small and large write operations. Additionally, C. Park et al. [45] propose a flexible group mapping method, building upon the previously presented log block scheme. This method allows for the configuration of the degree of log block sharing among different groups [49]. For the sake of space, we settle for only describing FTLs used in our simulations. Other works on FTLs can be found in [41, 50, 51, 14].

3.3.2. Flash Memory Simulation

Flash memory-based real disks are considered closed systems, which means they have limited accessibility to users. Often, manufacturers do not provide detailed technical specifications of the disk, making it challenging to customize or optimize its internal parameters. However, simulation offers a convenient solution to this limitation by allowing users to manipulate the internal parameters of the disk and conduct desired tests effectively. The common methods are (1) Trace-Based Simulation: it uses real-world workload traces of read and write operations to drive the simulation. This method allows a realistic representation of actual workload behavior, allowing for accurate evaluation of system performance under real-world conditions. However, it requires access to high-quality, representative traces, which might not be readily available. (2) Analytical Modeling: formulates mathematical models based on the fundamental characteristics of flash memory operations and system architecture. It provides theoretical insights into the system's behavior and performance under various scenarios. This method is fast and computationally efficient but the simplifications in the model may lead to less accurate results compared to more complex simulations. (3) Full-System Simulation: simulates the entire flash memory system, including internal NAND flash operations, wear leveling, garbage collection, etc. It allows a comprehensive and accurate representation of the flash memory system's behavior, enabling detailed analysis and optimization. However, this method induces high computational overhead and is time-consuming in addition to the thorough understanding of flash memory internals for accurate modeling. (3) FPGA-based Simulation: utilizes Field-Programmable Gate Arrays (FPGAs) to implement flash memory operations at hardware-level speeds. It allows high performance and real-time simulation capabilities, enabling hardware-software co-design and validation. This method requires FPGA expertise and hardware resources, making it less accessible to some researchers. (4) Hardware Emulation: uses specialized hardware (e.g., FPGA-based emulators) to mimic the behavior of real flash memory. It allows real-time and cycle-accurate simulation, resulting in detailed analysis of system behavior and debugging. The main drawback of this approach is its higher cost and reduced flexibility compared to software-based simulations.

4. Data Warehouse and OLAP

A data warehouse (DW) is explicitly crafted for analyzing data, entailing the retrieval of substantial data quantities to comprehend the relationships and trends within it. Throughout the remainder of this study, we will demonstrate through our experiments that data storage is more efficiently handled by storage medium utilizing flash memory. In the following section, we present some data warehouse fundamental concepts.

4.1. OLAP's Role in Data Analysis and Decision-Making

A data warehouse is a database dedicated to online analysis for decision support. OLAP operators allow business decision makers to extract data cubes corresponding to analytical contexts [1]. Data cubes are data structures that allow DW data to be grouped according to several business functions. Each cube contains the relevant data for a particular function. Data is consolidated, aggregated and optimized for fast and efficient analysis. The cube enables drilling down, slicing, dicing, or pivoting data to observe it from various perspectives. OLAP cubes are optimized for read access, generating reports for decision making is much faster than with transactional systems (OLTP).

4.1.1. Online Analytical Processing (OLAP)

OLAP is a technology used to organize, analyze, and process large volumes of multidimensional data. Unlike Online Transaction Processing (OLTP), which focuses on managing day-to-day operational data, OLAP deals with complex queries and data analysis tasks. At the core of OLAP lies the fundamental concept of conducting multidimensional analysis, empowering users to glean insights from their data through various viewpoints, commonly known as "slicing and dicing".

4.1.2. Key Aspects of OLAP

OLAP technology is characterized by: (1) Multidimensional Data Model: OLAP databases use a multidimensional data model, where data is organized into dimensions (e.g., time, product and region) and measures (e.g., sales and revenue) (2) Dimension Hierarchies: Each dimension typically contains hierarchies, allowing users to drill down or roll up the data to various levels of granularity (3) Aggregation: OLAP allows for pre-aggregation of data to accelerate query processing and improve performance.

4.1.3. Importance in Data Analysis and Decision-Making

OLAP plays a vital role in data analysis and decision-making processes for several reasons: (1) Complex Data Exploration: OLAP enables users to explore complex datasets efficiently. Decision-makers can quickly access and analyze large volumes of data from various angles, enabling better understanding and insight into business trends and patterns (2) Interactive and Ad-Hoc Analysis: OLAP systems allow users to perform ad-hoc queries interactively, providing real-time responses to complex analytical questions. This flexibility empowers users to investigate data anomalies and outliers, identify opportunities, and make data-driven decisions promptly (3) Business Intelligence and Reporting: OLAP is the foundation of Business Intelligence (BI) systems, providing essential tools for reporting, data visualization, and dashboards. It supports the creation of user-friendly reports and visualizations, making it easier for non-technical users to comprehend data and make informed decisions (4) Support for Decision-Making: OLAP's ability to present multidimensional data in a comprehensible format facilitates better decision-making at all levels of an organization. Decision-makers can evaluate performance, assess the impact of strategic decisions, and identify areas for improvement (5) Forecasting and Planning: OLAP systems support data forecasting and predictive analytics. By analyzing historical data and trends, organizations can make informed predictions and develop strategic plans for the future.

4.1.4. Rising Demand for Faster OLAP Data Processing

The demand for faster data processing in OLAP systems has grown exponentially due to several factors: (1) Data Volume and Complexity: With the increasing availability of big data and the proliferation of data sources, OLAP systems must handle larger and more complex datasets. Faster processing is necessary to deliver timely results for analysis (2) Real-Time Decision-Making: In today's fast-paced business environment, real-time decision-making is crucial. Decision-makers need instant access to up-to-date data and insights to respond to market changes and seize opportunities (3) Competitive Advantage: Faster data processing in OLAP systems provides a competitive edge. Organizations that can analyze data and make decisions faster can respond promptly to market trends and gain a competitive advantage (4) User Expectations: Users, including executives, managers, and analysts, expect near-instantaneous response times when interacting with OLAP systems. Slow query response times can lead to user frustration and hinder productivity and (5) Business Complexity: Businesses are becoming more complex, with data-driven strategies playing a significant role. Faster OLAP processing allows for more sophisticated data analysis, enabling organizations to uncover deeper insights. To meet these demands, OLAP systems are continuously evolving, incorporating technologies like in-memory computing, columnar databases, and hardware acceleration (e.g., GPUs) to enhance data processing speed and optimize performance. Faster OLAP systems are critical for empowering organizations to make informed decisions quickly and stay ahead in today's data-driven landscape.

5. Proposed Approach

In this section, we elaborate on our approach, which centers on conducting comprehensive tests and evaluations of a data warehousing and OLAP on various configurations, including Hard Disk Drives (HDD), Solid-State Drives (SSD), and hybrid configurations combining both technologies. Our approach aims to assess the performance of these storage options in the context of data warehousing. The motivation comes from the fact that: (1) HDDs are well-established, cost-effective storage devices known for their large storage capacities. However, they typically exhibit slower data access and retrieval times compared to SSDs (2) SSDs are relatively newer storage technology, renowned for their exceptional speed and low latency. They excel in rapid data retrieval and are ideal for high-performance computing tasks and (3) In addition to individually testing HDDs and SSDs, our approach extends to exploring hybrid solutions that leverage the strengths of both storage technologies. By combining HDDs for bulk storage and SSDs for caching frequently accessed data, we aim to strike a balance between capacity and performance. This hybrid approach is particularly interesting in scenarios where cost-effectiveness and performance optimization are paramount. In the following, we describe the fundamentals of the hybridization approach of the two storage media.

5.1. The Concept of Hybridization

Hybrid storage denotes storage solutions that combine solid-state drives (SSDs) and traditional mechanical hard drives, along with main memory (RAM), to provide an optimal balance between performance and cost-effectiveness. The basic idea behind hybridization is to store intermediate results shared by multiple queries on one of the RAM or SSD storage media. This technique is particularly suitable for data warehouses and flash memory systems because decision-making queries often involve a significant number of common operations. This phenomenon is known as multi-query optimization [52, 53, 54, 55]. Most of these studies optimize all queries within the same batch and generate a single execution plan for all queries. The advantageous intermediate results for the execution of the entire workload are selected and materialized on the secondary storage medium, which can be either the HDD, SSD or main memory. Intermediate query results encompass a series of materialized views utilized for the advance computation and storage of condensed aggregate data, such as the total sales amount. In this context, these materialized views are often termed "summaries" since they store condensed data. Furthermore, they can be used to perform pre-computations of joins in various sizes, both with and without aggregation operations. The use of a materialized view serves to eliminate the computational overhead linked to resource-intensive joins and aggregations for a wide range of queries.

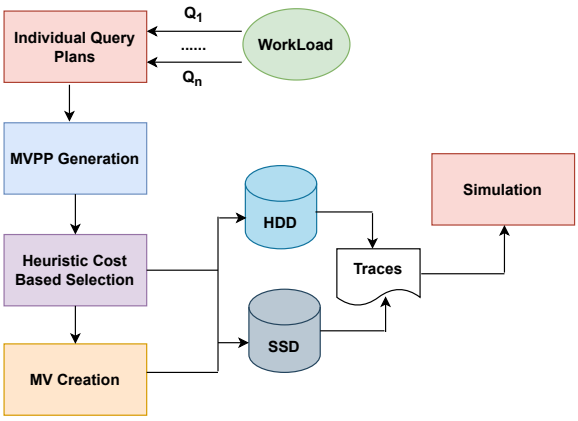

Figure 2: Hybrid approach.

The challenge in determining which intermediate results to materialize involves identifying the combination of node subsets that will be materialized to optimize the workload execution time. This task involves exploring a vast search space, encompassing all potential subexpressions within the database schema and the workload over a specific period. This problem has similarities to the well-known knapsack problem. In our context, the "knapsack" represents the HDD and SSD storage media, and the "objects" refer to the nodes to be placed in these storage media. It is worth noting that the knapsack problem is recognized as NP-complete [56, 57, 58].

As illustrated in Figure 2, the hybridization solution involves harvesting the most representative query workload through the logs of the Database Management System (DBMS). Subsequently, an execution plan is derived for each query. The collection of query execution plans is transformed into a structure called MVPP (Multiple View Processing Plan) [59], which provides an overarching plan highlighting query interactions and common nodes (to be materialized). Due to the extensive search space for nodes to materialize, a heuristic selection module is developed to choose the optimal configuration. The role of the heuristic selection module is to leverage all candidate execution plans and identify the one with the optimal cost by utilizing cost models tailored to each storage medium. For each candidate plan, its execution cost is computed using the cost model, and the plan with the minimal cost is chosen to evaluate the current query. Following this, the selected optimal nodes are created on the appropriate storage medium. The query workload is executed after the materialization of these nodes, and the execution traces are recorded for playback on

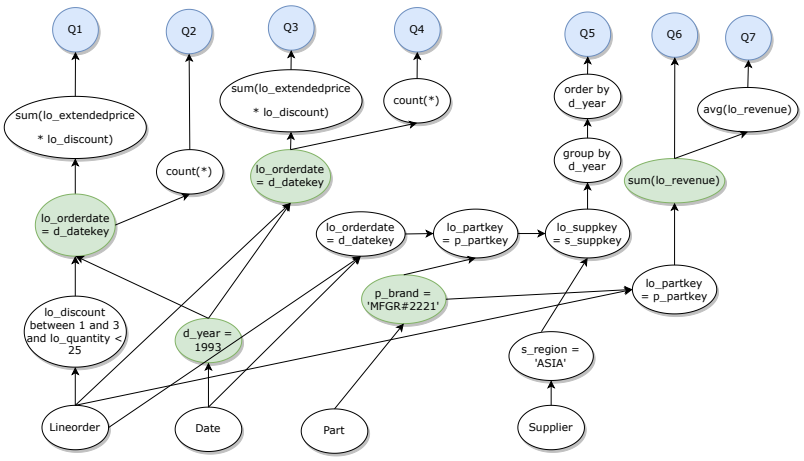

Figure 3: SSB MVPP example.

the simulated storage system. Figure 3 illustrates the integration of MVPP with the initial seven queries from the SSB benchmark. Table 1 outlines the nodes suitable for potential materialization. Each node is characterized by its frequency, size, and estimated calculation cost, as determined by the general cost model, regardless of the storage medium type.

Nodes 1 through 5 depict the full scan of the base tables: lineorder (L), date (D), customer (C), supplier (S), and part (P). It's important to highlight that the calculation cost of a node encompasses the costs associated with all its child nodes within the MVPP.

5.2. Formalization of the Hybridization Problem

The hybridization problem is formalized as follows: **(1)** Inputs: (i) a relational data warehouse RDW, (ii) A query workload $Q = \{q_1, q_2, ..., q_m\}$, each query q_i has an access frequency f_i, where $1 \leq i \leq m$. Queries are represented by an MVPP (Multi-Query View Plan), (iii) A set of intermediate result nodes $R = \{rn_1, rn_2, ..., rn_n\}$ from the MVPP, candidates for materialization on HDD or SSD. Each intermediate result can be assigned to at most one storage media (either HDD or SSD) **(2)** Constraints: HDD size S_{hdd}, SSD size S_{ssd} and **(3)** Outputs: an assignment that allows intermediate nodes to be allocated across the two storage media in order to optimize the overall cost of executing the input query workload. The objective is to minimize the total execution time by selecting the most efficient combination of intermediate result for each storage media.

The primary objective function within the hybridation problem is to minimize the weighted query processing cost, as defined by the formula 3.

$$\text{MIN(QPC)} = \sum_{i=1}^{m} f_i * cost_{q_i}^{(R)} \tag{3}$$

Where, $cost_{q_i}^{(R)}$ represents the processing cost associated with q_i, considering a set of intermediate nodes R. The dual problem revolves around maximizing the overall execution time gain by optimizing the benefit gained when particular intermediate nodes are stored on one of the storage media (HDD or SSD). Formally:

$$\text{MAX(GN)} = \sum_{k=1}^{2} \sum_{i=1}^{n} (G_i^k * x_i^k) \tag{4}$$

Subject to the following constraints:

$$\begin{cases} \sum_{i=1}^{n} (\|rn_i\| * x_i^k) <= SC_k, k = 1...2 \\ \sum_{k=1}^{2} (x_i^k) <= 1, i = 1...n \end{cases}$$

Where:

(i) G_i^k represents the cumulative gain when node rn_i is placed on medium k. The cumulative gain G_i^k when node rn_i is allocated on medium k is computed as follows:

$$G_i^k = \sum_{j=1}^{m} (f_{q_j} * Cost_{q_j}(rn_i))/nq_{si}$$

It represents the absolute value of I/O cost that the workload would accumulate if the node were stored on either the HDD or SSD, subtracting the cost incurred when recomputing the node. Here, f_{q_j} represents the frequency of query q_j, $Cost_{q_j}(rn_i)$ denotes the execution cost of query q_j accessing node rn_i, and nq_{si} represents the total number of queries sharing node rn_i.

Table 1
SSB potential nodes for materialization

ID	Description	#Occurence	Cost	Size(meg)
6	$\sigma_{dy=1993}(D)$	4	71,95	0,001460
7	$\sigma_{1\leq ld\leq 3 \wedge lq<25}(L)$	2	132274,76	4,479846
8	$\sigma_{sr='ASIA'}(S)$	9	58	0,001796
9	$\sigma_{pb='MFGR\#2221'}(P)$	7	4338,59	0,000468
10	$\sigma_{sr='EUROPE'}(S)$	3	58	0,001520
11	$\sigma_{cr='ASIA'}(C)$	3	651,82	0,072612
12	$\sigma_{\text{cn='UNITED STATES'}}(C)$	4	113520,15	45,01107
13	$\sigma_{\text{sn='UNITED STATES'}}(S)$	6	58	0,001140
14	$\sigma_{pmgfr='MFGR\#1'\vee pmgfr='MFGR\#2'}(P)$	4	4632,71	0,513948
15	$\sigma_{dy=1997\vee dy=1998}(D)$	6	78,34	0,004062
16	$\sigma_{sr='AMERICA'}(S)$	7	58	0,007560
17	$\sigma_{cr='AMERICA'}(C)$	3	650,63	0,023968
18	$\sigma_{1992\leq dy\wedge dy\leq 1997}(D)$	7	78,34	0,013152
19	$6 \bowtie 7$	2	133192,64	0,456950
20	$2 \bowtie 6$	2	164385,27	8,570180
21	$2 \bowtie 9$	3	18919,79	0,028096
22	$2 \bowtie 1$	4	227969559	51132,57
23	$2 \bowtie 11$	3	165012,32	18,15747
24	$2 \bowtie 8$	3	164379,63	29,64128
25	$2 \bowtie 12$	4	120932,94	1,785442
26	$2 \bowtie 13$	2	120157,12	1,615391
27	$2 \bowtie 17$	3	165010,39	19,17912
28	$2 \bowtie 5$	1	165416,86	204,0501
29	$2 \bowtie 16$	3	164378,75	31,75981
30	$9 \bowtie 22$	4	20516,89	0,11942
31	$8 \bowtie 30$	2	172688,99	43,11459
32	$30 \bowtie 10$	2	20097,7	0,009072
33	$21 \bowtie 10$	1	19998,54	0,049940
34	$23 \bowtie 8$	3	122598,28	3,623424
35	$34 \bowtie 18$	3	146912,11	7,457792
36	$24 \bowtie 18$	1	20440,12	0,174114
37	$25 \bowtie 13$	4	123212,14	0,268184
38	$37 \bowtie 18$	3	121370,58	0,102660
39	$27 \bowtie 16$	3	1320,3	0,000057
40	$39 \bowtie 15$	3	1320,6	0,000059
41	$40 \bowtie 14$	3	123749,59	0,114840
42	$28 \bowtie 16$	1	168488,65	64,65390
43	$45 \bowtie 15$	1	122951,02	7,385620
44	$43 \bowtie 14$	1	140176	8,202975
45	$15 \bowtie 29$	2	127334,04	3,75540
46	$21 \bowtie 8$	1	20119,48	0,07504

(ii) $\|rn_i\|$ represents the size of node rn_i.
(iii) SC_k is the storage capacity of storage medium k (S_{hdd} for $k = 1$ and S_{ssd} for $k = 2$).
(iv) The decision variable x_i^k is defined as follows:

$$x_i^k = \begin{cases} 1 & \text{if } rn_i \text{ is stored on the storage media } S_k \\ 0 & \text{else} \end{cases}$$

5.3. Resolution Approach

To emphasize the impact of incorporating SSD disks into an OLAP environment with the integration of materialized views, we focus solely on scenarios where the views are stored on either SSD or HDD media. In general, the problem of selecting materialized views is classified as an NP-hard problem, and its resolution typically relies on heuristic approaches. We utilize the simulated annealing algorithm [12] for selecting nodes to be materialized on the SSD disk. The core principle of the simulated annealing algorithm is derived from simulating the annealing process employed in the heat treatment of metals. The key concept is to traverse the solution space by considering both enhancing and non-enhancing solutions pro-

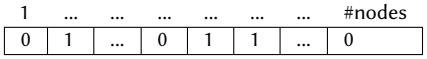

Figure 4: Solution representation.

gressively. This approach prevents the algorithm from becoming trapped in local optima and facilitates exploration of broader regions within the solution space. Let $f(S)$ represent the execution time of the workload in the presence of a subset S of intermediate nodes. The objective is to minimize $f(S)$ while adhering to the SSD capacity constraint (see equations 3 and 4). A solution can be depicted as shown in figure 4. An admissible solution is one that adheres to the storage constraint. The solution space comprises a collection of admissible solutions.

5.3.1. Simulated Annealing Iterative Process

Algorithm 1 Simulated Annealing for Node Selection

Require: Initial solution S, Initial temperature T, α: Cooling parameters, E_{th} : Energy threshold, T_{min} : Temperature minimum threshold,
Ensure: $S =$ Optimal sub set of nodes
1: $T \leftarrow T_{max}$
2: $E(S) \leftarrow$ compute the energy of S
3: **while** $(T > T_{min})$ and $(E > E_{th})$ **do**
4: Generate a neighbor solution S' (small change to the current solution S)
5: $E(S') \leftarrow$ compute the energy of S'
6: $\Delta E = E(S') - E(S)$
7: **if** $\Delta E > 0$ **then**
8: Accept the new solution S'
9: **else**
10: Generate a random number r between 0 and 1
11: **if** $r < e^{-\frac{\Delta E}{T}}$ **then**
12: Accept the new solution S'
13: **else**
14: Reject the new solution S'
15: **end if**
16: **end if**
17: Update temperature: $T \leftarrow \frac{T}{\alpha}$
18: **end while**
19: Return final solution S

As illustrated by algorithm 1, during each iteration, the algorithm generates a set of materialized views by making incremental adjustments to the existing solution. Subsequently, it assesses the disparity in energy (or objective function) between the prevailing solution and the newly generated solution. If the new solution is better, it is adopted as the current solution. Conversely, if the new solution is inferior, it is accepted with a probability that diminishes over time, contingent upon the energy difference and the temperature parameter. The temperature gradually decreases, thereby regulating the likelihood of embracing suboptimal solutions throughout the process. This iterative cycle is repeated until the stopping criterion is met. For both an optimal final solution and a reasonable convergence time, the initial solution is not generated randomly but rather with nodes checking the following constraint which represents their degree of eligibility to be materialized:

$$E(rn) = f_{rn} * \frac{cost(rn)}{\|rn\|} > th \tag{5}$$

This value gives preference to frequent, small nodes with a significantly high computational cost, ensuring that their values exceed a specific threshold (th). In creating a neighboring solution, we assess the impact on the overall workload execution (fitness function) for each intermediate node when removed from the current solution. If removing the node enhances the fitness, it is substituted with a randomly selected node from the pool of unselected nodes.

6. Implementation and Experimentation

In our experiments, we relied on the technique of simulation to evaluate the performances of our proposal. This can be justified by the following three reasons: (1) The performance evaluation on different real flash memories and for the same given workload gave different results [60, 61, 62] (2) The simulation enables precise control over all physical parameters of the disk, specifically facilitating the simulation of multiple FTL and wear leveling algorithms (3) separately control the IOs generated by different processes (OS processes and the different processes executed by the DBMS). In this work, we used a modified version of DiskSim and FlashSim and simulator developed in C language at the University of Michigan, and Canergy Mellon [63].

6.1. Disk and Flash Simulators

DiskSim and FlashSim are two powerful tools that enable the creation of simulation environments and facilitate the evaluation of flash memory performance. The DiskSim tool serves two primary purposes: first, to gain insights and conduct in-depth analyses of storage system performance, and second, to assess the effectiveness of new architectural designs [37]. DiskSim operates with two essential inputs: (1) a configuration file that defines the system's structure to be tested. In this file, various essential components such as buses, controllers, disks, etc., can

be instantiated. (2) a trace file, representing the simulated trace that models the requests received by the system, including reads, writes, and other relevant operations. The configuration file comprises essential information, including:

(i) Global: general simulation options are specified here.
(ii) Stats: this section defines various statistics to be collected during the simulation.
(iii) Iosim: options related to the input trace file used in the simulation.
(iv) System Components: the file includes details about the components to be simulated, such as Buses, Drivers, Controllers, and Disks, each of which can have numerous parameters. These components are instantiated based on provided settings.

Once the components are instantiated, the configuration file defines the system's topology by interconnecting instances of the components to create a coherent system structure. The trace file contains the sequence of I/O requests that the system will execute during the simulation. Each log file entry includes the following details:

(i) Device: The device involved in the I/O operation.
(ii) Block: The specific block associated with the request.
(iii) Size: The size of the I/O request.
(iv) Flags: Additional flags or attributes related to the request (read/write, etc.).

FlashSim is a flash memory simulator [48] that can be seamlessly integrated into DiskSim. It emulates the behavior of an SSD disk and is implemented in the C programming language. One of its key functionalities is to assess the performance of flash memories with various Flash Translation Layers (FTLs) implemented. The FTLs available in FlashSim include pagemap (page mapping), fast2 [64], and DFTL [51]. Similar to DiskSim, FlashSim has undergone validation by comparing its results with measurements obtained from real SSD disks [49]. In FlashSim, flash memory pages are divided into sectors, typically with four sectors per page. Flash memory is implemented through a comprehensive set of data structures and functions that efficiently manage various operations, including reading and writing individual pages, erasing blocks, and handling data invalidation for both pages and blocks. Additionally, the implementation includes functions to initialize and terminate instances of the flash memory model. The flash memory is conceptually represented as an array of data structures; each structure is specifically associated with a physical block of memory. These data structures store essential information pertaining to the block, allowing for effective management and control of the flash memory system. Each structure contains the following information:

(i) Block erase counter;
(ii) Number of free pages;
(iii) Number of invalid pages;
(iv) Last page listed;

In addition, a table that has a size equivalent to the number of pages present in the block. For each page within the block, it provides specific information, corresponding to the out of band area (OOB) of the pages. This information contains:

(i) A bit to indicate if the page is valid;
(ii) A bit to indicate if the page is free;
(iii) A field (30 bits) containing the logical page number corresponding to the physical page in which it is written.

Additionally, many global variables are present containing various information like memory size and number of free blocks. In FlashSim, the FTL algorithms facilitate the computation of various metrics, including response times, page read and write counts, and block erasure counts, enabling comprehensive performance evaluation.

6.2. Principle of the simulation

Figure 5 shows how I/O requests are generated from SQL code and processed by the subsystem simulated by DiskSim and FlashSim. The simulation of the execution of a query involves: (1) the execution of the workload at the DBMS level and the generation of trace files; (2) analysis of the traces by the DiskSim parser, which sends the request to the SSD interface (interface between DiskSim and FlashSim) and which converts requests addressing sectors into requests addressing pages; (3) Then the FTL (depending on its type) will perform different tasks to determine the physical address targeted by the request and send the request to flash memory for execution. We've developed a simulator equipped with both a 64GB flash SSD and an 80GB HDD. The flash storage is partitioned into uniform erase blocks, each having a size of 16 KB. Within each erase block, there are 32 pages used as read/write units, and each page has a size of 512 bytes.

6.3. Hardware environment and Dataset

Our experiments were carried out using an open-source operating system and DBMS on a server equipped with a 2.60GHz CPU and 8GB of RAM. The dataset used was an expanded version of the TPC-H benchmark database, generated with a scale factor of 2. The schema comprises 8 tables with a cumulative size of approximately 2 gigabytes, hosted on the Oracle DBMS.

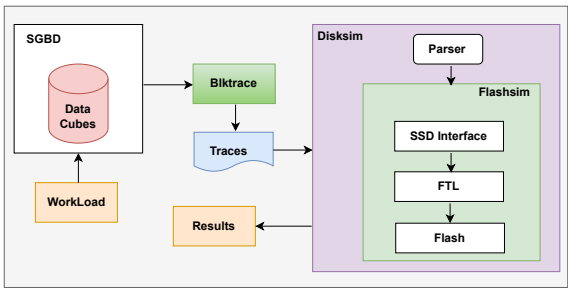

Figure 5: Principle of the simulation.

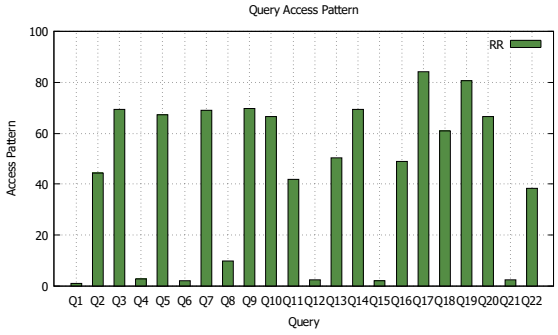

Figure 6: TPC-H query pattern.

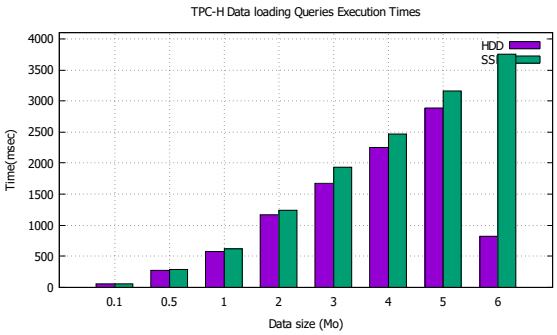

Figure 7: Data writing on SSD & HDD.

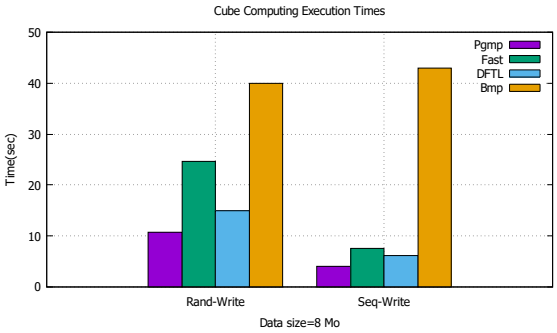

Figure 8: FTL impact on writing data.

6.4. Discussion

The characterization of the resultant workload access patterns from the TPC-H data related the software and hardware environment described above generated the patterns depicted in the figure 6; where RR denotes Random read. Even in the presence of indexes, we notice that a good number of query have induced a sequential access pattern to the data. These queries are: $Q_2, Q_4, Q_6, Q_8, Q_12, Q_16$ and Q_{21}. The rationale is that the data concerned by these queries had a high spatial locality compared to the other data. Most OLAP operations can be broken down into four main database operations: Insert, retrieval or table scans, Join, and Group-By. For example, slicing and dicing require the use of the Select statement while OLAP cube generation mainly utilizes both Group-By and Select statements. For each of these database operations, we have performed experiments based on the TPCH 22 queries. Within a database management system, the speed of Insert statements rely heavily on the writing speed of the storage drive and involve little, if any, read speeds. For this statement, the simulated SSD should not be able to outperform the ordinary HD as shown in figure 7. In fact, the erase before write constraint plays a negative role for the write operation. Indeed, the writing of data strongly depends on (1) the logic of the FTL algorithm (2) the state of the flash memory (percentage of free, busy and invalid blocks) and (3) The strategies initiating the garbage collector can lead to erasure operations that have a harmful impact on performance. Indeed figure 8 shows the impact of the type of FTLs on the overall performance of the SSD disk. It is clear that block mapping type FTL presents rather poor performance because in the case where the memory already contains data, each page write triggers the invalidation of a block and its erasure (according to the principle used by garbage collector). We also notice that the sequential write resulting from the groupings of several insert instructions has greatly improved performance because writing several small amounts of data is the bad use case for flash memory. The second database operation essential to fast OLAP operations is Select statements or, more simply, the speed at which the drives can perform a table scan. As depicted in Figure 9, the introduction of the SSD disk has led to a notable enhancement in the execution time of all queries. In this situation, SSDs have two main advantages over HDs. First, the lack of a seek and rotation latency means that SSDs have the same access speed regardless of where the data is stored. In

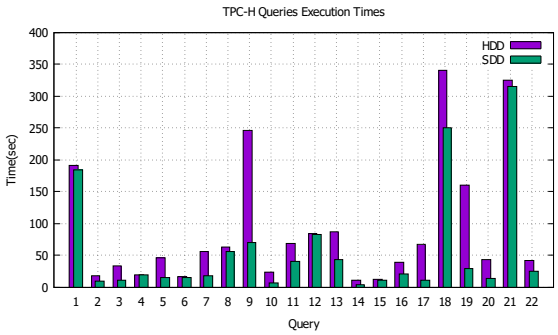

Figure 9: TPC-H query execution time compared across HDD and SSD.

most systems, large datasets are not going to be stored in continuous blocks, and will instead be fragmented across the drive. As a result, the need to move a mechanical head around greatly inhibits a HDD.

For the third database operation, OLAP techniques often use Joins to combine a fact table with smaller dimension tables to obtain additional information. Since most OLAP Joins are between large tables and smaller tables, we assume that hashes are used to match the rows. This operation relies on the speed of accessing the tables and also on the speed at which the rows can be matched. For our work, we are less concerned with the second portion because it depends more on CPU speed and algorithm strategy than storage drive. In addition, since the storage drives are stored in the same computer (the trace is extracted from the same workload execution), there should be little difference in matching speed. As a result, the main difference in Join performance relies on access speed. Since we know that SSDs have much faster access speeds than HDs, it can be expected that Joins will also perform faster on solid state drives than hard drives. Group-By statements are the final database operation essential to OLAP techniques. As with Joins, the performance of this operation in our situation relies heavily on access speed. Group-By operations is composed of reading from a dataset and aggregating the appropriate results. Again, since we are using the same CPU, the time to aggregate common rows should be equivalent between the SSD and HD. Access speed of the data is again the main cause of the difference in performance between the two storage devices. In this case, we expect always that the SSD should greatly outperform HD when it comes to grouping large datasets. In fact, we would expect the performance gap to increase as the size of the data warehouse increases in particular for queries whose result cardinality depends on the size of the database. The final approach pertains to multi-query optimizers that rely on the similarity of a specific set of queries over a duration. Queries deemed similar share a substantial number of intermediate results, such as accessing identical tables, performing joins on the same columns, or applying filters based on similar predicates. While the TPCH benchmark facilitated a comprehensive examination of utilizing databases on SSD, the 22 queries employed did not sufficiently underscore the aspect of similarity in the realm of multi-query optimization. To underscore the utility of employing materialized views on SSD, we turned to the 30 queries provided by the SSB benchmark [10]. The Star Schema Benchmark (SSB) was specifically crafted to evaluate star schema optimization, aiming to tackle the challenges identified in TPC-H. Its primary purpose is to evaluate the efficiency of multi-table JOIN queries within a star schema.

Figure 10 shows the costs calculated by our cost model for all queries without and with materialization. The materialized views are those of the best solutions found by the simulated annealing algorithm. This solution includes 7 nodes $N_1 \dots N_7$, where $N_i \in \{(1 \bowtie 6), (6 \bowtie 7), (1 \bowtie 2 \bowtie 4 \bowtie 9), (1 \bowtie 2 \bowtie 8 \bowtie 5), (1 \bowtie 12 \bowtie 13 \bowtie 18), (1 \bowtie 3 \bowtie 13), (1 \bowtie 3 \bowtie 14 \bowtie 15 \bowtie 16)\}$ (see table 1 for node definition and property).

Figure 11 depicts the overall performance of the simulated workload of SSB using materialized views on different SSDs configurations. Considering that the SSB workload encompasses a significant volume of physical-level reads characterized by a random pattern, all configurations of simulated SSDs produced superior results compared to the HDD, even when materialized views were considered. Nevertheless, the variations observed in the performance of each disk type (FTL) are predominantly dictated by the expenses associated with garbage collection and address translation operations.

In Figure 12, although FAST demonstrates commendable performance, it falls short of achieving the performance level of page-level FTL. This shortfall is attributed to the impact of update operations in the TPC-H workload, influencing overall performance due to merge operations in FAST. Additionally, the incurred extra data read cost in log blocks contributes to the degradation of its read performance. DFTL operates on a two-tier page-level mapping. However, when the workload lacks high temporal locality, it becomes susceptible to additional address translation overhead. This inherent characteristic is the primary reason why DFTL does not showcase a relatively robust random read performance comparable to the ideal scenario of the page mapping scheme.

We applied the materialized view schemas provided by the simulated annealing algorithm in Oracle 21c, employing both real HDD and SSD disks. Figures 13 and 14 depict the execution times for each query and the durations required to materialize the selected nodes on each storage medium, respectively. Based on Figure 13, we can infer the following: (1) Oracle DBMS consistently

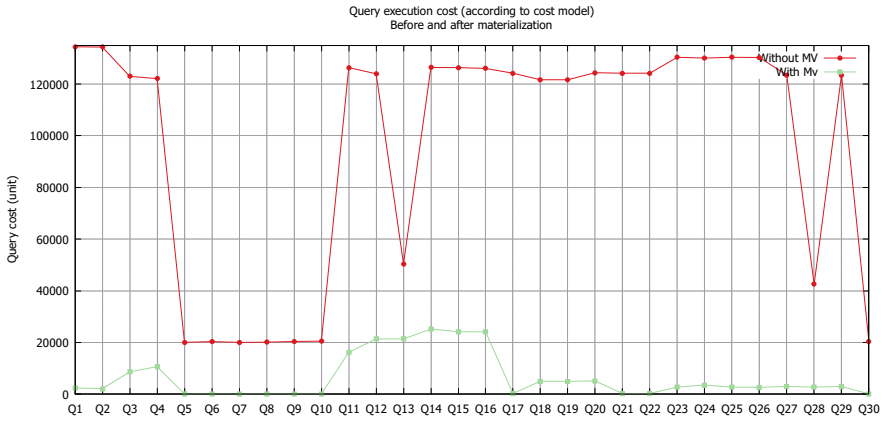

Figure 10: Query execution cost determined by the optimal pattern identified through the simulated annealing algorithm.

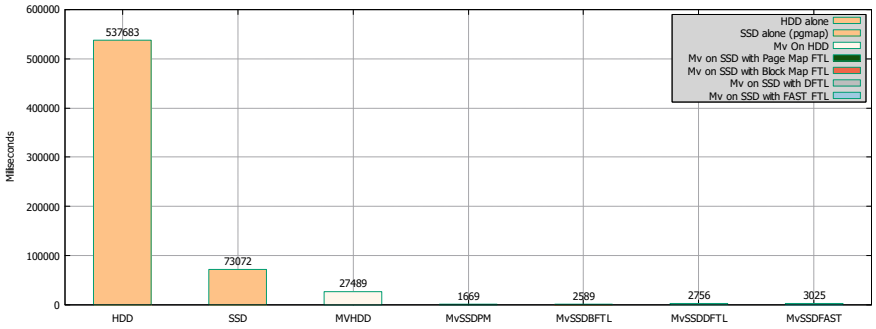

Figure 11: SSB workload execution using materialized views.

selects materialized views as the optimal plan for executing queries, and (2) the query execution times using materialized views on SSD are consistently superior to those on HDD. This is attributed to the fact that these times are primarily influenced by IO costs rather than CPU costs. Upon analyzing the response times provided by both the disk and flash memory simulators, it becomes evident that they are considerably higher compared to the times observed in actual deployment on a real SSD disk (as depicted in figures 11 and 13). This difference can be attributed to the incorporation of recent high-performance techniques, such as caching and parallelism, in real SSD disks. Figure 14 indicates that, for a 1GB data warehouse, the average materialization time for an intermediate node is around ten seconds. Nevertheless, this time significantly increases as we scale up. In a scenario with offline environment with predetermined workloads, this isn't a concern, as it's feasible to materialize nodes during periods of low user system usage. However, in a dynamic environment with frequently changing query workload patterns, it is advisable to employ SSD as a second-level cache and implement more sophisticated cache management techniques.

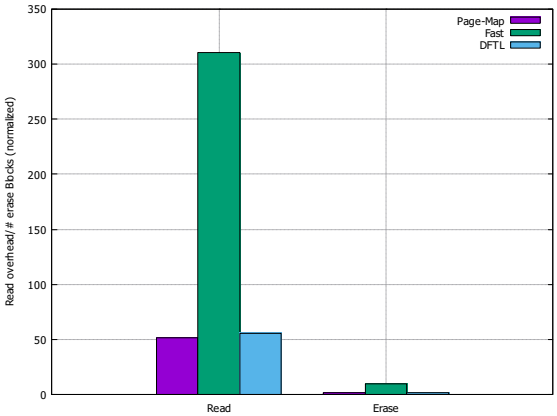

Figure 12: Garbage Collection Overhead for Various SSDs using TPC-H.

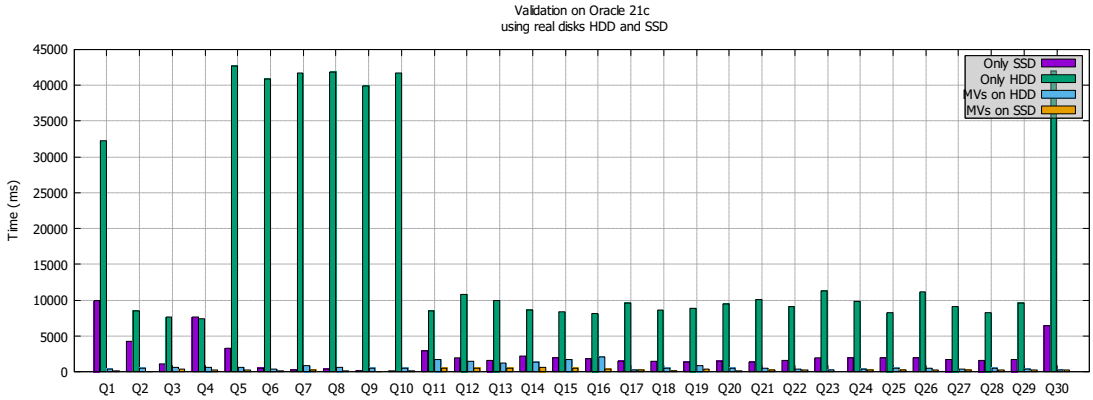

Figure 13: Execution time on oracle 21c.

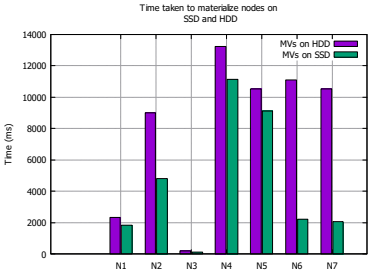

Figure 14: Materialization time of shared nodes.

7. Conclusion

In this paper, we showed that the SSD is able to perform at or above that of the HD for all experiments with the exception of Insert. The SSD had better performance for all other primitive database operations, such as Selects, Joins, and Group-By statements, that were vital to fast OLAP processing. We have also shown that the hybridization of HDD and SSD media using techniques offered by modern DBMSs such as materialized views can enormously contribute to the optimization of the overall operation of the storage system. In the end, we believe that it is indeed feasible to perform OLAP operations on a solid state drive as compared with a traditional magnetic drive. A significant amount of work remains to improve the understanding of the SSD utilisation in data warehouse and OLAP environments.

We expect in future work: to study and understand the patterns of sequentially and randomly distributed writes and the behaviour of different FTLs in their management. In this work, we used mostly the simulator with its default options. We plan to implement more advanced FTLs of different types and garbage collectors to better control what happens inside an SSD disk to better address current shortcomings, particularly those of writing. Furthermore, we consider it essential to enhance the comprehensiveness of our experimental analysis by incorporating different DBMSs to validate our findings and gain insights into broader trends. Ultimately, we aim to implement optimization techniques tailored for flash memory within the DBMS, including strategies such as partitioning, indexing, and caching.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] W. H. Inmon, Building the Data Warehouse, 4 ed., Wiley, Indianapolis, IN, 2005.

[2] C.-H. Wu, T.-W. Kuo, L. P. Chang, An efficient b-tree layer implementation for flash-memory storage systems, ACM Trans. Embed. Comput. Syst. 6 (2007) 19–es. URL: https://doi.org/10.1145/1275986.1275991. doi:doi:10.1145/1275986.1275991.

[3] I. E. Tibermacine, A. Tibermacine, W. Guettala, C. Napoli, S. Russo, Enhancing sentiment analysis on seed-iv dataset with vision transformers: A comparative study, in: Proceedings of the 2023 11th international conference on information technology: IoT and smart city, 2023, pp. 238–246.

[4] N. Brandizzi, V. Bianco, G. Castro, S. Russo, A. Wajda, Automatic rgb inference based on facial emotion recognition, in: CEUR Workshop Proceedings, volume 3092, 2021, p. 66 – 74.

[5] Z. Chen, C. Ordonez, Optimizing olap cube processing on solid state drives, in: Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP, DOLAP '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 79–84. URL: https://doi.org/10.1145/2513190.2513197. doi:doi:10.1145/2513190.2513197.

[6] A. Alfarano, G. De Magistris, L. Mongelli, S. Russo, J. Starczewski, C. Napoli, A novel convmixer transformer based architecture for violent behavior detection, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 14126 LNAI, 2023, p. 3 – 16. doi:doi:10.1007/978-3-031-42508-0_1.

[7] G. Capizzi, C. Napoli, S. Russo, M. Woźniak, Lessening stress and anxiety-related behaviors by means of ai-driven drones for aromatherapy, in: CEUR Workshop Proceedings, volume 2594, 2020, p. 7 – 12.

[8] N. Brandizzi, S. Russo, G. Galati, C. Napoli, Addressing vehicle sharing through behavioral analysis: A solution to user clustering using recency-frequency-monetary and vehicle relocation based on neighborhood splits, Information (Switzerland) 13 (2022). doi:doi:10.3390/info13110511.

[9] I. Naidji, A. Tibermacine, W. Guettala, I. E. Tibermacine, et al., Semi-mind controlled robots based on reinforcement learning for indoor application., in: ICYRIME, 2023, pp. 51–59.

[10] P. E. O'Neil, E. J. O'Neil, X. Chen, The star schema benchmark (SSB), 2009.

[11] C. Napoli, V. Ponzi, A. Puglisi, S. Russo, I. Tibermacine, et al., Exploiting robots as healthcare resources for epidemics management and support caregivers, in: CEUR Workshop Proceedings, volume 3686, CEUR-WS, 2024, pp. 1–10.

[12] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680. URL: http://www.jstor.org/stable/1690046. doi:doi:10.1126/science.220.4598.671.

[13] N. Brandizzi, A. Fanti, R. Gallotta, S. Russo, L. Iocchi, D. Nardi, C. Napoli, Unsupervised pose estimation by means of an innovative vision transformer, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 13589 LNAI, 2023, p. 3 – 20. doi:doi:10.1007/978-3-031-23480-4_1.

[14] T.-S. Chung, H.-S. Park, Staff: A flash driver algorithm minimizing block erasures, Journal of Systems Architecture 53 (2007) 889–901. URL: https://www.sciencedirect.com/science/article/pii/S1383762107000458. doi:doi:https://doi.org/10.1016/j.sysarc.2007.02.005.

[15] A. Tibermacine, D. Akrour, R. Khamar, I. E. Tibermacine, A. Rabehi, Comparative analysis of svm and cnn classifiers for eeg signal classification in response to different auditory stimuli, in: 2024 International Conference on Telecommunications and Intelligent Systems (ICTIS), IEEE, 2024, pp. 1–8.

[16] B. Nail, B. Djaidir, I. E. Tibermacine, C. Napoli, N. Haidour, R. Abdelaziz, Gas turbine vibration monitoring based on real data and neuro-fuzzy system, Diagnostyka 25 (2024).

[17] J. Gray, Tape is dead, disk is tape, flash is disk, ram locality is king, http://research.microsoft.com/enus/um/people/gray/talks/Flash_is_Good.ppt, 2006).

[18] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, G. Graefe, Query processing techniques for solid state drives, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 59–72. URL: https://doi.org/10.1145/1559845.1559854. doi:doi:10.1145/1559845.1559854.

[19] J. Do, J. M. Patel, Join processing for flash ssds: remembering past lessons, in: Proceedings of the Fifth International Workshop on Data Management on New Hardware, 2009, pp. 1–8.

[20] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, D. J. DeWitt, Query processing on smart ssds: Opportunities and challenges, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 1221–1230. URL: https://doi.org/10.1145/2463676.2465295. doi:doi:10.1145/2463676.2465295.

[21] S.-W. Lee, B. Moon, Design of flash-based dbms: An in-page logging approach, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD '07, Association for Computing Machinery, New York, NY, USA, 2007, p. 55–66. URL: https://doi.org/10.1145/1247480.1247488. doi:doi:10.1145/1247480.1247488.

[22] J. Do, D. Zhang, J. M. Patel, D. J. DeWitt, J. F. Naughton, A. Halverson, Turbocharging dbms buffer pool using ssds, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD '11, ACM, New York, NY, USA, 2011, pp. 1113–1124. URL: http://doi.acm.org/10.1145/1989323.1989442. doi:doi:10.1145/1989323.1989442.

[23] B. Nail, M. A. Atoussi, S. Saadi, I. E. Tibermacine, C. Napoli, Real-time synchronisation of multiple fractional-order chaotic systems: an application study in secure communication, Fractal and Fractional 8 (2024) 104.

[24] S. Bouchelaghem, I. E. Tibermacine, M. Balsi, M. Moroni, C. Napoli, Cross-domain machine learning

approaches using hyperspectral imaging for plastics litter detection, in: 2024 IEEE Mediterranean and Middle-East Geoscience and Remote Sensing Symposium (M2GARSS), IEEE, 2024, pp. 36–40.

[25] Y. Ou, T. Härder, Improving database performance using a flash-based write cache, in: Proceedings of the 17th international conference on Database Systems for Advanced Applications, DASFAA'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 2–13. URL: http://dx.doi.org/10.1007/978-3-642-29023-7_2. doi:doi:10.1007/978-3-642-29023-7_2.

[26] D. J. DeWitt, J. Do, J. M. Patel, D. Zhang, Fast peak-to-peak behavior with ssd buffer pool, in: Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013), ICDE '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 1129–1140. URL: http://dx.doi.org/10.1109/ICDE.2013.6544903. doi:doi:10.1109/ICDE.2013.6544903.

[27] N. Boutarfaia, S. Russo, A. Tibermacine, I. E. Tibermacine, Deep learning for eeg-based motor imagery classification: Towards enhanced human-machine interaction and assistive robotics, in: CEUR Workshop Proceedings, volume 3695, 2023, p. 68 – 74.

[28] A. Tibermacine, I. E. Tibermacine, M. Zouai, A. Rabehi, Eeg classification using contrastive learning and riemannian tangent space representations, in: 2024 International Conference on Telecommunications and Intelligent Systems (ICTIS), IEEE, 2024, pp. 1–7.

[29] A. Tibermacine, N. Djedi, Neat neural networks to control and simulate virtual creature's locomotion, in: 2014 International Conference on Multimedia Computing and Systems (ICMCS), IEEE, 2014, pp. 9–14.

[30] Z. He, P. Veeraraghavan, Fine-grained updates in database management systems for flash memory, Inf. Sci. 179 (2009) 3162–3181.

[31] C. Salmi, A. Nacef, L. Bellatreche, J. Boukhobza, What can emerging hardware do for your dbms buffer?, in: Proceedings of the 17th International Workshop on Data Warehousing and OLAP, DOLAP '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 91–94. URL: https://doi.org/10.1145/2666158.2666181. doi:doi:10.1145/2666158.2666181.

[32] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, Y. Kotidis, Dwarf: Shrinking the petacube, in: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, SIGMOD '02, Association for Computing Machinery, New York, NY, USA, 2002, p. 464–475. URL: https://doi.org/10.1145/564691.564745. doi:doi:10.1145/564691.564745.

[33] A. Tibermacine, S. M. Amine, An end-to-end trainable capsule network for image-based character recognition and its application to video subtitle recognition., ICTACT Journal on Image & Video Processing 11 (2021).

[34] F. Chen, D. A. Koufaty, X. Zhang, Understanding intrinsic characteristics and system implications of flash memory based solid state drives, SIGMETRICS Perform. Eval. Rev. 37 (2009) 181–192. URL: https://doi.org/10.1145/2492101.1555371. doi:doi:10.1145/2492101.1555371.

[35] S. eddine Boukredine, E. Mehallel, A. Boualleg, O. Baitiche, A. Rabehi, M. Guermoui, A. Douara, I. E. Tibermacine, Enhanced performance of microstrip antenna arrays through concave modifications and cut-corner techniques, ITEGAM-JETIA 11 (2025) 65–71.

[36] S. Russo, I. E. Tibermacine, A. Tibermacine, D. Chebana, A. Nahili, J. Starczewscki, C. Napoli, Analyzing eeg patterns in young adults exposed to different acrophobia levels: a vr study, Frontiers in Human Neuroscience 18 (2024) 1348154.

[37] Y. Lim, J. Lee, C. Campes, E. Seo, Parity-stream separation and slc/mlc convertible programming for lifespan and performance improvement of ssd raids, in: Proceedings of the 9th USENIX Conference on Hot Topics in Storage and File Systems, HotStorage'17, USENIX Association, USA, 2017, p. 21.

[38] B. Ladjal, I. E. Tibermacine, M. Bechouat, M. Sedraoui, C. Napoli, A. Rabehi, D. Lalmi, Hybrid models for direct normal irradiance forecasting: A case study of ghardaia zone (algeria), Natural Hazards 120 (2024) 14703–14725.

[39] A. TIBERMACINE, W. GUETTALA, I. E. TIBERMACINE, Efficient one-stage deep learning for text detection in scene images., Electrotehnica, Electronica, Automatica 72 (2024).

[40] S. Russo, S. Ahmed, I. E. Tibermacine, C. Napoli, Enhancing eeg signal reconstruction in cross-domain adaptation using cyclegan, in: 2024 International Conference on Telecommunications and Intelligent Systems (ICTIS), IEEE, 2024, pp. 1–8.

[41] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, H.-J. Song, A survey of flash translation layer, J. Syst. Archit. 55 (2009) 332–343. URL: https://doi.org/10.1016/j.sysarc.2009.03.005. doi:doi:10.1016/j.sysarc.2009.03.005.

[42] A. Hunter, A brief introduction to the design of ubifs, http://www.linux-mtd.infradead.org/doc/ubifs_whitepaper.pdf, 2008).

[43] Y. Editor, Yaffs overview, https://yaffs.net/yaffs-overview, 2022).

[44] D. Woodhouse, Jffs: The journalling flash file system, in: Ottawa linux symposium, volume 2001, Citeseer, 2001.

[45] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, J.-S. Kim, A reconfigurable ftl (flash translation layer) architecture for nand flash-based

applications, ACM Trans. Embed. Comput. Syst. 7 (2008). URL: https://doi.org/10.1145/1376804.1376806. doi:doi:10.1145/1376804.1376806.

[46] M.-L. Chiang, C.-L. Cheng, C.-H. Wu, A new ftl-based flash memory management scheme with fast cleaning mechanism, in: 2008 international conference on embedded software and systems, IEEE, 2008, pp. 205–214.

[47] E. Gal, S. Toledo, Algorithms and data structures for flash memories, ACM Comput. Surv. 37 (2005) 138–163. URL: https://doi.org/10.1145/1089733.1089735. doi:doi:10.1145/1089733.1089735.

[48] J. Kim, J. M. Kim, S. Noh, S. L. Min, Y. Cho, A space-efficient flash translation layer for compactflash systems, IEEE Transactions on Consumer Electronics 48 (2002) 366–375. doi:doi:10.1109/TCE.2002.1010143.

[49] Y. Kim, B. Tauras, A. Gupta, B. Urgaonkar, FlashSim: A simulator for NAND flash-based solid-state drives, in: 2009 First International Conference on Advances in System Simulation, IEEE, 2009, pp. 125–131. URL: https://doi.org/10.1109%2Fsimul.2009.17. doi:doi:10.1109/simul.2009.17.

[50] D. Park, B. K. Debnath, D. H.-C. Du, Cftl: A convertible flash translation layer with consideration of data access patterns, Minneapolis, MN: University of Minnesota (2009).

[51] A. Gupta, Y. Kim, B. Urgaonkar, Dftl: A flash translation layer employing demand-based selective caching of page-level address mappings, SIGPLAN Not. 44 (2009) 229–240. URL: https://doi.org/10.1145/1508284.1508271. doi:doi:10.1145/1508284.1508271.

[52] T. K. Sellis, Intelligent caching and indexing techniques for relational database systems, Inf. Syst. 13 (1988) 175–185. URL: http://dx.doi.org/10.1016/0306-4379(88)90014-2. doi:doi:10.1016/0306-4379(88)90014-2.

[53] C.-M. Chen, N. Roussopoulos, The implementation and performance evaluation of the adms query optimizer: Integrating query result caching and matching, in: M. Jarke, J. A. B. Jr., K. G. Jeffery (Eds.), Advances in Database Technology - EDBT'94. 4th International Conference on Extending Database Technology, Cambridge, United Kingdom, March 28-31, 1994, Proceedings, volume 779 of *Lecture Notes in Computer Science*, Springer, 1994, pp. 323–336.

[54] P. Roy, S. Seshadri, S. Sudarshan, S. Bhobe, Efficient and extensible algorithms for multi query optimization, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD '00, ACM, New York, NY, USA, 2000, pp. 249–260. URL: http://doi.acm.org/10.1145/342009.335419. doi:doi:10.1145/342009.335419.

[55] Y. Kotidis, N. Roussopoulos, Dynamat: A dynamic view management system for data warehouses, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, SIGMOD '99, ACM, New York, NY, USA, 1999, pp. 371–382. URL: http://doi.acm.org/10.1145/304182.304215. doi:doi:10.1145/304182.304215.

[56] M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1990.

[57] D. Pisinger, Where are the hard knapsack problems?, Comput. Oper. Res. 32 (2005) 2271–2284. URL: http://dx.doi.org/10.1016/j.cor.2004.03.002. doi:doi:10.1016/j.cor.2004.03.002.

[58] D. S. Hirschberg, C. K. Wong, A polynomial-time algorithm for the knapsack problem with two variables, J. ACM 23 (1976) 147–154. URL: http://doi.acm.org/10.1145/321921.321936. doi:doi:10.1145/321921.321936.

[59] J. Yang, K. Karlapalem, Q. Li, A framework for designing materialized views in data warehousing environment, in: Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), ICDCS '97, IEEE Computer Society, USA, 1997, p. 458.

[60] L. Bouganim, B. Jonsson, P. Bonnet, uflip: Understanding flash IO patterns, in: Fourth Biennial Conference on Innovative Data Systems Research, CIDR 2009, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings, 2009. URL: http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_102.pdf.

[61] D. Ajwani, I. Malinger, U. Meyer, S. Toledo, Characterizing the performance of flash memory storage devices and its impact on algorithm design, in: Proceedings of the 7th International Conference on Experimental Algorithms, WEA'08, Springer-Verlag, Berlin, Heidelberg, 2008, p. 208–219.

[62] P. G. Harrison, N. M. Patel, S. Zertal, Response time distribution of flash memory accesses, Performance Evaluation 67 (2010) 248–259. URL: https://www.sciencedirect.com/science/article/pii/S0166531609001412. doi:doi:https://doi.org/10.1016/j.peva.2009.10.003, performance Evaluation Methodologies and Tools: Selected Papers from VALUETOOLS 2008.

[63] G. R. Ganger, B. L. Worthington, Y. N. Patt, The disksim simulation environment – version 1.0 reference manual, 1998.

[64] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, H.-J. Song, A log buffer-based flash translation layer using fully-associative sector translation, ACM Trans. Embed. Comput. Syst. 6 (2007) 18–es. URL: https://doi.org/10.1145/1275986.1275990. doi:doi:10.1145/1275986.1275990.