Efficient Multilevel Cache Design for Solid State Drive's

Bakhtiar Kasi, Mumraiz Kasi, Riaz UlAmin
Faculty of Information & Communication Technology , BUITEMS, Quetta, Pakistan,
Corresponding Author: bakhtiar.kasi@buitms.edu.pk

Abstract -- **Flash memory-based Solid-State Drives (SSDs) are becoming popular as the storage media in domains ranging from laptops and embedded systems to enterprise-scale storage systems. The main reasons are SSDs durability and low energy consumption. Performance behavior of SSDs differs from those of magnetic disks. However, SSDs possess poor random write performance because of the erase-before-write problem. The cache memory has multiple novel features including advanced support for performance monitoring, data pre-fetching, and coherency. In this research, we have incorporated multi-level caching with solid-state drives. We evaluated our technique using the standard state-of-the-art DiskSim simulator. We found a significant reduction in number of writes with multi-level caching. The overhead was comparable.**

***Keywords*- SSD, Cache, Multilevel, DiskSim, LRU**

Date Received 20 May 2019
Date Accepted 22 June 2019
Date Published 5 July 2019

I. Introduction

DUE to several reasons, flash memory is rapidly becoming an important and promising technology for the next-generation storage. Some of the reasons are: (i) low access latency, (ii) low power consumption, (iii) higher resistance to shocks, and (v) increasing endurance. A lot of research done in the past decade has focused on improving the performance and reliability of flash devices and their associated drivers and software [7, 5, 9, 18, 19].

In general, flash devices are made of NAND and NOR technologies [20]. The NAND-type flash memory may be written and read in blocks; generally smaller in size than the entire device. NOR-type flash allows a single machine word (byte) to be written to location previously erased. NAND-based flash devices have emerged as a more acceptable candidate in the storage market. One of the main reason behind this is the logical gate structure of NOR and NAND. If case of NOR, gates are connected in parallel where as in case of NAND it's serial. As parallel setup occupies more on board are compared serial setup, NAND based SSDs are more preferred.

The NAND-flash based solid-state storage devices (SSDs) can produce exceptional bandwidth and random I/O performance that is in orders of magnitude better than that of rotating disks. Moreover, SSDs offer both a significant savings in power budget and an absence of moving parts, improving system reliability.

Flash memory-based SSDs exhibit much better performance for random reads compared to hard disks because NAND flash memory does not have seek delay. In a hard disk, the seek delay can be up to several milliseconds. For sequential read and write requests, an SSD has a similar or better performance than a hard disk [1]. However, SSDs exhibit worse random writes due to the unique physical characteristics of NAND flash memory. It is called the *erase-before-write* property of flash memory

A NAND flash memory chip has several blocks that can be erased independently. Each block has a fixed number of pages where data can be written to or read from. Before data can be written to or read from. Before data can be written to an already used page, the block containing the page must be erased as overwriting is not allowed. A typical block size and page size is 16~256 KB and 0.5~4KB, respectively. Block erase takes around 1.5~2 milliseconds, while reading and writing page takes tens of microseconds and hundreds of microseconds respectively [10,11].

Following are couple of systems issues which are relevant to SSD performance:

Parallelism: To achieve optimal performance, the bandwidth and operation rate of any given flash chip is not sufficient. Hence, memory components must be coordinated to operate in parallel.

Workload management: Performance highly depends on workload. For example, architectural designs that produce good performance under sequential workloads may not benefit workloads that are parallel, and vice versa.

Write ordering: Small, randomly-ordered writes are especially tricky as far as NAND flash based SSDs are concerned.

Data placement: Careful placement of data across the chips of an SSD is critical for load balancing and wear-leveling.

To improve the average performance for data accesses, Hard-Disk Drives (HDDs) typically include an on-board cache based on DRAM technology. In most HDDs, the cache acts primarily as a buffer that matches the speed of the I/O interface to the slower access speed of the hard disk platters. The cache buffer might also incorporate some advanced features such as pre-fetching, in order to further reduce the average read access time. For writes, the operating system (OS) is given the illusion of a fast access time by signaling the completion of a write operation as soon as data is accepted into the cache, before the write is propagated to the hard disk platters.

Due to the constraint that page overwrite is forbidden, researchers have tried to solve this problem of *erase-before-write* problem by two different angles. Either by designing sophisticated software techniques such as Flash Translation Layer (FTL) [12, 14] or by designing flash-aware buffer management algorithms (e.g. CFLRU, BPLRU, PUD-LRU) [16,17, 19]. In this paper, we have turned our focus on to the write-cache buffer management system. We have tried to implement a two-level cache buffer management system on top of the FTL layer of a typical SSD.

II. BACKGROUND

Flash memory devices are the current de facto media adopted in SSDs. The architecture of the internal logic in a flash device is almost like volatile SDRAM or SRAM. However, due to differences in the storage cells used, the flash media has the capability to retain the stored information without power.

As shown in Fig. 1, flash memory stores information in an array of memory cells made from floating-gate transistors [2]. In general, the flash cell stores only a single bit of data, and is therefore called a single-level cell.

The wiring and interconnection scheme between the individual single-level cells determines whether it is a NAND flash or NOR flash.

In all flash memories, the single-level cell in its default state is logically equivalent to a binary “1” value. By programming the floating-gate transistors its status can be changed to “0”.

An erase operation is required to bring back the flash cell to its default state with value “1”. In other words, if a cell must be written with a new value, it should be first erased in preparation for the write (since the value of the new bit is unknown), followed by programming the new value. The need for erasure with every write operation is one of the factors resulting in longer write access time.

A single flash package comprises billions of flash cells that are organized in a hierarchical architecture, as depicted in Fig.2. The basic unit is a flash page. Multiple flash pages compose a block, which further forms a plane. In NAND flash memories, reading and writing are performed in a granularity of flash page. However, erasure is carried out in a granularity of flash block. The typical sizes of a page and block, as well as the specifications for basic read/write operations, are summarized in Table I [21].

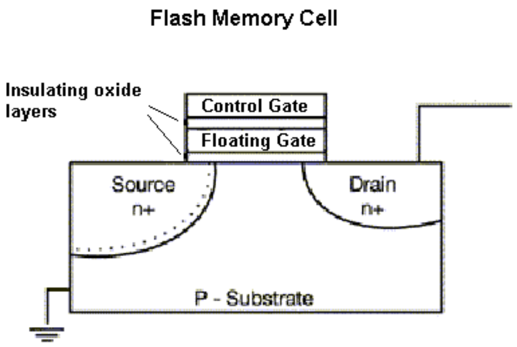

Fig. 1. Flash memory information storage mechanism

TABLE I
Typical Sizes of a Page and Block

Page Read to Register	25μs
Page Program (Write) from Register	200μs
Block Erase	1.5ms
Serial Access to Register (Data bus)	100μs
Die Size	2 GB
Block Size	256 KB
Page Size	4 KB
Data Register	4 KB
Planes per die	4
Dies per package (2GB/4GB/8GB)	1,2 or 4
Program/Erase Cycles	100 K

The internal architecture of an SSD device is illustrated in Fig. 3. It consists of three basic components.

Host Interface Logic: This component handles the communication with the OS and emulates an HDD interface as well.

Control Logic: The basic function of the SSD controller is to convert logical block address to logical flash page address and further to physical page address, i.e., the functionality of Flash Translation Layer (FTL) [12]. This component is responsible for additional advanced features, such as interleaving, wear leveling, etc.

An Array of Nonvolatile Flash Packages: Multiple flash packages are combined to give the large storage size typical of SSDs. The array is organized appropriately to achieve the required performance through interleaving.

Constraints on the amount of data that can be written to an SSD stem from the properties of NAND flash. Specifically, a block must be erased before being re-written, and only a finite number of erasures are possible before the bit error rate of the device becomes unacceptably high [7]. SLC (single-level cell) flash typically supports 100K erasures per flash block. However, as SSD technology moves towards MLC (multi-level cell) flash that provides higher bit densities at lower cost, the erasure limit per block drops as low as 5,000 to 10,000 cycles.

According to Micron’s data sheet [10], under a specific workload, its 60 GB SSD only has write lifetime of

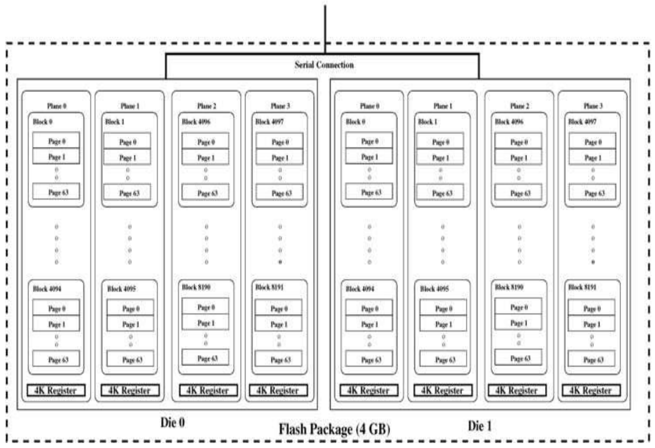

Fig. 2. Hierarchical Architecture

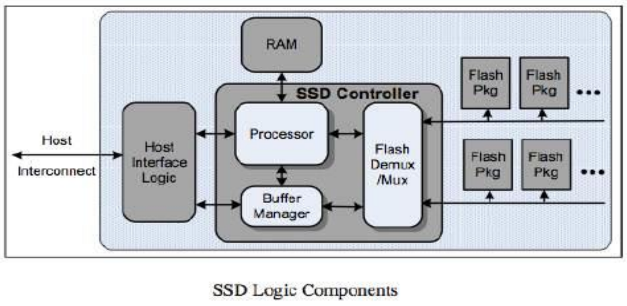

Fig. 4. SSD Logic Components

42 TB, which is a reduction in write-lifetime by a factor of 7. It is conceivable that under a more stressful workload, SSD write-lifetime decreases by more than an order of magnitude.

III. RELATED WORK

Modern day SSDs have mainly two drawbacks: 1) poor random write performance because of *erase-before-write* property, and 2) low lifetime because of limited number of erase per block property. For the last decade researchers have proposed many solutions to negate these two drawbacks of SSDs. Proposed solutions are can be categorized into two divisions 1) by implementing start-of-the-art flash translation layer, or 2) by implementing state-of-the-art write buffer management system.

FTL (shown in Fig. 4) is nothing but a software layer residing on top of the physical medium of SSD. It maintains a mapping between logical pages and physical pages. By means of intelligent wear-leveling and garbage collection mechanisms, FTL can evenly distribute erasures to flash block. Thus, it improves performance and increases the lifetime of SSD.

There have been many FTL proposals like Log-structured FTL architecture [6], then page-mapping FTL [14], block-mapping FTL [15] , hybrid mapping FTL and most recently Demand –based page level FTL (DFTL) [10].

Write buffer management system is the alternative approach of the FTL approach. In this approach buffer memory is added above FTL to serve as write buffer. As a write buffer, frequently updated data blocks can be kept on the buffer for as long as possible before being de-staged to the flash physical medium. Thus, write buffer reduces number of erases and offers a better performance [16].

There have been many proposals. Clean first LRU (CFLRU) [17] is buffer cache management algorithm. According to this algorithm it attempts to choose a clean page as a victim rather than dirty page because writing cost is much more expensive.

Block padding LRU (BPLRU) [19] is also a buffer management scheme to be applied to the write buffer inside SSDs. BPLRU manages buffer only for write requests. BPLRU implements LRU algorithm in the block level instead of page level. It uses CFLRU as the cache replacement algorithm.

There have been proposals of efficient write cache [8] to provide balanced read/write performance for flash memory. Also, it uses an efficient updating technique to provide more responsive SSD architecture by writing back stable but dirty flash pages.

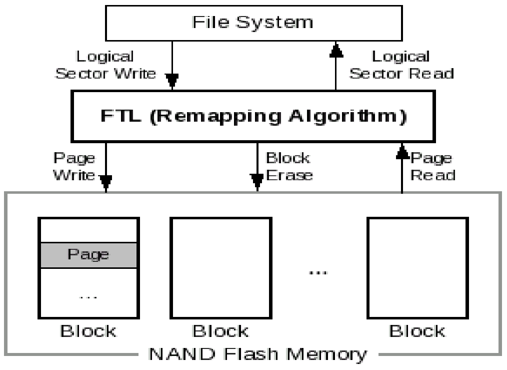

Fig. 3. NAND Flash Memory

Predicted average update distance LRU (PUD-LRU) [16] is also a write buffer management system where predicted average update distance (PUD) is used as the key block replacement criterion on top of log-block FTL scheme. To take advantage of the log block FTL, PUD-LRU maximizes number of valid pages in de-staged block in each erase operation.

IV. MOTIVATION

Well-designed FTL cannot replace buffer management. Also, these write-caches reduce write operation on SSDs [16]. This in turn reduces number of erase operations on the SSD. These disk-based write caches improve SSD lifetime significantly without sacrificing performance.

Other than BPLRU [19] none of the works previously used multilevel cache in their design. But BPLRU used two caches in two sides of the design. They used one cache to in the File system side to manage the I/O and the other cache in the SSD side to make it work as write buffer.

RAM can be used as a fast and effective write cache; however, the overriding problem with RAM is that it is not persistent. Increasing the RAM size or the timer interval for periodic flushes may reduce the number of writes to storage but only at the cost of a larger window of vulnerability during which a power failure or crash could result in lost updates.

The large performance gap between reads and writes is due to two reasons described. Firstly, a basic write operation (excluding erasure) is by itself slower than reading. From Table I, it is seen that a write takes 200 μs to program a flash page; the time to read a flash page is merely 25 μs. Secondly a flash cell must be erased before it can be re-programmed.

Since the granularity of writes is a flash page (4K bytes) but the granularity of an erase operation is a flash block (64 pages), a page-write operation can incur a large penalty if the block being written has valid data (thereby requiring erasure). When a page is being written to the flash, because of the erasure the entire block containing the page can lose data; the FTL in the SSD controller ensures data integrity by performing data copy before erasure [11]. The FTL also attempts to minimize block erasures by maintaining appropriate mappings between logical in the flash memory.

The basic policy used is write-back, i.e., data are always first written into the cache, and only later propagated to flash memory. In our design, we have tried to merge the write back property of caching with LRU cache replacement algorithm.

A cache miss refers to a failed attempt to read or write a piece of data in the cache, which results in a main memory access with much longer latency. The graph shown in Fig. 5 summarizes the cache performance seen on the Integer portion of the SPEC CPU2000 benchmarks, as collected by Hill and Cantin [4].

We have tried to implement two levels of write-cache instead of single level write-cache of greater size cause of latency issue. Increase of size may increase the hit rates, but it also will increase the cache latency.

V. MULTILEVEL ARCHITECTURE

SSD has an inherent problem with the writing of data. Data reads are at the granularity of a page, writing is slower because media blocks must be erased before they can be reused for new data. An erase operation is considerably more expensive than a read or writes operation by itself. In addition, each block can be erased only a finite number of times before becoming unusable, therefore, the life of an SSD is limited to a finite number of writes. Previous attempts as in [5] have targeted the problem of slow writes in SSD by making sequential writes rather than random writes and perform random reads. Random reads have proved to be faster for SSD; therefore, caching page read does not pay much significance to the overall performance of SSD read.

Our approach is based on the previous findings, stating that reads are faster as in [5] and writes shall be targeted for improvement. As proposed in [8] introduction of a cache in SSD for caching all write operations before actually writing them to SSD can have significant improvement in write performance. This way writes can be made to the SSD from the cache only when the SSD is idle, furthermore, it would be useful to explore the write sequencing as done in [5] to leverage the sequential write capability rather than a random write.

All write operations on a page are cached; all read operations are directory sent to the SSD; however, a read hit can take place if the data is already loaded in cache by a previous write operation on the same block. L1 write hits takes place if the page being requested is already located in the cache, in case of a L1 cache miss the request is forwarded to L2 cache, which generally is larger than L1 cache. An L2 cache hit occurs if the page being requested is present in L2 cache. In case of a miss at the L2 the page is first loaded from the SSD into L2 cache and then copied onto the L1 cache as well. Ideally the L1 cache is frequently accessed; L1 cache must be updated with the frequently accessed blocks to augment spatial locality and temporal locality.

Our implementation is software based, we would like to incorporate optimized cache management algorithm in our approach. To overcome the erase overhead in case of write operation, we would like to write dirty data from cache only when the SSD is idle. Likewise, care would be taken to write dirty data this is not accessed frequently; perhaps cache data is only written when it is not accessed for any longer. This way we can improve the locality of pages that are accessed frequently and eventually written when they are no longer required, thus we can improve the life of SSD by minimizing the number of writes. After the write to SSD is completed, we may or may keep a copy of the page in cache, if the write was required as in case of cache capacity exceeded, the pages that

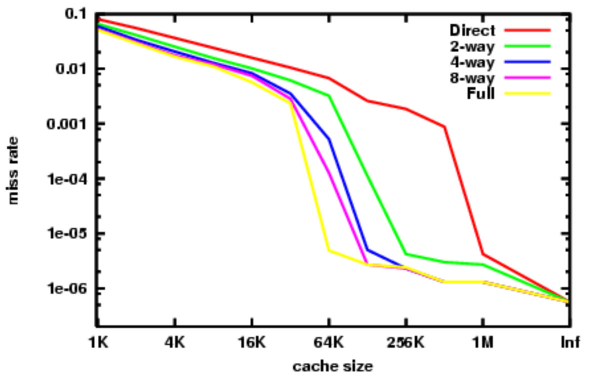

Fig. 5. Cache Size

are least frequently accessed would be removed and written to SSD if they were marked dirty.

We would like to explore and use the following cache replacement algorithms, both algorithms provide a set of benefits over the other available techniques. Efficient use of cache would help in achieving more performance, the performance measures are almost always constraint on the performance of cache management techniques being used.

A. The Least Recently Used (LRU)

LRU maintains an ordered list (the cache directory) of resource entries in the cache, with the sort order based on the time of most recent access. New entries are added at the top of the list, after the bottom entry has been evicted. Cache hits move to the top, pushing all other entries down.

Since, L1 is most frequently accessed and updated we would expect better performance in this case.

B. Second Level Buffer Cache

The page replacement algorithms decide which data pages to page out (swap out, write to SSD) when an allocation must be made. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. [13]

The LRU algorithm is used for L2 level caching, we expect better performance outcomes with this algorithm. Capacity of L2 cache is higher than that of L1 cache, therefore, L1 cache write miss should ideally result in L2 cache hit. Furthermore, efficient techniques can be used at the L2 level to minimize the number of direct access of SSD in the case of frequent writes.

VI. IMPLEMENTATION

We used an object-oriented programming language: Java. Our goal would be to simulate the L1 and L2 caching with the SSD. Our implementation would contain the centralized component: Cache Controller (CC), which manages page loads and writes on both the L1 and L2 caches. The cache controller would act as an interface between the processor and the L1 and L2 caches, so all read and write request would be managed by the cache controller. The cache controller would keep track of the SSD status and would invoke the cache management algorithms whenever either of the cache has reached a certain capacity threshold. Furthermore, the cache controller would interface with the SSD and invoke the Garbage Collection (GC) functionality of SSD after some regular intervals or after a certain number of writes to the SSD.

Ideally the cache controller would maintain meta information on both L1 and L2 cache, this information would contain hash table containing the mapping from tag-id on cache onto a physical address in the SSD i.e. logical to physical address mapping, additionally we have maintained logging information for certain blocks that are accessed frequently and a counter for the number of times data is written to the same block.

We tested our approach against the traditional SSD system with no caching. We also compared the performance of the LRU Algorithm for Second Level Buffer Caches. We expect greater improvement with this approach, and the write hits should help in reducing the response time. Additionally, managing the data writes to the SSD would improve the overall life and hence reliability of the SSD.

The following diagrams shown in Fig. 6 and Fig. 7 explains the flow data in cache.

VII. EVALUATION

We used a workload of 100 K write operation of a size of 70MB size of operation developed by Iozone. Iozone is filesystem benchmark tool. The benchmark generates and measures a variety of file operations. Iozone has been ported to many machines and runs under many operating systems. Iozone is useful for performing a broad filesystem analysis of a vendor's computer platform. The benchmark tests file I/O performance for the following operations: *Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, aio_write* [23].

The DiskSIM configurations that we used in this experiment are summarized in the Table II.

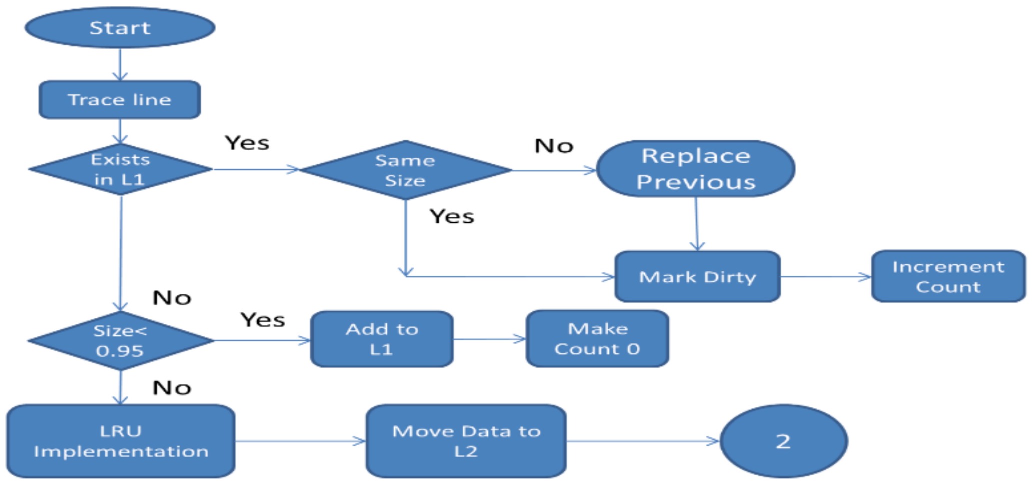

Fig. 6. L1 Cache

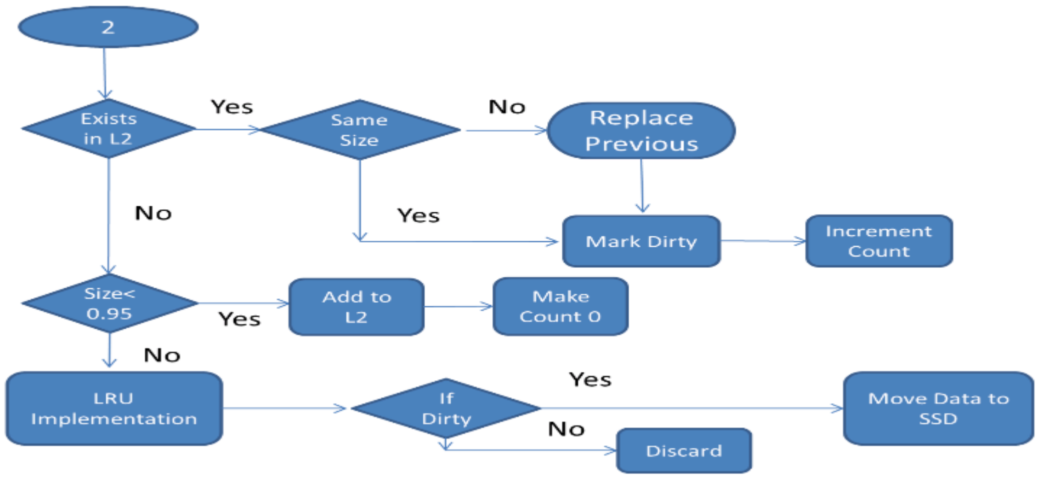

Fig. 7. L2 Cache

TABLE II
DiskSIM parameter for this experiment

Parameter	Value
Page Write	200 μs
Page Read	25 μs
Block Erase	1.5ms
Block Size	256 KB
Page Size	4 KB
Erase Cycles	10K

The trace file had approximately 80 % cache-able data which is ideally suitable for using in this scenario we used an L1 cache size of 8 KB and L2 cache size 16 KB and ran the trace file in 10 iterations of equal size of our implementation.

We assumed the same access time for both L1 and L2 cache. However, generally L1 is faster than L2, as future work our implementation can be extended by including access time for L1 which is less than access time of L2 for this simulation.

We present our results by evaluating our implementation against the standard SSD without caching. We compared a number of factors including the number of writes, response time, and the overhead of using two cache with the SSD.

From Figure 8, it is clear that number of writes were reduced by almost 63 %. Case 9 is unique where the data lacked spatial consistency and size of each write was comparatively greater than in previous iterations.

Results (see Figure 9) shows the response time comparison between our implantation and the traditions SSD. The average response time did not show much improvement in this case. Case 9 had unusual performance in this case as well.

From the results shown in Figure 10 it is clear that there was on average performance in the execution time in the Multi-level cache design. This execution time performance occurred as a result of the overall reduction the number of write request on SSD.

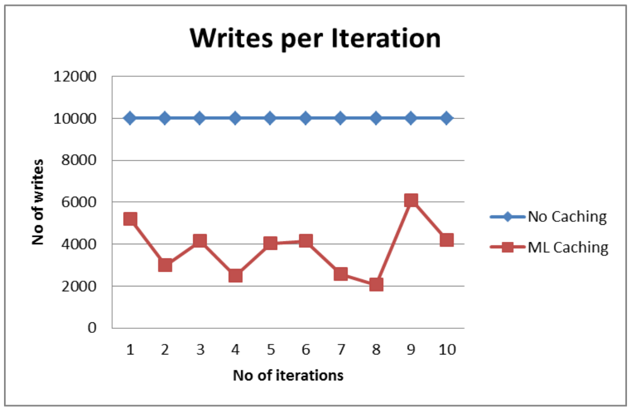

Fig. 8. Write per iteration

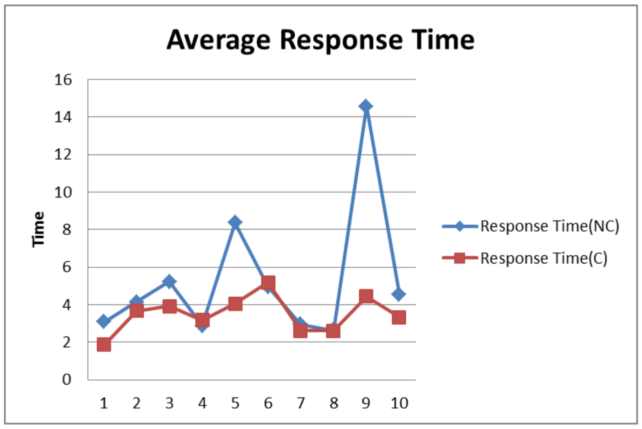

Fig. 9. Average Response Time

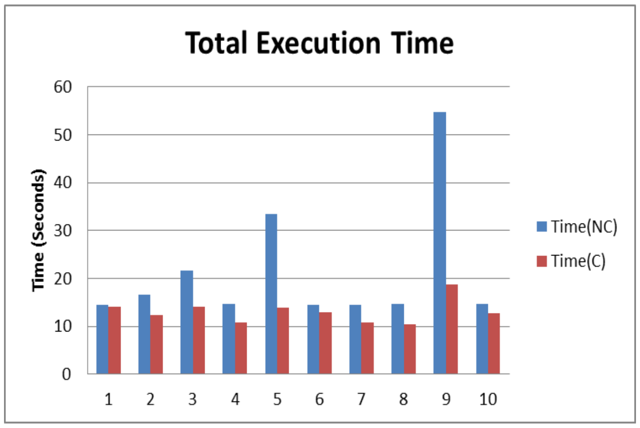

Fig. 10. Total Execution Time

The overhead of using Multi level cache with SSD is acceptable, as depicted in the Fig. 11.

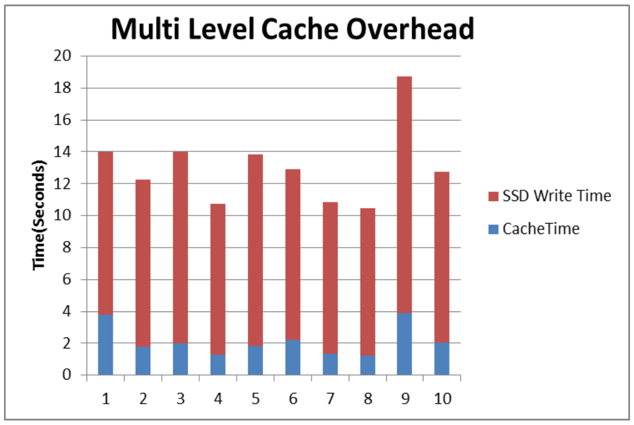

Fig. 11. Multi-level Cache Overhead

VIII. CONCLUSION

We recorded Significant (63% in our experimental setup) reduction in the number of writes. We also observed a reduced number of SSD which indirectly helps in reducing the number of erase operations in SSD and eventually helps in wear-leveling. We also conclude that execution time reduced with number of cache hits. Finally we can predict an anticipated improvement with Multi level caching.

There were some limitations in our study that can be addressed in future implementation. Specifically, we used a workload of all write operations; it would be interesting to see if the performance of SSD is affected in some way by including read operations as well.

Secondly, we used a single trace file used. We used LRU cache replacement algorithm in both L1 and L2 cache's. It would be interesting to use State-of-the art algorithm's (BPLRU, PUD-LRU etc.) in our multilevel cache architecture and compare performance with single level write buffer architecture.

As part of the future work we would like to compare performance with existing buffer management schemes.

IX. References

[1] Dumitru, D. (2007, Aug 16). *esayco-flashperformance-art.pdf*. Retrieved from http://managedflash.com/news/papers/easyco-flashperformance-art.pdf

[2] *[23]Ssd extension for disksim simulation environment*. (2009, March 6). Retrieved from http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/

[3]]Bucy, J, Schindler, J, Schlosser, S, & Ganger, G. (2010, October 28). *The disksim simulation environment (v4.0)*. Retrieved from http://www.pdl.cmu.edu/DiskSim/

[4] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach, 4th Edition. Morgan Kaufmann, Amsterdam, Boston, 2006

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD Performance. In ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference, pages 57{70, Berkeley, CA, USA, 2008.USENIX Association.

[6] Gal, E, & Toledo, S. (2005). Algorithms and data structures for flash memories. *ACM Computing Surveys (CSUR)*, *37*(2),

[7] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo. Endurance enhancement of Flash-memory Storage Systems: Ancient static wear leveling design. In DAC '07: Proceedings of the 44th annual Design Automation Conference, pages 212{217, New York, NY, USA, 2007. ACM.

[8] "Huang, M, Serres, O, Narayana, V, El-Ghazawi, T, & Newby, G. (2010). "Efficient cache design for solid-state drives". *Proceedings of the Proceedings of the 7th acm international conference on computing frontiers* Bertinoro, Italy".

[9] H. Dai, M. Neufeld, and R. Han. ELF: An Efficient Log-structured Flash File System for Micro Sensor Nodes. In SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pages 176{187, New York, NY, USA,2004. ACM.

[10] *realssd_c200_1_8.pdf*. (2007). Retrieved from http://download.micron.com/pdf/datasheets/realssd/realssd_c200_1_8.pdf

[11] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim. A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications. ACM Transactions on Embedded Computing Systems (TECS), 7(4):38:1–38:23

[12] Kim, J, Kim, J, Noh, S, Min, S, & Cho, Y. (2002). A space-efficient flash translation layer for compact flash systems . *Consumer Electronics, IEEE Transactions*, *48*(2), 366-375.

[13] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill. Bit error rate in NAND Flash memories. In IEEE International Reliability Physics Symposium (IRPS), pages 9–19, April 2008.

[14] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A flash-memory based file system. In *USENIX Winter*, pages 155-164, 1995.

[15] SSFDC Forum. SmartMedia Specification. http://www.ssfdc.or.jp

[16] Jian Hu, Hong Jiang, Lei Tian, Lei Xu, "PUD-LRU: An Erase-Efficient Write Buffer Management Algorithm for Flash Memory SSD," mascots, pp.69-78, 2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2010.

[17] Park, S, Jung, D, Kang, J, Kim, J, & Lee, J. (2006). CFLRU: a replacement algorithm for flash memory. *Proceedings of the Cases '06 proceedings of the 2006 international conference on compilers, architecture and synthesis for embedded systems* Scottsdale, AZ

[18] T. Kgil, D. Roberts, and T. Mudge. Improving NAND Flash Based Disk Caches. In ISCA '08: Proceedings of the 35th Annual International Symposium on Computer Architecture

[19] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In M. Baker and E. Riedel, editors, Conference on File and Storage Technologies: FAST, pages 239{252. USENIX, 2008

[20] M-Systems. Two Technologies Compared: NOR vs. NAND. White Paper, http://www.dataio.com/pdf/NAND/MSystems/MSystems_NOR_vs_NAND.pdf, 2003.

[21] Samsung Corporation. K9XXG08XXM Flash Memory Specification. http://www.samsung.com/global/system/business/semiconductor/product/2007/6/11/NANDFlash/SLC_LargeBlock/8Gbit/K9F8G08U0M/ds_k9f8g08x0m_rev10.pdf, 2007.

[22] Design Tradeoffs for SSD Performance", Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, Rina Panigrahy. Usenix Annual Technical Conference (USENIX '08), June 08, Boston, MA.

[23] IOzone Filesystem Benchmark http://www.iozone.org