University of Kaiserslautern
Department of Computer Science
Database and Information Systems

Seminar
Recent Trends in Database Research

Summer Semester 2013

Table of Contents

Flash-Based Caching For Databases - Energy Efficiency and Performance

Ankit Chaudhary

University of Kaiserslautern

Abstract. Purpose of this report is to explain benefits of using flash memory for storing databases. The beginning of this report contains information about the traditional storage devices and mechanisms used for databases. Then it will take a look at how can we utilize flash memories with databases. It will further discuss various different architectures that can be used with flash memory in order to achieve optimum utilization and efficiency. This report will also discuss various algorithms used in database operations to further optimize their performance.

1 Introduction

1.1 Database History

In the 21st century, storing information is becoming absolutely necessary for any organization to excel and evolve. In fact, the efforts of storing the information began from mid 20th century with an initial effort on storing system-critical data, which was about an order of magnitude smaller than that of today's data-crazy applications. A study of 2010 by EMC Corporation (NYSE:EMC) shows that, in 2009 amid the Great Recession, the amount of digital information grew 62% over 2008 to 800 billion gigabytes (0.8 Zettabytes). One Zettabyte equals one trillion gigabytes [ID08].

This raise in data volume is the curtsy of information-hungry applications, which not only consume the information but also produce it with same rate. As we grow and advance in Information Technology, we automatically start to generate information. You may not have noticed, but, every time we interact with some machine, we produce information or data. You want to drink a cup of coffee. You select the type of coffee you want to have on a coffee machine and well you just generated information that will be used by the machine to produce coffee. You travel to a foreign destination, right from the planning of the trip to the time you arrive at the destination, you take part into a process which generates a tremendous amount of information, like your inventory, food preference, luggage information, flight information, etc., and this list of data-consuming and -producing applications is growing rapidly with time.

Now, the more we increase the use of information technology in our daily life, the more data we produce. Hence, we need to look for various storage mechanisms to store the wide range of heterogeneous data. In fact, this look out for suitable data storage mechanisms started as early as in 1960 when IBM came up with the

SABRE system that was used by American Airlines to manage its reservations data. After that between 1970–1972, E.F. Codd came up with the concept of the relational database model. It turned out to be the revolution in database technology and its applications. This actually broadened the scope of databases by introducing the concept of schema and by separating it totally from the data physically stored; this became the base principle on which future database systems worked.

Shortly after E.F. Codd came up with his concept of separating the conceptual and physical organization of a database, query languages like QUEL and SQUEL were designed. Later they led to the development of database systems like MS SQL Server, Sybase, DB2, Allbase, Oracle, etc. During this decade of revolution in database systems, the term Relational Database Management System, or RDBMS, got recognized in industry. Since then, we are continuously working on standardizing the various database paradigms, like the query language by SQL in 1980's, and coming up with newer and faster database systems and mechanisms.

1.2 Caching in a Database

In our database systems today, we use mainly HDDs (hard disk drive, magnetic disk) for storing data persistently and DRAM (dynamic random access memory) for keeping data to be processing in a volatile memory. While HDD is used as external storage (because of huge size, slow R/W speed, and cheap cost), DRAM is used for caching a small amount of currently processed data (because of small size, fast R/W speed and higher cost).

With the huge amount of data being produced, it is important now to use databases. Database management deals with storage and retrieval mechanisms (and much more). Nowadays, such systems are usually designed as a multi-tier architecture, where the application-tier and the data-tier are assigned to completely independent computing nodes connected via a network. In order to run these applications efficiently and without any latency, mostly due to the network, we have to come up with a mechanism of caching the data on the application-tier itself by using some lightweight database mechanisms. In some of the cases, we have application and data lying on the same machine. In these cases also, performance can get impacted by continuous update and retrieval of data from physical disks, which exhibit slow read and write performance. For this reason, we use the much faster main memory, with much superior write and read performance compared to disks, for caching frequently used data. In this report, we are analyzing an architecture where application and data are lying on same machine.

Caching of data is very important for performance improvement of any application. In later sections, we will also discuss how efficient and clever usage of data caching not only helps in improving performance and throughput of a database system, but also helps in achieving energy efficiency by saving a large number of HDD operations.

1.3 Caching using Flash Memory

Flash memory is the latest technology used for storing small to large amounts of data. It is a type of semiconductor-based non-volatile memory. Flash is of special interest, because, unlike HDDs, it provides high read throughput and, unlike DRAM, it consumes much less electricity for accessing the data. Because of the missing mechanical arm movement, flash memory is also shock-proof, which enables many interesting applications of flash memory in mobile devices. Recently, flash memory has been actively used as an external storage device in the form of SSDs (solid state disks) which provide the same interface as HDDs.

In contrast to DRAM, flash memory has an upper hand in cost, but provides less read throughput. Its cost has substantially reduced in the past few years, because of increase in sale's volume and better mechanisms of production. If we compare it to HDD, it has no mechanical arm and hence does not have latency due to missing arm movement for data operations. This has enabled flash memory to achieve a high number of IOPS (input/output operations), which could only matched by having several HDDs working in parallel. Hence, we can use flash memory in place of HDDs (in the form of SSDs) or maybe in combination with DRAM (as secondary buffer storage) to further improve the DB performance.

If we use flash memory instead of DRAM, we can use it for the purpose of caching. Because we have a faster DRAM, but limited in size due to high cost per GB, we can make use of flash memory with a larger size, while approximating the read throughput of DRAM. Because flash memory provides much less write throughput, which is comparable to that of HDDs, we can't use flash to totally replace DRAM. Instead, we need to look for a mechanism to use flash memory in combination with DRAM for caching. This will be one of the main problem statements discussed throughout this report.

1.4 Problem Statement: Energy Efficiency and Performance

Energy efficiency and performance are two important properties, which not only make a product unconquerable but lagging in any of these properties will make it obsolete in no time. Like for any industry product, both of them are important properties for database products as well. We now consider three different storage devices : HDD, Flash, and DRAM, for use in database systems. All of them exhibit varying behavior in terms of performance and energy consumption.

In 2005, it was estimated that power consumed by servers in U.S. is 0.6% of its total annual consumption. When the cooling and auxiliary infrastructure is also included, the percentage became 1.2%, which is nearly equal to five 1000 MW power plants. The cost of running these servers in U.S. came up to 2.7 billion dollars per year [KM07]. Hence, it becomes extremely important to use a technology which can cut down this huge operational cost. This is necessary, because the number of server installations is increasing rapidly with every passing year. The relationship between the cost of installation and cost of operation is

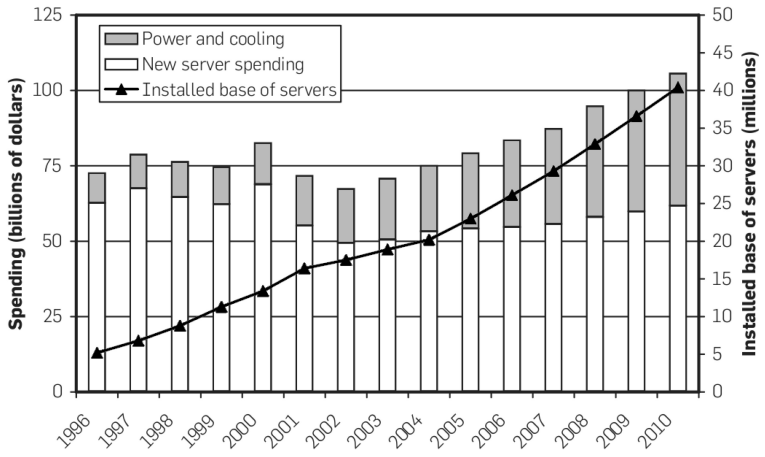


Fig. 1. IDC estimates for worldwide annual cost spent on powering and cooling servers [RB09]

shown in Fig. 1, which illustrates a clear trend of increase in operational cost compared to the installation cost of the servers.

Let us consider HDDs first. HDDs are introduced in the 1950's by IBM and are still most commonly used as external storage devices. They have a very cheap price per GB in contrast to the other two storage devices under consideration. HDDs suffer from high latency and, despite of more than half decade of research and development, a high-end HDD can only achieve a maximum throughput of 1000 IOPS (I/O per second). Now, the mechanical arm movement, apart from contributing towards high latency, also leads to high consumption of energy. Hence, despite being cheap in price, HDDs suffer heavily in terms of energy efficiency and performance. Next we have DRAM, which provides very high throughput and hence performance, but is very expensive in terms of per GB cost. Apart from having high cost, it also consumes a lot of energy because of frequent refresh cycles for its capacitors due to charge leakage. Hence, DRAM is faster than HDD but is more costly and consumes a high amount of energy.

To overcome the problems of these two storage devices, we have to use flash memory smartly. Flash memory is the latest technology used for storing large amounts of data. It is a type of semiconductor-based non-volatile memory. Flash memory has comparable read throughput to DRAM and does not have any latency issues as we have in HDD. As compared to DRAM, flash memory has cheaper per GB cost, but sensitively higher than what we have for HDD. In comparison to both DRAM and HDD, flash is much more energy efficient. Hence, flash memory is the potential candidate for achieving both high energy efficiency and performance.

1.5 Outline

In the succeeding chapters, we will cover in detail information about how we can use flash memory in database systems.

Chapter 2: We will describe in detail the properties of flash memory and their limitations. We will further discuss, how and what kind of flash memories are suitable for the use in database systems.

Chapter 3: We will discuss the standard architecture for database systems and different hybrid architectures that we can use to take advantage of flash memory in order to improve performance and throughput.

Chapter 4: We will talk about the importance of energy efficiency and performance in case of database systems. We will discuss different caching algorithms that can be used for achieving energy efficiency and performance while using flash memory in database systems.

Chapter 5: Based on our discussions and findings in Chapter 3 and 4, we will conclude how flash memories can be used by a middle-tier cache manager in a 3-tier architecture to achieve our goal of energy efficiency and performance.

2 Flash Memory - Future Storage

In this chapter we will discuss in detail about flash memory. We will discuss what different kind of flash devices are available and their respective properties. This discussion will help us in selecting the flash memory that will best suit our problem statement. We will further discuss about the limitation in flash memories and how to cope-up with them.

2.1 About Flash Memory

Flash memory is latest technology used for storing small to large amounts of data. It is a type of semiconductor-based non-volatile memory. Flash is of special interest, because, unlike HDDs, it has high read throughput and less power consumption. Also, unlike DRAM it consumes no power to store data, is cheaper in cost, and has comparable read throughput.

Flash memory is a kind of electrically erasable programmable read-only memory (EEPROM). It provides persistent storage, i.e., if you switch off the device, flash memory retains the data stored in it. Flash memory consists of arrays of memory cells, made up of floating gate transistors. There are typically Single-Level cells (SLC) which store a single bit in each cell. Nowadays, a more advanced type of flash memory comes with Multi-Level Cells (MLC) which stores multiple bits in each cell. MLC works by selecting from multiple level of charge to apply to flash memory cells. Though, the MLC devices can store more information compared to the SLC counterpart, but they are much slower than SLC flash devices and also have a very low life span because of the high number of P/E cycles. MLC flash can be a better solution for large storage devices like HDDs, but is not good when used in high operation zones like the buffer or cache system [SE10]. Hence, for our problem statement we will consider an SLC flash device.

Depending upon how these memory cells are arranged in flash memory, a further flash memory classification is done as: NOR- or NAND-based flash memory. NOR-based flash memory was used earlier when flash memory came into the picture. It provided flash memory with different advantages like faster read access and a feature called eXecute In Place (XIP), which enabled the program to directly execute from flash memory instead of loading it into RAM first. A big disadvantage of NOR-based flash memory is its cost effectiveness because of low capacities (1–4 Mbytes) and suffers severely due to the large number of writes and erase operations. Because the read latency of NOR-based flash memory is very low, it allows for both code execution and data storage. Despite having features like XIP, NOR-based flash memories are not useful, because they sustain only a very low number of erase cycles [AR02].

The other type of flash memory is called NAND-based flash memory. It has a much larger cell density compared to NOR-based flash memory. It gives more storage capacity and enhanced write and erase rates. NAND-based flash memory enables the addressing by page, word, and bit. This property of NAND-based flash helps to emulate hard disks, which work on bit-level addressing. A relative comparison between NOR- and NAND-based flash memory is shown in Table 1.

Table 1. Comparison between NOR and NAND based flash memories [AR02]

PARAMETER	NOR	NAND
Performance Erase	Very Slow (5 s)	Fast (3 ms)
Performance Write	Slow	Fast
Performance Read	Fast	Fast
Erase cycle range	10,000 to 100,000	100,000 to 1,000,000
Price	High	Very Low

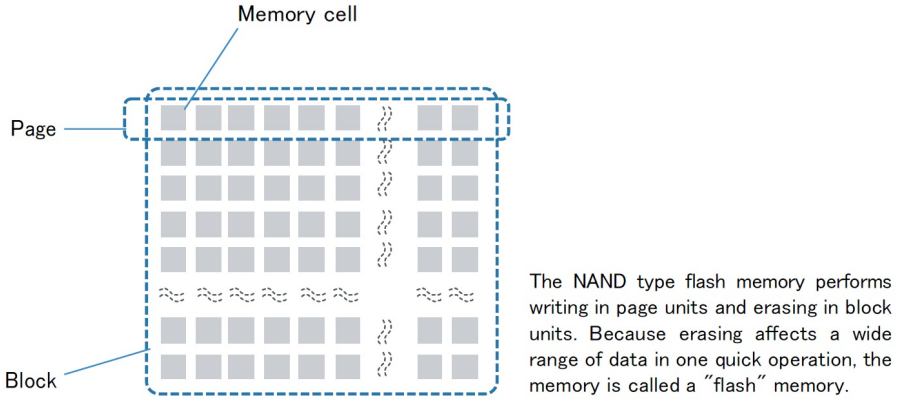
From this comparison, we can easily deduce that, out of the two types of flash memory, it will be beneficial to consider NAND-based Flash memory to achieve our target.

2.2 Flash Operations

Flash memory provides 3 different operations, Read, Program (a.k.a Write), and Erase. These three operations are used by the processor to work with flash memory. The Read and Program operation work on collections of memory cells called flash pages, while the Erase operation works on a larger entity called flash block. Fig.2 shows the arrangement of flash pages and a block in a flash memory.

The latency in flash memory varies with the operations, with the Read operation having the least latency (in microseconds) and the Erase operation having the maximum latency. Also, this latency factor differs with SLC and MLC flash memory (here we are considering NAND-based flash memory). SLC-based flash memory is having the smallest latency factor in all the three operations (i.e., Read, Program, and Erase) compared to MLC-based flash memory.

Fig. 2. Flash Memory Cell Representation [TD12]



During an Erase operation, the flash block becomes a free block and, hence, all pages inside the block and consecutively all the memory cells inside the pages are freed. After the Erase operation, the flash memory block becomes available for fresh writes again, as if the block was never been written before. With the Erase operation a big disadvantage of flash memory is associated, which is called flash wearing or write endurance. Write endurance is discussed later in section 2.3. Using the Write or Program operation, the memory cell values are changed from 1 to 0. This operation is performed on a page unit and hence multiple cells are involved in the operation. To enable the write operation again, an erase operation is performed at block level for again setting the value of all the cells from 0 to 1[TD12]. Hence, to perform a write operation on a used page, an Erase operation must be carried out beforehand. This may create further delays for Write operations, called erase-before-write limitation.

In order to manage the flash wearing due to continuous Write and Erase operations and to interact with computer and electronic devices, M-Systems and SCM Microsystems in 1994 introduced the concept of Flash Translation Layer (FTL), which was later endorsed by Intel.

2.3 Field Translation Layer

Flash memory is not directly accessible by processors and, hence, we need a translation mechanism to work with it. The so-called Flash Translation Layer (FTL) is used for communicating with computer and electronic devices. FTL is responsible for carrying out read, program, and erase operation.

FTL supports logical page operations. Using a traditional update operation on a flash page, a block-level erase operation is carried out and then the new values are re-written to the flash pages, this is called in-place update operation. The in-place update operation is typically very expensive and hence is avoided by the concept of logical page operation. In this scenario, FTL, instead of adopting

in-place update, opts for what is called out-of-place update mechanism. In this process, the updated data is placed into the next free page and is then associated with the logical page address. The previous version of the data is then invalidated by the FTL.

This process increases the *write endurance* of flash memory. The memory cells of flash memory can be used only for a limited number of time (10000 to 1,00,000 Program/Erase cycles [GA05]), before they wear out and can not be used for storing data persistently. The use of logical page addressing is done in order to reduce the number of erase operations to be performed in case of in-place updates and hence, improving the endurance of flash memory. Now, during the update operation, multiple version of a page co-exist. The last updated version is called valid page and rest are called invalid pages. Once, all the free pages are consumed and the cache block is full. A mechanism called Garbage collection is used by FTL to reorganize invalid pages on the blocks and remove them to produce more free pages.

To improve the lifespan of flash memory, FTL needs to take care of the number of erases applied to the individual blocks. If we keep on erasing using the same group of blocks, the blocks will rapidly wear off and hence may lead to an unusable device, because usable blocks are in short supply. Hence, a mechanism called wear-leveling is used by FTL to shuffle among the blocks to equally distribute the number of program/erase operations to be performed.

Hence, we can say that the job of FTL is not just to perform the *Read, Write, and Erase* operations, but also to perform management tasks like *out-of-place updates, garbage collection, and wear-leveling.*

2.4 Flash Performance

The performance of the flash memory depends on how the operation is getting carried out on it. We will discuss few of these performance issues in this section, which need to be taken care of by different algorithms later in this report.

1. **Randomness in operation**: The performance of flash memory suffers from randomness of read and write operations. Though, the flash drives do not have any mechanical arm like disk drives, the performance of read operations can get impacted by random read requests. If we have sequential read requests, flash can take advantage of the read-ahead mechanism and hence can improve the read throughput. Though, during random read requests, impact on IO throughput is not so significant, as it can be with the random write requests. The performance varies as much as four magnitudes when it comes to random write throughput of flash devices [IO09].
2. **Type of Operations**: Performance of flash devices, as discussed earlier, is also severely impacted by the type of operations. Let C_r be the cost of read operation, $C_{w/p}$ the cost of write/program operation, and C_e the cost of erase operation. Then the relationship between the three factors is given by the following equation:

$$C_r < C_w/p < C_e \quad (1)$$

The cost relation is based on the time (ms) taken in performing the operation. The flash update operation involves one write operation and one or multiple flash erase operation. This problem is because of erase-before-write mechanism explained in section 2.2. Also, the number of erase operations decides the life of the flash device. The higher the number of erase operations, the shorter will be its life. This is further explained by the endurance of the flash device.

3. **Other Important Factors**: The type and order of operations primarily effect the performance of a flash device. However, the background operations like Garbage Collection, etc., also impact performance severely. For example, a mixture of read and write operations can have significant negative impact on each other's latency. A read operation may suffer due to a write operation involved in writing back the modified data, required for the read operation. Furthermore, the distribution of flash pages and blocks can also have significant impact on the flash device performance.

2.5 Summary

This chapter talked about the significance of flash memory in terms of energy and performance efficiency. It outlined different kinds of flash memories available and their types suitable for the use in caching. This chapter then discussed different operation types and performance issues faced during the operations. This chapter further described FTL, an important part of a flash device, responsible for garbage collection, out-of-place update, and wear leveling. In the last section of the chapter, we discussed performance issues of flash devices that need to be taken care when used as a database caching device.

3 Database Architecture – An Overview

Before we go for an integration of flash devices with a database system for improving performance and energy efficiency, it is necessary to understand the basic components of database systems. We will discuss a DBMS reference architecture. We then talk about an important DBMS component called buffer management and later discuss the different kinds of architectural variants possible with flash devices used in a DBMS context.

3.1 DBMS Reference Architecture

Database management systems (DBMS) are important parts of today's tech savvy life style. The more we increase use of information technology in our daily life, the more data we produce. Hence, we need to look for various storage mechanisms to store a wide range of heterogeneous data. A DBMS is the solution

for all our storage and management needs. To study a DBMS system in greater detail, we need to first visit its architecture design and understand its layers.

A widely used DBMS architecture is proposed by Theo Härder which is based on five hierarchically ordered abstract layers. This five-layered architecture is shown in Fig. 3. This architecture describes the abstraction from physical storage to the user interaction level. The more we move upwards in the layers, the more complex the layers become enabling feature-rich operations.[TH05]

Fig. 3. Reference Architecture of DBMS [TH05]

	Level of abstraction	**Objects**	**Auxiliary mapping data**
L5	Nonprocedural or algebraic access	Tables, views, tuples	Logical schema description
L4	Record-oriented, navigational access	Records, sets, hierarchies, networks	Logical and physical schema description
L3	Record and access path management	Physical records, access paths	Free space tables, DB-key translation tables
L2	Propagation control	Segments, pages	DB buffer, page tables
L1	File management	Files, blocks	Directories, VTOCs, etc.

The bottom-most layer is called *File Management*, it deals at the bit level with data stored at non-volatile storage devices. It works in coordination with the operating system's file management for managing the data files on physical storage device. It is responsible for reading, writing, and modifying the data of non-volatile storage devices, i.e., HDDs and flash drives. The L1 layer is responsible for number, type and address location of storage devices, which makes upper layers to operate without worrying about these details. The next-upper layer is called *propagation control*, it is responsible for buffering pages in main memory to speed up the database operations. L2 accesses the pages from L1 physically, whereas the upper layers access the pages logically from L2 independent of their physical location. Hence, L2 accesses the pages from L1 using physical page addressing, but upper layer access the pages using logical page addressing.

The L3, L4 and L5 layers deal with further abstractions and complex features. The layer L3 provides functions like insert, update, and delete at the record level. L4 is responsible for providing access to data and tables via different scans or access paths. The last and top-most layer is eventually responsible for providing records, tuples, tables via standard query languages such as SQL.

In our discussion, we will mainly focus on file management (L1) and propagation control (L2), which deal with non-volatile storage and buffer management.

3.2 Buffer Management

Apart from non-volatile storage areas where all the data of the DBMS is physically stored, another important memory component called Buffer pool is used for processing the data present in the database. Data from physical storage is moved to a much faster memory device (RAM) for processing. Data in the buffer pool is stored in pages and logical data blocks. Now, because of the substantial performance difference between external storage and main memory, the number of pages served from the buffer should be as high as possible and number of accesses to the storage devices should be minimum. The size of main memory is not limited and remains typically much smaller than that of the storage device due to high cost and, hence, the pages stored in the buffer pool located in main memory should have a high re-reference probability, i.e., only frequently requested data are stored in buffer pool.

This property is necessary for the performance and efficiency of any DBMS. It can be explained as follows: A DBMS uses a buffer management algorithm represented by equation 2. This algorithm is responsible for fetching the data from physical storage into the buffer pool to speed up the DB operations. For fetching the data into the buffer pool to satisfy Y physical requests, a cost function is given by equation 3.

$$algo(X, b) \tag{2}$$

X is the sequence of logical data request; b is the number of pages in buffer pool.

$$Cost(Y) = (Y) * C_h \tag{3}$$

Y is the sequence of physical data request; C_h is the constant representing access cost associated with disk drive.

Hence, for a successful buffer algorithm the Cost(algo(X,b)) should always be as low as possible. The main aim of any buffer management algorithm is to keep the *Hit* ratio as high as possible. The Hit ratio is defined as number of physical data requests served directly by buffer pool and hence reducing the cost included in serving the data from physical storage area. In case of a buffer fault i.e., if the data does not exist in buffer pool, the data has to be loaded from physical storage area. To improve the Hit rate many different factors needed to be considered. One of them is to continuously lookout for "Cold" pages and removing them from buffer pool to make room for requested data which are not present in buffer pool. How to decide which page is cold and which one is hot, is decided by buffer management algorithm. This algorithm based on different policies decide on the victim page needed to be removed from buffer pool.

The buffer management algorithm has to optimize the utilization of main memory by considering two different factors, *Spatial* and *Temporal* page locality.

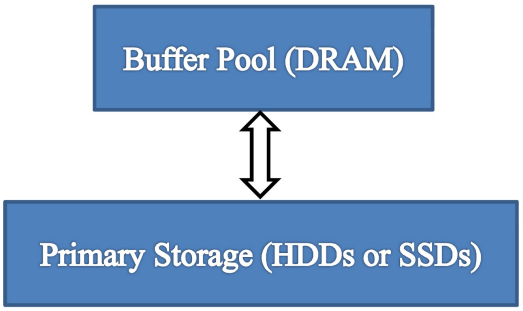
Temporal locality says that if a page is referenced at a point in time, there is a high probability that the same page will be referenced in near future again. Hence, it will be appropriate to keep the page in main memory for faster access. The algorithm named Least Recently Used (LRU) is a popular algorithm which works based on temporal locality. LRU makes sure to choose the page which is being used least recently as the victim. The spatial locality says that the probability of a page to be accessed in near future is higher, if pages in close physical neighborhood (on disk) have been accessed recently. This give rise to the concept of prefetching called Read-ahead, which helps most during sequential access by loading the data into the buffer pool in advance, e.g., in case of table scans.

3.3 Architectural Variants

In this section, we will discuss different kinds of architectures, where flash devices are used for our DBMS. Flash memory has not only revealed lots of opportunities for performance optimization, but has also significantly contributed to the evolvement of different architectural scenarios. We will be discussing three of such architectures below:

1. **Two-tier Architecture** : In this architectural style, we will use flash-based SSDs to replace the standard HDDs for use as primary storage devices. We will take advantage of the fact that flash-based SSDs have the same interface as HDDs. This will follow the standard DBMS architecture that was explained in section 3.1. Replacing the HDDs by SSDs will give us the advantage of high read throughput and energy efficiency. We will use SDDs in bottom-tier and DRAM in top-tier for the buffer pool. The reference architecture is shown in Fig. 4.

Fig. 4. Two-tier Architecture

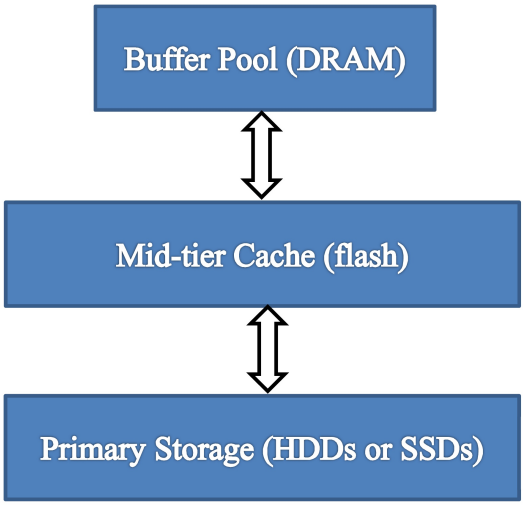


2. **Three-tier Architecture**: Using a three-tier architecture, we introduce a middle-tier between top-tier (based on DRAM) and bottom-tier (based on

SSDs or HDDs) using flash memory. This architecture is illustrated by Fig. 5. In this architecture, we try to leverage the performance advantage of flash memory in the form of main memory. For this reason, we introduce extended main memories, one small and very fast in top layer and other bigger and slightly slower in the middle layer. This will give the system in general a bigger main memory setup at comparatively low cost and higher energy efficiency.

3. **Hybrid Architecture**: In a hybrid architecture, we use SSDs in combination with HDDs at bottom-tier. The architecture is similar to the 2-tier architecture except that now the data will partially resides on both SSDs and HDDs. SSDs usually keep all the hot data, which will help to speed up transfers to the buffer pool (main memory) in case of a buffer fault at the top-tier.

Fig. 5. Three-tier Architecture



The hybrid architecture and the two-tier architecturse are not used for our report, as we want to utilize flash memory as part of the cache. Hence, we have used the three-tier architecture for our analysis and conclusion.

3.4 Summary

In this chapter, we started our discussion by explaining the 5-layer DBMS architecture. We discussed the functionality of important layers like file management and propagation control of a DBMS, which are of relevance when building a system using flash memory as cache. In section 3.2, we discussed the buffer

management algorithm and how it works within the DBMS. We talked about the hot and cold pages and how a buffer management algorithm works to maintain a high hit ratio. We further talked about temporal and spatial locality exploited by the buffer management algorithm to optimize main memory usage. In section 3.3, we discussed different conceivable architectures by inclusion of flash memory as part of a DBMS. We found out that the three-tier architecture is the best possible option to utilize flash memory for a DBMS cache.

4 Energy Efficiency and Performance

In previous chapter, we discussed about different kind of architectural styles. In this chapter, we will discuss in depth about 3TA and 2TA and find out which, among the two, is more energy and performance efficient. In 2-tier architecture we use DRAM as buffer/caching device and disk drive as primary storage device. In 2TA, as we increase the data size, the performance gets limited due to smaller and expensive DRAM used in buffering/caching device. This limitation in performance due to smaller DRAM won't have much improvement, even if, SSD is used as primary storage device. Hence, 3-tier architecture with extended cache device (using flash memory) is considered as more appropriate. In order to prove this, we have to conclude that the 3TA is not only performance efficient but also energy efficient.

4.1 Assumptions

The 3-tier architecture consists of 3 different layers, described below :

1. Top layer consists of DRAM as buffer device. This layer is capable of holding T_t pages.
2. Middle layer consists of flash memory as cache device. This layer is capable of holding T_m pages.
3. Bottom layer consists of disk drive as primary storage device. This layer is capable of holding T_b pages.

The number of pages stored at each level, increases from top-tier to bottom-tier and is given by following equation:

$$T_t \prec T_m \prec T_b \tag{4}$$

The hottest page is kept in the top-tier i.e. in T_t and T_m holds the hot pages that can't be retained in T_t due to size constraint. Hence, we need to come up with some kind of page replacement policies to swap pages between T_t and T_m. Whenever, a page fault occurs in T_t, the page is to be looked in T_m and if not found then should be loaded from T_b. In case of 2TA, T_m won't be present in the architecture and T_t will directly access T_b for all its requests.

4.2 Replacement Algorithms

In this section, we will discuss about the replacement algorithm for T_m only (as T_t and T_b are same as standard top and bottom layer of DBMS system). We will discuss following algorithms: Local (LOC) and Global (GLB) algorithms.

Before we proceed explaining the 2 algorithms, let us discuss few general goals of the algorithms. Both of the algorithms are used for maintaining a cache slot L_s=T_m in an LRU fashion. The cache slot uses single bit for representing clean/dirty state of the page. A dictionary (H) is maintained which maps the currently cached pages with corresponding cache slots.

Local (LOC) Algorithm This algorithm play's *(*locally) in T_m, without requiring information from T_t layer, in LRU fashion. As, LOC is not considering the content information of T_t, there is a possibility that some or complete data of T_t is replicated in T_m (but not vice versa because $T_m \geq T_t$).

For the read request of a page P, from a cache slots L_s using a directory H.

1. Search cache slot location C for page P in directory H
2. If C belongs to T_m then
   - Read the page P and serve the request.
   - Move C to MRU position of L_s.
3. Else
   - select the victim slot location V from LRU of L_s.
   - Page Q is the page stored at location V.
   - If page Q is dirty then
     - Read the page Q.
     - Flush it to T_b.
   - Read page P from T_b.
   - Move it to slot V.
   - Move this slot V to MRU position of L_s.
   - Update the new slot and location information in directory H for future use.

Global (GLB) Algorithm GLB is second algorithm for page replacement and was firsts introduced in [Zu04].This algorithm takes into consideration content of T_t along with T_m for creating a global logical LRU list called L_g. It takes care that no page is replicated between T_t and T_m (double caching) [KO08]. L_g consists of LRU list of the T_t at its MRU end and LRU list of T_m at its LRU end. Whenever, a page fault occurs in T_t, it looks for the page P in T_m. In case, if it does not exists in T_m the page will be loaded directly from T_b, without being copied into T_m.

Hence, whenever a page fault occurs in T_t, a page Q is evicted from T_t to T_m. The evicted page Q from T_t will become MRU page of T_m. Hence, we have to introduce a new function *evict* in T_m.

***evicting* a page P, a cache slots L_s, a directory H and bottom tier T_b.**

1. select the victim slot location V from LRU of L_s.
2. Page Q is the page stored at location V.
3. If page Q is dirty then
   - Read the page Q.
   - Flush it to T_b.
4. Store P at location V.
5. Move this slot V to MRU position of L_s.
6. Update the new slot and location information in directory H for future use.

Based on the properties of the 2 algorithms, GLB appears to be better in terms of Global cache hit count because of bigger effective buffer size (consists of T_t and T_m). But in GLB, the number of flash writes will be equivalent to number of cache evict operations. This will create problem for flash based secondary buffer device and will result in reduced lifespan and performance. Nonetheless, we can leverage upon the non-volatile nature of flash memory and can avoid performing flushing operation in T_m buffer memory at the time of shut down. This will not only speed up the shutting down time but will also help in enhancing the performance of DBMS by keeping the hottest pages pre-ready in buffer memory at the time of DBMS start up.

4.3 Experiment

In this section, we will see different experimental results to compare the performance and energy efficiency of 2-TA and 3-TA with LOC and GLB replacement algorithms. In the study, following assumption is taken into consideration while running the experiment for energy efficiency :

A1 The cost and power consumption of a storage media is linear to the their capacity in use.

This assumption may not hold for fine grain systems but it is true for coarser grained systems. All the experiment the program only communicate with T_t by sending the logical page request which is handled by buffer management policies in LRU fashion.

In the experiments, the size of the buffer pool b is scaled logarithmically. For 2TA, the size b is equivalent to T_t, but, for 3TA it is represented by T_t and T_m by following formula:

$$|T_m| = b \times s \tag{5}$$

and

$$|T_t| = max(1, \lfloor b - |T_m| \times (M_f/M_r + S_d/S_p \rfloor) \tag{6}$$

Where, M_f/M_r is the per-GB price ratio of flash to RAM; S_d is the byte size of dictionary H entry; S_p is the page size in bytes

s is used to scale the size of $|T_m|$ in case of 3TA. In the experiment, we will show the performance index by measuring *Virtual Execution Time* t_v, define by:

$$t_v = t_m + t_b \tag{7}$$

Here, t_m represents the device access time elapsed in reading or writing data from middle-tier T_m, further define by:

$$t_m = t_F R + t_F W = n_F R \times C_F R + n_F W \times C_F W \tag{8}$$

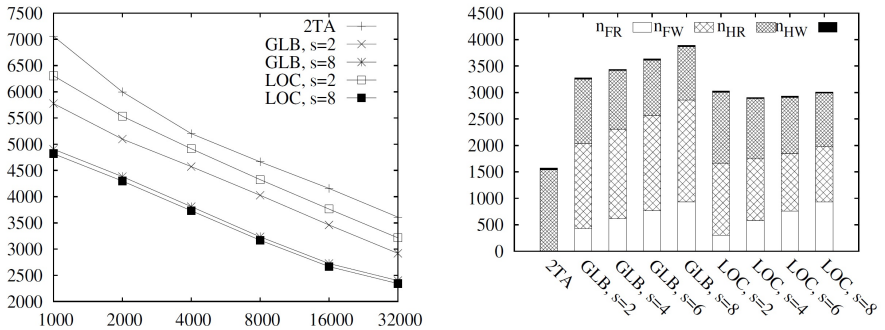
$t_F R$ and $t_F W$ are time taken in reading and writing from the flash device. $n_F R$ and $n_F W$ are number of flash read and write with average cost of read and write given by $C_F R$ and $C_F W$.

Similarly, t_b is given by:

$$t_b = n_H \times C_H \tag{9}$$

Where, n_H represents number of disk access and C_H the average latency of disk access.

Fig. 6. TPC-E trace performance



(a) t_v (sec) for each $b \in \{1000, ..., 32000\}$ **(b)** Device accesses (x1000) for $b = 1000$ (pages)

In figure 6(a), you can see that for t_v all 3TA configurations has significantly outperformed the 2TA *Virtual Execution Time*. In case of 3TA LOC for s=8, we can observe an improvement of 32% to 35% in *Virtual Execution Time*. This improvement is further described in figure 6(b), where majority of the 3TA page requests are getting served by T_m. We can further observe that in both LOC and GLB, by increasing the size of T_m using s, the number of buffer hits increases and consequently reduces the number of reads from disk.

In figure 7, shows the energy efficiency of 2TA and 3TA for b=100 based on assumption **A1**. The power consumption of T_t is given by :

$$P_t = |T_t| \times S_p \times \dot{P}_R \tag{10}$$

Here, $\dot{P}_R$ is the unit power of RAM. Using power values P_t, P_m and *Virtual Execution Time* t_v, we can calculate the energy consumption in all the setups. From the last column, we can deduce that 2TA consumes more energy than 3TA.

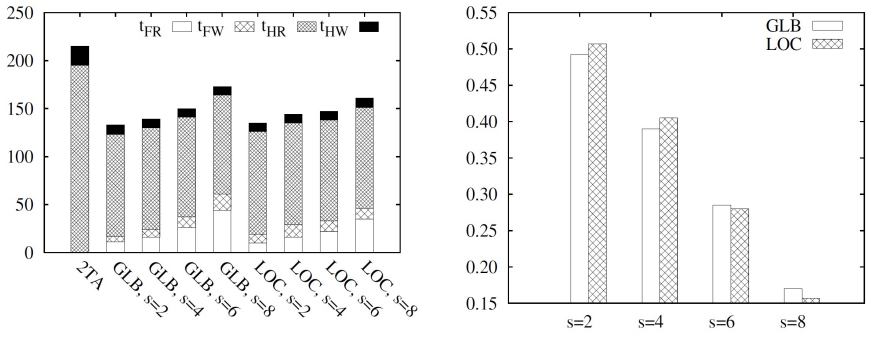
Fig. 7. Energy consumption of the TPC-E trace for b=1000

algo	s	$\|T_m\|$	$\|T_t\|$	P_m (mW)	P_t (mW)	$P_m + P_t$ (mW)	t_v (s)	E (J)
2TA		0	1000	0.000	4.121	4.121	7059	29.09
GLB	2	2000	799.02	0.014	3.292	3.307	5776	19.10
GLB	4	4000	598.05	0.029	2.464	2.493	5304	13.22
GLB	6	6000	397.07	0.043	1.636	1.679	5061	8.50
GLB	8	8000	196.09	0.057	0.808	0.865	4905	4.24
LOC	2	2000	799.02	0.014	3.292	3.307	6305	20.85
LOC	4	4000	598.05	0.029	2.464	2.493	5372	13.39
LOC	6	6000	397.07	0.043	1.636	1.679	5024	8.44
LOC	8	8000	196.09	0.057	0.808	0.865	4818	4.17

Furthermore, the power consumption reduces drastically for both LOC and GLB when the size of T_m is increased.

These observations can further be confirmed by running the experiment on a real device. The observations can be seen in figure 8.

Fig. 8. Statistics running the real-life trace for b=32000



(a) Device I/O breakdown (sec)

(b) Energy consumption relative to 2TA

4.4 Summary

From the experiments done on simulation and real environment, we can conclude that 3TA outperform 2TA in both power and performance efficiency. Furthermore, for 3TA we can conclude that for smaller value of s i.e., smaller size of T_m, GLB performs better. But, in case of large size of T_m, LOC is a better choice for replacement policy. In next chapter, we will study the impact of FTL in the performance measurement of flash device and also, an important problem called *CPM (cold page migration)*.

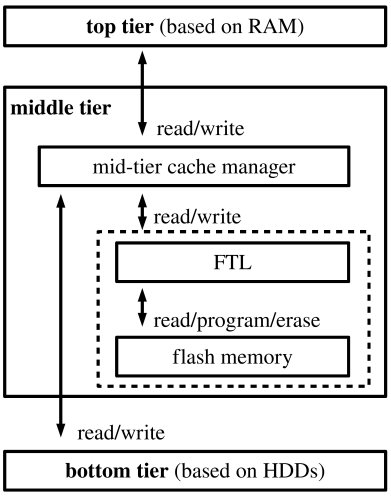
5 Inside View Of FTL

Till now, we have successfully derived the conclusion about how the flash memory usage can be leveraged in 3-tier architecture. In deriving the conclusion, we made our decision based on monitory cost and performance cost of DRAM over flash memory. Importantly, we also assumed that the FTL interface is available for the flash memory to be accessed as block device and we can apply the algorithms without any modifications. In this chapter, we will discuss the importance and impact of FTL in performance measurement and the different problems faced when accessing flash memory using FTL.

5.1 Revisiting FTL

We discussed the significance of FTL in section 2.3. In chapter 4, we made the assumption of accessing the flash memory indirectly using FTL. FTL enables the interface of flash memory to be transparent for middle-tier cache manager. This is shown in figure 9.

Fig. 9. Middle-tier cache manger accessing the flash memory transparently using FTL.



Flash memory has its own logical page addresses provided by FTL, called *FTL logical addresses*, and these logical addresses are used for addressing FTL logical pages. Now, to keep track of valid logical pages the FTL maintain a table $m_{FTL} : A_F \mapsto A_f$, where A_F represents FTL logical addresses and A_f represents physical addresses available on the device.

5.2 Problem

FTL helps in using flash memory transparently as middle-tier cache by providing external interface to cache manger(ref. figure 9). Hence, FTL gives the advantage of implementing the standard cache management algorithms directly on flash devices without bothering about its internal architecture. But, this brings in dependencies on different vendors because FTLs are proprietary solutions. This makes it difficult to standardize the performance of flash devices, as their performance depends on FTL (which to vendor specific). Also, the potentially expensive operation like, GC (Garbage Collection) is implemented by vendor and operate independently, leading to unpredictable response time for the operation.

The problem with GC is further explained in this section. There are 3 steps for GC:

1. Select the sets of garbage blocks. Each garbage block consists of valid/invalid pages.
2. Move all sets of valid pages from garbage blocks to another sets of free blocks and update the management information.
3. Erase the garbage blocks, which in return will create a free blocks.

Following is the illustration on how the steps will work :

1. A block of M pages got selected as garbage block using Step 1. Let us assume that there are v valid flash pages.
2. Step 2 will consume v free flash pages and will increase the total number of free flash pages by M-v.
3. Total cost of the operation will be $(C_{fr}+C_{fp})\times v+C_{fe}$, where $(C_{fr}+C_{fp})\times v$ is done by Step 2 and C_{fe} by Step 3.

The Ratio of v/M is called *block utilization*. The GC will be more effective and efficient if, block utilization is small. Hence, block utilization is important for garbage block selection. For high utilization or complete utilization the ratio will become 1, making GC ineffective and expensive. This will result in no free page generation and expensive GC operation.

Apart from the inefficient and expensive GC during high utilization of buffer memory, another problem called cold-page migration (CPM) contribute towards inefficiency of GC operation. Though, only hot pages are considered to be kept in buffer memory, FTL make sure that all valid pages should be accessible irrespective of the pages hot/cold state. These cold pages, though not required to be kept in buffer memory, contribute a lot towards number of valid pages v. Resulting in high block utilization ratio and hence, contributing towards inefficient GC operation.

Hence, the CPM creates two significant problem for use of flash memory in middle-tier :

1. Increases the cost of GC because of movement of cold pages along with hot pages.

2. The GC operation will result in less number of free pages and hence, result in frequent GC operations. More number of GC operations result in more number of erase operations(Step 3). This will reduce the life of flash device due to endurance limitation.

5.3 Savior: Logical Page Drop and Native Flash Access

The solution to *cold page migration* is given by 2 different approaches. The base principle is to drop the cold pages proactively, so that, GC can produce more number of free pages (due to reduced block utilization).

– **Logical Page Drop (LPD)** :
In LPD, we access the flash memory indirectly by using FTL interface and drop cold pages proactively while ignore them during GC operation. To achieve this proactive dropping of cold pages, a new operation called *delete* is introduce just like read and write. Job of *delete* is to turn the desired page into invalid page, so that, that page can be ignored during GC operation. LPD contains an address mapping table $m_{LPD} : A_b \mapsto A_F$, this table is used for locating the pages cached into the device. A_b represents the set of bottom tier addresses and A_F represents the set of FTL logical addresses. Each cache slot corresponds to exactly one FLA and also contains the corresponding dirty/clean state of the page. If the dirty page is selected for eviction, it must be first written back to the bottom-tier and then the delete instruction should be passed by FTL for the page.
In LPD, if we have no free page available we will evict and delete d pages from the cache using the LPD algorithm. The parameter d should be selected cautiously because it will determine the number of cold pages evicted from the cache to produce equivalent number of free pages. If we became too greedy and increased the value of d to drop more valid pages, the advantage would surpass the cost of loading them back from bottom-tier during page faults. This process of evicting d pages is called *page dropping* and will occur only after first cache replacement, indicating no more free flash pages available.
For a parameter d, a set F of free slots, a set S of occupied slots; on a cache hit the free page is selected by following algorithm:

1. If F is not empty then select one element from F and reduce number of slots by 1. Return the free slot.
2. Else
   - Select a cache slot v and remove the victim from S
   - Evict the page represented by slot v
   - Loop from 0 to d such that S is not empty (*Page Dropping*)
     * Select a cache slot s and remove the victim from S
     * Evict the page represented by slot s
     * Issue delete operation for page represented by slot s
     * add s to set of F
   - Return v

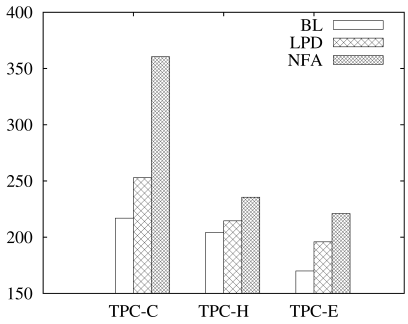
– **Native Flash Access (NFA)** :
In NFA, the cache manager directly accesses flash memory, removing FTL dependency for GC and other operations. The NFA cache manager is responsible for implementing all the basic operations and GC operation. The NFA cache manager maintains the table $m_{NFA} : A_b \mapsto A_f$ for mapping the BTAs (bottom-tier address) A_b to FPAs (FTL page address) A_f. In NFA, the block management information is represented in block management structure (BMS). The BMS contains the valid/invalid and dirty/clean state of the pages inside the block. These information are used during GC and cold page dropping operations.
In NFA, the free page is serve using following algorithm :
a *wp* pointer, set of free page F, watermark w_l, w_h

1. If the selected block is fully written then
   - set *wp* to the free page of the available free block
   - If $|F| \leq w_l$ then
     * while $|F| \prec w_h$ do GC
   - return *wp*
2. Else return $wp = wp + 1$

In this algorithm which provides free flash pages, few important properties needed to be explained beforehand. Whenever a request for free flash page arrives in the system, the write pointer *wp* provides the free flash page and then points towards the next free flash page in the block. If the block does not have any more free page left, then the next free block is selected and the pointer *wp* points to the first free page of the block. During this process, the system compares the value of $|F|$ to the lower watermark w_l. If the value is less then equal to w_l, then GC is triggered until $|F|$ becomes equal to higher watermark w_h. This helps in doing bulk GC processing.
Next is the NFA GC operation, which differs from typical FTL GC algorithm because of victim page selection policy (used for *page dropping*) and garbage block selection policy for performing GC. The dropping of victim pages happens when all the pages in the block are valid and the block is fully utilized. In NFA GC operation, the garbage block is selected by victim selection policy based on the last access time t. All the pages in the garbage block accessed earlier than t is then dropped and this threshold is passed to other garbage blocks for page dropping.
The NFA GC algorithm is given as follow:
for a page-dropping threshold t

1. Select the garbage block b
2. If all the pages in b are valid then
   - select a victim block b
   - set t as the last access time of block b
   - foreach page p in block b do
     * if last access time of p $\leq$ t then drop page p (make it invalid)
     * else move p to a free flash page
3. erase b and mark it as a free block

Fig. 10. Throughput (IOPS)

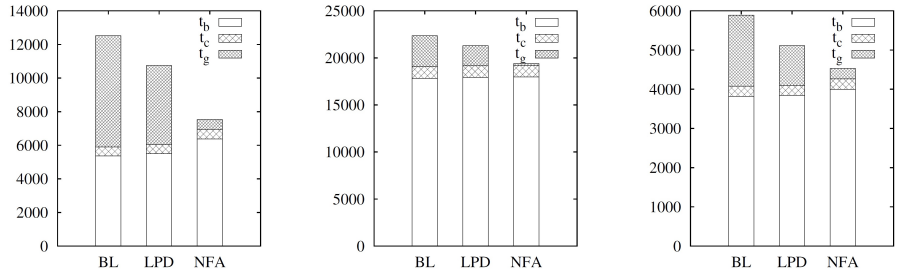


5.4 Experiment

In this section, we compare the performance of both the algorithm i.e., NFA ad LPD, with a baseline (BL) algorithm. The BL approach is indirect access of flash device without any additional operation like, delete or cold page drop. In this experiment we will specify the value d for LPD as 1024. We have a top-tier with buffer pool of 10000 pages and serving the request based on LRU policy. Whenever, we have a page fault at top-tier, it reads and writes the data from the middle-tier, which will help us in calculating the performance of three approaches(TB, NFA, and LPD). To analyze the performance, we uses the throughput as the criteria. Throughput is given by N/t_v, i.e, N number of request served in t_v time. t_v is given by $t_v = t_m + t_b$ (as explained in section 4.3).

The performance of three approaches is shown in figure 10 by running the TPC-C, TPC-H and TPC-E traces. Furthermore, the breakup of the execution time is represented by figure 11, for all the 3 traces. Here, $t_m = t_g + t_c$ where, t_g is the time taken for garbage collection and t_c is the time taken for normal caching operation.

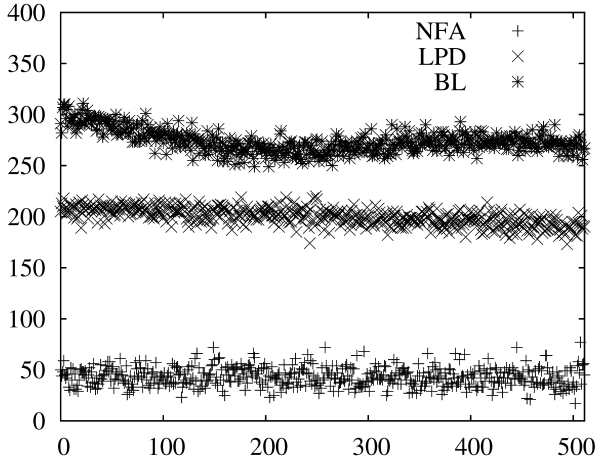
Fig. 11. Breakup of execution time into garbage collection time t_g, caching operation t_c and disk access time t_b



(a) TPC-C **(b)** TPC-H **(c)** TPC-E

From the experiment result, we can clearly see that both the algorithms i.e., NFA and LPD, significantly reduce the time taken for GC. If we further analyze the pattern of wear-leveling (explained in section 2.3), we can see that in figure 12 it has been taken care of automatically.

Fig. 12. The number of erase operation for each block. where, x-axis represent the block number



5.5 Summary

In this chapter, we studied the role of FTL in performance analysis of a flash memory. We analyzed that, the common problem associated with indirect use of flash memory using FTL called CPM, can significantly impact the device performance. To overcome such a problem, two approaches were discussed, one is by direct access of the flash memory without using FTL (NFA) and other is by indirect access of the flash memory (LPD). By running the traces, we analyzed that both the algorithms enhanced the performance of the middle-tier significantly by reducing the GC operation time. We also analyzed that the use of these 2 policies automatically resolve the wear-leveling issue faced in flash based devices. Out of the two approaches, we found that Native Flash Access is better and hence, can conclude that direct access of the flash based memory yield better result than the indirect approaches.

6 Conclusion

In this report, we discussed in detail about the flash memory architecture and how it can be used for DBMS systems. We looked for different architectural

scenarios and algorithms to improve the performance and efficiency of the flash memory. We discussed about the 2TA and 3TA approaches, for using flash memory as main storage device and as middle-tier for caching respectively. Based on the experimental results, we concluded that 3TA approach yield better results in terms of energy efficiency and performance. In 3TA, we discussed two cache management algorithms namely : Local algorithm (LOC) and Global algorithm (GLB). In LOC, we only keep account of the pages stored in middle-tier and in GLB, we consider top tier data while computing the global mapping table. GLB is the efficient when we have smaller middle-tier flash memory compared to top-tier DRAM and LOC is better when we have bigger middle-tier flash memory compared to DRAM.

We also discussed about the significance of indirect (using FTL) and direct access of the flash memory by the cache management algorithms. With FTL we have the issue of CPM (cold page migration) during GC, which hampers the performance of the flash memory significantly. Also, FTL is a proprietary application and its performance varies from vendor to vendor and can not be used for standardizing the result. Hence, we discussed two different approaches namely : Logical Page Drop (LPD) and Native Flash Access (NFA). These two approaches drop the valid cold pages in advance such that, during GC operation these pages can be ignored and more free pages can be generated. While, LPD access the flash memory indirectly (using FTL), the NFA accesses the flash memory directly and implements out-of-place update and garbage collection operation. In the result, we can see that NFA performs better.

Hence, we can conclude that using flash memory in middle-tier as cache device, will not only provide us better efficiency in terms of performance, but also, in terms of energy consumption.

7 References

References

[RB09] D.Roberts, T.Kgil, et al.: Integrating NAND device onto servers. Communications of the ACM, vol. 52, no. 4, pages 98-103, 2009.

[KM07] J.Koomey: Estimating total power consumption by servers in the US and the world. http://sites.and.com/de/Documents/svrpwrusecompletefinal.pdf, February 2007.

[ID08] The diverse and exploding digital universe-(an IDC white paper). http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf, March 2008.

[AR02] ARIE TAL, M-Systems Newark, CA: NAND vs. NOR flash technology.The designer should weigh the options when using flash memory-(Article). *http* : $//www.electronicproducts.com/Digital_ICs/NAND_vs_NOR_flash_technology.aspx$,, January 2002.

[BY10] Byung-Woo Nam, Gap-Joo Na and Sang-Won Lee: A Hybrid Flash Memory SSD Scheme for Enterprise Database Applications, April 2010.

[TD12] TDK Global: SMART Storage Solution for Industrial Application (Technical Journal), January 2012.

[GA05] Eran Gal and Sivan Toledo, School of Computer Science, Tel-Aviv University: Algorithms and Data Structures for Flash Memories, January 2005.

[IO09] Ioannis Koltsidas and Stratis D. Viglas, School of Informatics, University of Edinburgh: Flash-Enabled Database Storage, March 2010

[KO08] I.Koltsidas & S.D.Viglas. The case for flash-aware multi-level caching. Rapport technique, University of Edinburgh, 2009.

[SE10] Seongcheol Hong and Dongkun Shin, School of Information and Communication Engineering Sungkyunkwan University Suwon, Korea : NAND Flash-based Disk Cache Using SLC/MLC Combined Flash Memory, May 2010.

[Zu04] Y.Zhou,Z. Chen, et al. Second-level buffer cache management. IEEE Transactions on parallel and Distributed System, vol.15, no. 6, pages 505-519, 2004.

[TH05] Theo Härder DBMS Architecture - The Layer Model and its Evolution. March 2005

[Yi12] Yi Ou Thesis report, Caching for flash-based databases and flash-based caching for databases. August 2012

List of Figures

List of Tables