# Performance Analysis of FTL Schemes

M. N. Kale
PG student,
Department of IT- PDVVP COE,
Pune University, Ahmednagar.

A. S. Jahagirdar
Asst. Professor,
Department of IT-MIT COE,
Pune University, Pune.

## ABSTRACT
Today Nand Flash memory is not only used in hand hold electronic devices but also as secondary storage medium. It serves as an alternative to Hard Disk Drives (HDDs) in the form of Solid State Dives (SDDs). However, unlike HDD flash memory does not support in place update, i.e. for updating data, old data can not be replaced by new data. Data can be written only at clean i.e. already erased place. This is called as erase before update. This erase before update nature of Nand Flash memory is kept hidden with the help of a functionality called as address mapping or address translation. Many efforts for optimizing the working of address mapping schemes have been done by different research workers. Though various schemes are designed and proposed but there is no literature available providing mathematical computations comparing the performance of the various mapping schemes in the form of time complexity. In this paper we have tried to find out the comparative cost of block merge operation required during garbage collection for some representative mapping schemes like BAST [9] and FAST [7].

This paper also presents a review of all these schemes and presents a comparative trade offs among all these major schemes. The paper is divided into five sections: section 1 is introduction of Flash memory, section 2 describes various mapping schemes and presents their comparative performance, Section 3 does the conclusion. Section 4 is acknowledgement and section 5 describes the future scope.

## 1. INTRODUCTION

### 1.1 Flash Memory Organization
Nand Flash Memory (simply called as Flash hence after) has became a very common storage medium nowadays due to it's low cost, light weight, low power consumption and faster access. Flash is a kind of Electrically Erasable Read Only Memory (EEPROM). Flash has two major types; NOR Flash which is directly accessible and NAND flash which is addressable through a single 8-bit bus, that is used for both data and addresses. The control lines are separate. The erase (clean) operation set each bit in Flash to one '1'. This bit can be reset to zero '0' by write operation. Flash has many desirable characteristics, however, a major drawback of the flash memory is that it does not allow in-place update (i.e., overwrite operation). Flash chips are organized in blocks, where each block is comprised of pages; 32 pages of 512+16 bytes each for a block size of 16 KB, 64 pages of 2,048+64 bytes each for a block size of 128 KB [12], each of which has an extra "out of band" storage space, intended to be used for error correction codes (ECC) [10]. Every page carries a data part for user data and a spare part for error correction code associated with the user data, such as mapping and ECC information. Data part size is a multiple of the sector size (512 bytes), and the spare part is typically 16 bytes for each sector in the data part [4]. The smallest unit of read and write operation in Flash is page. Data is stored in flash in an array of blocks. Each block spans 32-64 pages. In flash a page is the smallest unit of read, write operation [5]. Page write operations in a flash memory must be preceded by an erase operation. Within a block pages need be to written sequentially. Update operation becomes more critical due to the different access units used for read and erase operation in Flash. The unit for read, write operation is page, whereas the unit for erase operation is block. The typical access latencies for read, write, and erase operations are 25 microseconds, 200 microseconds, and 1500 microseconds, respectively. While a flash device can read any of its pages, it may only write to one that is in a special state called *erased*. This is called as erase before update characteristics of Flash. Flashes are designed to allow erases at a much larger spatial unit than pages since page-level erases are extremely costly [3]. As a typical example, a 16GB flash product from Micron has 2KB pages while the erase blocks are 128KB. Before erasing a block, the pages containing valid data need to be moved to some clean block. Due to this, block erase operation accompany with a lot of page read and write operations. Thus the erase operation becomes critical in view of performance.

Another undesirable characteristic of Flash is; after a limited number of erase operations (10K-100K) [8], Flash undergoes wear out problem, reducing the lifetime of Flash. To reduce this wear out problem it is necessary to ensure that no particular block in Flash reaches to this wear out limit, as far as possible. For this erase operations need to be evenly distributed around the Flash [13]. Flash is written by loading the required data into an internal buffer one byte at a time, then issuing a write command [10].

### 1.2 FTL: Flash Translation Layer
To get around the erase before update, a special purpose software called as Flash Translation Layer (FTL) is implemented inside flash storage medium. FTL resides inside a small controller, mounted inside the flash storage medium. FTL make the flash device to act like a block storage device such as magnetic disk drive as shown in Figure 1. FTL allows Flash to be accessed as a set of standard 512Byte, 2K or 4K logical sectors or pages. That is FTL allows an array of NAND flash to be addressed as a set of standard 512 byte, 2K or 4K logical sectors or pages. The main task of FTL is to hide the erase before update nature of flash. FTL does this with the help of a functionality called as address mapping or address translation. Address mapping maps two address domains: the logical address and the physical flash address. It does so by maintaining a mapping table of virtual addresses received from upper layers (e.g., those coming from file systems) to physical addresses in the Flash. For any page update FTL works in following manner,

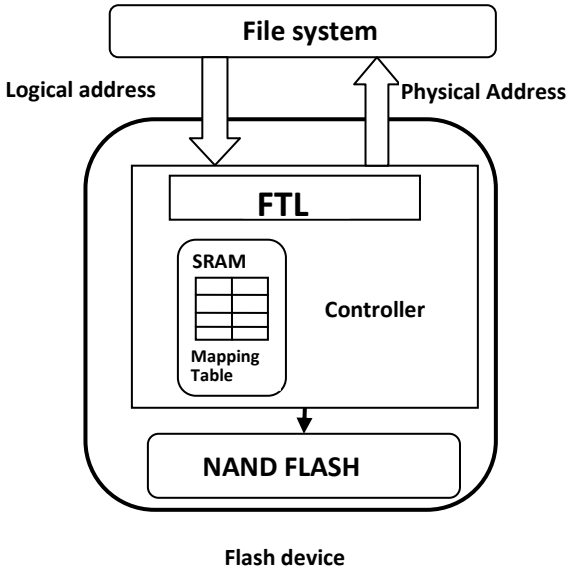


**Figure 1: Flash Memory Interface to file system**

It takes the page number received from the upper layer like operating system called as logical page number, from the page map table it then locates the physical page number for that logical page; if that page is already written then it finds another clean page, and copies the updated data there [1]. That is, FTL emulates disk-like in place update for a logical page number (LPN) by writing the new page data to a different physical page number (PPN). It maintains a mapping between each LPN and its current PPN. Finally, it marks the old PPN as invalid for the later garbage collection. Thus, FTL enables existing application to use flash memory without any modification. FTL emulate the functionality of a normal block device by exposing only read/write operations to the upper software layers. This update type is called as out of place update.

Thus an out-of-place update involves: (i) choose an already erased page, (ii) writes to it, (iii) invalidates the previous version of the corresponding page, (iv) finally updates mapping table to reflect this change. The out-of-place update requires Flash to employ a garbage collection (GC) mechanism. Because after certain number of page writes the number of invalid pages will increase reducing the free pages available for further data write or update operations. The role of the GC is to reclaim the data blocks containing invalid pages by erasing these blocks and relocating any valid pages within these blocks to some new locations [4]. Evidently, FTL crucially affects Flash performance. One of the main difficulties the FTL faces in ensuring high performance is the severely constrained size of the *on flash SRAM-based cache* where it stores its mapping table. For example, if the capacity of a device is 4GB, the page size is 2KB, and each mapping entry is 4 bytes in size, the memory required for the mapping table is 8MB [2]. The performance of Flash memory mainly depends on three factors: address translation time, the look up time to search the requested data, Garbage Collection.

# 2. TYPES OF MAPPING SCHEMES

Depending upon the spatial unit of mapping, mapping schemes are classified two categories [6]. These are; Page mapping, Block mapping. And later on, a hybrid mapping scheme proposed by D. Park [9] came into existence. This hybrid scheme is combination of the earlier two schemes. All these three schemes are discussed in following subsection.

## 2.1 Page Mapping

In page mapping [7] scheme the smallest logical unit that FTL uses for address translation is a page. A page is made up of certain number of smallest units called as sector. Where a sector is a smallest physical unit in which data actually can be read or written to flash memory.

Generally a page is made up of single sector. In page mapping scheme, during any read or write operation, the address of the page containing the data to be read or updated is passed from the application that is running on the top of Flash memory, to the Flash memory device. This address is called as the logical page number. The actual physical page containing the information corresponding to this address may be located somewhere else in the Flash. That is the logical page may be mapped to a different physical page in Flash. This mapping information is kept in the form of tables called as mapping tables. For page mapping scheme the page map table consists of entries storing the physical page number of the corresponding logical page number. The index of the entry denotes the logical page number and the value stored in the entry indicates the corresponding physical page number as shown in figure 2. For example one want to see: in which physical page the logical page 10 is located, then just go to the page map table entry having index 10 and check the value stored there, say it contains the value 17. Then this is the actual physical page number of the flash memory, which contains the data corresponding to the logical page number 10.

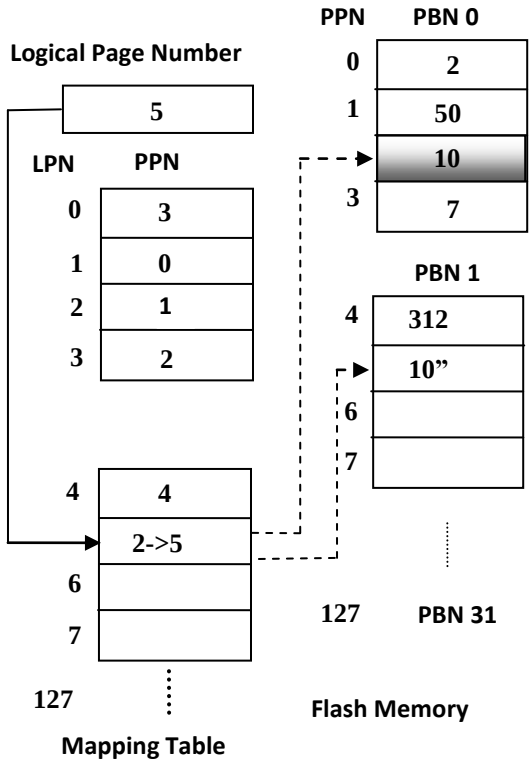


**Figure 2: page mapping scheme**

Advantage: This mapping technique provides the flexibility that for update operation any logical page coming from the top layer can be mapped to any clean physical page available

in the flash memory in any block [11]. Thus there is no necessity of aligning the page at the same physical offset within a block in Flash memory as that of its given logical offset. But it suffers from a critical drawback which prevents it from becoming a scalable system i.e. the size of page map tables increase with the size of Flash Memory. For page mapping scheme the page map table consists of entries storing the physical page number of the corresponding logical page number. The index of the entry denotes the logical page number and the value stored in the entry indicates the corresponding physical page number

**Table 1. Measures of page mapping scheme**

| | |
|---|---|
| **Garbage collection cost** | Block Erase is done when a block is completely utilized. |
| **RAM requirement** | Proportional to flash size |
| **Search time** | Not required |
| **Usefulness** | Useful in case of strict time requirement |

## 2.2 Block mapping

In block mapping [7] scheme, the smallest logical unit that the FTL uses for address translation is a block. A block is also smallest erase unit for flash memory. For block mapping scheme the block map table consists of entries storing the physical block number of the corresponding logical blocks. The index of the entry in the map table denotes the logical block number and the value stored in the entry indicates the corresponding physical block number as shown in figure 3. In this scheme the top layer passes the address of the data which is to be updated. This address consists of the block number and the page offset i.e. page number within that block. This is called as logical address. Using logical block number, corresponding physical block number is looked in the map table. The data then is written in that physical block at the same page offset, as that of it's offset in the given logical block [11]. If data is already written at that offset in the physical block, then a new clean block is allocated and the data which is to be updated along with the data in the old block are written in the new allocated block.

Advantage: this mapping scheme requires less space for storing the mapping information. The disadvantage is that the garbage collection overhead is more, as it has to perform a block copy and a block erase for a single page update.

**Table 2. Measures of block mapping scheme**

| | |
|---|---|
| **Garbage collection cost** | Block Erase is done for a single page update |
| **RAM requirement** | Proportional to block size and flash size |
| **Search time** | Not required |
| **Usefulness** | Useful in case of read dominant access pattern |

## 2.3 Hybrid mapping scheme

In hybrid mapping scheme the total available blocks of flash memory are divided into two types: log blocks also called as update blocks, this entire set of log blocks is called as log buffer. Data blocks or primary blocks, the entire set of data blocks is called as data buffer. This combines the page mapping for data blocks and block mapping for log blocks. That is whenever an update to some data is to be done, the data page to be updated is appended sequentially in the corresponding log block and the mapping table is updated accordingly. The log blocks are very less in number 3% of total data blocks. The advantage of the scheme is that it requires less space for storing mapping information, just like block mapping scheme and also requires less overhead of garbage collection as that of page mapping scheme. Based on the hybrid mapping schemes various mapping schemes have been proposed and various state of art FTLs has been implemented.

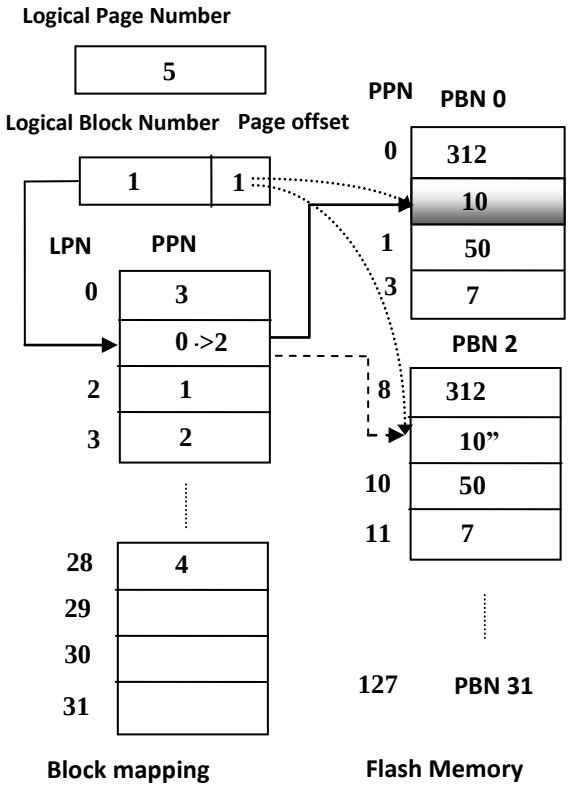


**Figure 3: Block mapping scheme**

Most representatives of them have been analyzed in next section by presenting time complexity computations. The following context tries to derive the time complexity of merge operation. Merge operation merges more than one data blocks into a single block with the intention to reclaim some data blocks. The computations presented here are derived for the two representative schemes, BAST and FAST.

### *2.3.1 Block associative sector translation layer*

BAST [9] scheme is a hybrid scheme proposed by Kim. It consists of two types of blocks: data blocks mapped by block mapping and log block that act as update buffer, mapped by page mapping. A separate log block is associated with each data block. They are less in number than the data block i.e. 3% of total data block. The proposed merge cost of this scheme is calculated as follows: considering, there are total T blocks in the complete Flash space. Then assuming 3% log blocks, total log blocks will be 3T/100, considering random write pattern it is clear that after 3T/100 number of random write requests from 3T/100 different data blocks, there is need of free blocks, Hence to free all the 3T/100 log blocks number of read and writes required will be:

((3T/100 *N) Read + ((3T/100) * N) Write + ((3T/100) + (3T/100)) Erase (data and log blocks) operations, where N is

number of pages per block. These computations gives the merge cost required after every $3T/100^{th}$ random write request where Log blocks are considered 3% of the total blocks. Thus in random write pattern, log blocks are required more frequently and the system goes under the deficiency of log blocks, this problem is called as log block thrashing [6] and initiates a merge operation. But the data lookup complexity of BAST scheme is less as compared to other scheme as each log data block has only one log block associated with it. Therefore no search cost is involved. Only one entry it has to examine in the page map table, i.e. the page number of the given page. RAM requirement of BAST is also less as the data blocks are block mapped. Only the page tables of log blocks are to be maintained in RAM and log blocks are very few in number hence RAM space for keeping mapping information is less.

**Table 3. Measures of BAST scheme**

| | |
|---|---|
| **Garbage collection cost (Worst case: considering number of random requests = number of log blocks)** | ((3T/100)*N )Read + ((3T/100)*N) Write + (2*3T/100) Erase<br>T: total blocks in flash<br>N: number of pages/block<br>Log block: 3% of T |
| **RAM requirement** | Less, Proportional to number of log block |
| **Search time (worst case)** | Time to search the page map table of a single log block |
| **Usefulness** | In case of sequential read write and update pattern |

### *2.3.2 Fully associative sector translation layer*

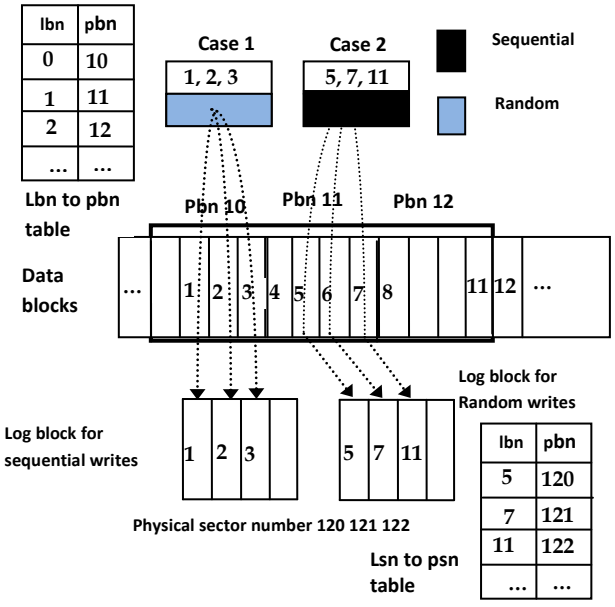


**Figure 4: FAST Scheme**

FAST[7] scheme, proposed by Lee is a hybrid scheme. It consists of log blocks and data blocks as shown in figure 4. All the data blocks are mapped using block mapping scheme. Log blocks are mapped using page mapping. The scheme uses two types of log blocks: a single sequential log block dedicated to accommodated sequential write operations. And another kind of log blocks called as random log blocks are dedicated for random writes. Both type of log blocks are shared by all the data blocks and data from a single data block can scatter over many log blocks. Hence, name fully associative. The advantage of the scheme is that it does the better utilization of space as hence garbage collection overhead is less. Again the worst case merge cost of this scheme can be calculated by the same method: considering, there are total T blocks in the complete Flash memory space. Then assuming 3% log blocks, total log blocks will be 3T/100, considering random write pattern and K as the associativity of log block i.e. a Log block can accommodate data from K different data blocks, it is clear that after 3T/100*K number of random page write requests from 3T/100*K different data blocks, there is need of free blocks Hence to free all the 3T/100 log blocks number of read and writes required will be:

((3T/100)*K*N) Read + ((3T/100) * K*N) Write + ((3T/100) + (3T/100)) Erase (data and log blocks) operations, where N is number of pages per block. This computations gives the merge cost required after every $(3T/100*K)^{th}$ random write request, where Log blocks are considered 3% of the total blocks.

It can be observed that this cost is less than the cost derived for BAST scheme seen earlier. However Lookup complexity of FAST is much larger than BAST, as in this scheme the data structures maintaining the mapping information are not simple, due to associativity of data and log blocks. Therefore for doing a lookup operation all the log blocks need to be searched because in the mapping table the index does not indicate the logical page number, but instead it indicates the physical page number and the value stored at that index entry indicate the logical page number. Hence for lookup of a logical page number linear search is performed, a time consuming operation. For example if the capacity of the storage flash under consideration is 4GB, the block size is 128KB, the page size is 2KB, and the number of log blocks is 320 (1% of the entire space). Considering, 64 pages per block, it has to examine total 64*320 entries in worst case, if it has to examine mapping information of all log blocks. Considering the controller speed as 100 Mhz, and examining an entry suppose takes 3 cycles, load compare and increment, finding the location of a page will require 600µs, which is 3 times longer than write latency of a NAND page. The RAM requirement is less and is same as that of the BAST scheme.

**Table 4. Measures of FAST mapping scheme**

| | |
|---|---|
| **Garbage collection cost (worst and best case : considering number of random requests = K* number of log blocks)** | ((3T/100)*K*N) Read + ((3T/100) * K*N) Write + (2*3T/100) Erase<br>T: total blocks in flash<br>N: number of pages/block<br>Log block: 3% of T |
| **RAM requirement** | Less, Proportional to num. of random log blocks |
| **Search time (worst case)** | Time to search the page map table of all log blocks. |
| **Usefulness** | In case of random read write and update pattern |

### *2.3.3 Shared Block associative sector translation*

SBAST [2] scheme is similar to FAST except one additional constraint has been imposed is that each data block is associated with only one log block, this is in contrast to the FAST scheme. The advantage of this scheme is that the data structures storing the mapping information becomes simple

shown in figure 5, where PPI is physical page index i.e. offset or page number within a block. For a page lookup operation only the page map table associated with a single log block is to be searched Hence Lookup cost gets reduced than that of FAST, whereas merge cost does not get much affected and remains as that of FAST.

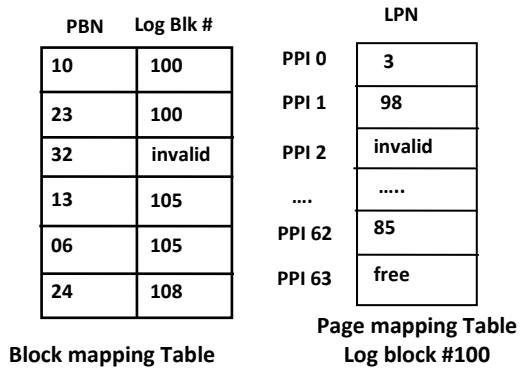


**Figure 5: SBAST page map table**

The RAM requirement of this scheme is also less, as that of the above mentioned three schemes.

**Table 5. Measures of SBAST mapping scheme**

| **Garbage collection cost** | Better than BAST |
|---|---|
| **RAM requirement** | Same as that of BAST |
| **Search time** | Time to search a page map table of a log block |
| **Usefulness** | Useful in case of random read, write and update pattern |

### *2.3.4 Locality Aware Sector Translation Layer*

LAST [6] is a combination of FAST and BAST schemes. Unlike other schemes it divides its log blocks again into two types i.e. sequential log blocks and random log blocks which are mapped using block mapping scheme. It handles sequential updates using sequential log blocks and handles random updates using random log blocks which are mapped using page mapping scheme. It uses a locality detector to separate the random and sequential updates. This scheme is having merge cost less than FAST scheme because some of the full merges are converted to switch merges due to sequential log blocks. The lookup complexity is less than FAST, as it also involves the mapping like BAST scheme for sequential log blocks and mapping like FAST scheme for random log blocks.

**Table 6. Measures of LAST mapping scheme**

| **Garbage collection cost** | Better than FAST |
|---|---|
| **RAM requirement** | Same as that of FAST |
| **Search time** | Time to search a page map tables of all log block |
| **Usefulness** | Useful in case of random read, write and update pattern |

BAST has to examine only one entry in map table for lookup. RAM requirement is less than that of the FAST and BAST scheme.

### *2.3.5 Demand paged mapping Scheme (DFTL)*

DFTL [10] Recently a page level based mapping scheme called Demand paged Flash translation Layer i.e. DFTL has presented by Gupta. It has much better performance. It maintains two types of tables in SRAM, namely, Cached Mapping Table (CMT) and Global Translation Directory (GTD). CMT stores only a small amount of page mapping information like a cache for a fast address translation in SRAM. GTD keeps track of all scattered page mapping tables stored in flash. This scheme is paradigm shift from the previous hybrid mapping schemes. It is fundamentally a page mapping scheme. It maintains the complete page map table in the flash memory itself. Hence its RAM requirement is much less the other schemes.

**Table 7. Comparative analysis of various FTL schemes**

| **FTL SCHEME** | **Merge cost of blocks during GC** | **Lookup Performance** | **RAM requirement** | **Mapping granularity** |
|---|---|---|---|---|
| Pure page level | N/A | No lookup cost | Much more | Page |
| Pure block level | Much more | No lookup cost | Very less | Block |
| BAST | More | Lesser | Less | Page for log blocks<br>Block for data blocks |
| SBAST | Less | less | Less | Page for log blocks<br>Block for data blocks |
| FAST | Lesser than SBAST | Much more | Less | Page for log blocks<br>Block for data blocks |
| LAST | Lesser than FAST | Much more | Less | Page & block for log blocks<br>Block for data blocks |
| Demand paged | N/A | Lesser | Lesser | Page |

## 3. CONCLUSION

The conclusion is summarized in the form of a table; table 7. It shows the comparative trade offs among various FTLs and analysis of various FTLs that has been calculated in above section. It is observed that the page mapping scheme has the greatest advantages except it's drawback of high memory consumption and hence the DFTL tries to retain this advantage of page level mapping scheme, as well as found a solution to resolve the need of high memory demand.

## 4. ACKNOWLEDGMENTS

Our special thanks to the guide who have contributed towards development of this paper as they were continuously insisting and inspiring to frame a review paper presenting the comparative analysis of various FTL schemes.

## 5. FUTUTE WORK

In proposed future work the reduction in data lookup time is planned. To do this the advanced data structures like hashing and search trees should be implemented to store the mapping information, so that the search time will get reduced

## 6. REFERENCES

[1] Yeonseung Ryu, "A Flash Translation Layer for NAND Flash-Based Multimedia Storage Devices", IEEE TRANSACTIONS ON MULTIMEDIA, pages: 563-572, VOL. 13, NO. 3, JUNE 2011

[2] Shin I.: Light weight sector mapping scheme for NAND-based block devices, IEEE Transactions on Consumer Electronics, pages: 651 – 656, May 2010, ISSN: 0098-3063 Volume: 56 Issue:2

[3] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," in ASPLOS, 2009.

[4] Aayush Gupta Youngjae Kim Bhuvan Urgaonkar "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings" Computer Systems Laboratory, Department of Computer Science and Engineering. The Pennsylvania State University, University Park, PA 16802, Technical Report CSE-08-012August 2008

[5] Dongchul Park, Biplob Debnath, and David Du "CFTL: A Convertible Flash Translation Layer with Consideration of Data Access Patterns", Technical Report Department of Computer Science and Engineering University of Minnesota September 14, 2009.

[6] S. Lee, D. Shin, Y. Kim, and J. Kim. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems, in Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED2008), February 2008.

[7] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song. A Log Buffer based Flash Translation Layer Using Fully Asso-ciative Sector Translation. IEEE Transactions on Embedded Computing Systems, 6(3):18, 2007. ISSN 1539–9087.

[8] J. Kang, H. Jo, J. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In Proceedings of the International Conference on Embedded Software (EM-SOFT), pages 161–170, October 2006. ISBN 1-59593-542-8.

[9] Chung, D. Park, S . Park, D. Lee, S. Lee, and H. Song. System Software for Flash Memory: A Survey. In Proceedings of the International Conference on Embedded and Ubiquitous Computing, pages 394–404, August 2006.

[10] JFFS : The Journalling Flash File System David Woodhouse Red Hat, Inc. dwmw2@cambridge.redhat.com

[11] Yang Hu "Achieving Page-Mapping FTL Performance at Block-Mapping FTL Cost by Hiding Address Translation(HAT)" Huazhong University of Sci. & Tech. China

[12] Micron Technical Report (TN-29-07): Small-Block vs. Large-Block NAND Flash Devices. Technical Report (TN-29-07): Small-Block vs. Large-Block NAND Flash Devices.

[13] Micron Technical Report (TN-29-07): Small-Block vs. Large-Block NAND Flash Devices. Technical Report (TN-29-07): Wear-Leveling Techniques in NAND Flash Devices