International Journal of Digital Information and Wireless Communications (IJDIWC) 1(1): 175-194
The Society of Digital Information and Wireless Communications, 2011 (ISSN 2225-658X)

An Efficient Hierarchical Dual Cache System for NAND Flash Memories

Jalil Boukhobza, Pierre Olivier

Université Européenne de Bretagne, France
Université de Brest ; CNRS, UMR 3192 Lab-STICC,
20 avenue Le Gorgeu
29285 Brest cedex 3, France
{jalil.boukhobza, pierre.olivier}@univ-brest.fr

Abstract. NAND flash memories are the most important storage media in mobile computing and tend to be less confined to this area. Nevertheless, it is not mature enough to allow a widespread use. This is due to poor write operations' performance caused by its internal intricacies. The major constraint of such a technology is the reduced number of erases operations which limits its lifetime. To cope with this issue, state-of-the-art solutions try to level the wear out of the memory to increase its lifetime. These policies, integrated into the Flash Translation Layer (FTL), contribute in decreasing write operation performance. In this paper, we propose to improve the performance and reduce the number of erasures by absorbing them throughout a dual cache system which replaces traditional FTL wear leveling and garbage collection services. C-lash enhances the state-of-the-art FTL performance by more than an order of magnitude for some real and synthetic workloads.

Keywords: NAND Flash memory, cache, FTL, wear leveling, performance, storage system, I/O workload.

1 Introduction

NAND flash memories are more and more used as main storage systems. We can find them in mp3 players, smart phones, laptop computers, and a huge set of electronic appliances. NAND flash memory is based on semiconductor chips giving them some very interesting characteristics. They are small, lightweight, shock resistant, and very power efficient. For all these reasons, they are considered as a promising technology for large mass storage systems [9][10][11]. Flash memory storage is, however, very expensive; it is one order of magnitude (~5$/GB in early 2009 [10]) more expensive than hard disk storage (~0.3$/GB in 2008 [9]), however, the costs are falling drastically due to fabrication process mastering and exponential market growth [12]. This continuous price-per-byte falling encourages both large scale storage systems vendors and users to test flash memories for future evolutions. When doing so, enterprises are faced with a major problem related to flash poor

performance [2]: some disks are still outperforming flash memories for both sequential and random write intensive workloads. Our current research focuses on the enhancement of sequential and some random write operations workloads on NAND flash media using caching mechanisms.

The poor lifetime of flash memories is due to the limited number of erase operations one can perform on a given block (see background section). In addition to this limitation, flash memories present poor write performance because writing can be performed only in one direction (from 1 to 0), so a block must be erased before being modified. The FTL is a hardware/software layer implemented inside the flash-based device. One of its main functionalities is the mapping of logical addresses to low level physical addresses. Throughout this mapping is performed the wear leveling, which consists in spreading the write/modify operations over the flash memory surface to increase the block average lifetime. In order to modify data, the block must be erased and rewritten or completely copied to another location. The use of such a technique implies to take into account the validity of data as many versions can be present in different blocks. Garbage collection mechanisms are used to recover free space.

Wear leveling in FTL relies on an on-flash SRAM-based buffer where the logical-to-physical mapping tables are stored. The use of such a SRAM being expensive, FTL designers tried to minimize the size of such metadata. In fact, one have to find a trade-off between increasing the performance of the flash memory by allowing a page mapping algorithm, which consumes a big amount of SRAM but reduces the number of block erase operations, and reducing the SRAM usage by increasing the granularity of the mapping which increases the number of erase operations (see background section). State of the art solutions are located between those two extremes.

We propose, in this paper to replace the existing wear leveling techniques by a small, flexible dual cache that takes into account the structure and the constraints of flash memory. C-lash system is mainly used to delay and to order write operations.

The main research contributions of this study are: 1) a novel caching technique to replace the existing wear leveling (and garbage collection) solutions. 2) The implementation of a simulator used to achieve performance evaluation based on FlashSim [13] [2] that is built by extending DiskSim 3.0 [14]. 3) The validation with a large number of real enterprise workloads such as different OLTP (On-Line Transaction Processing) applications, and synthetic workloads to demonstrate the performance impact for different workload patterns.

The paper is organized as follows: a description of flash memories is given in the background section and some related works are discussed. In the third section, C-lash architecture and policies are depicted. The following section gives performance evaluation methodology while the fifth section discusses the simulation results, and finally we conclude and give some perspectives in the last section.

2 Background & Related Work

Flash memories are nonvolatile EEPROMs. They are mainly of two types: 1) NOR flash and 2) NAND flash. NOR flash memories support bytes random access and have a lower density and a higher cost as compared to NAND flash memories. NOR

memories are more suitable for storing code [15]. NAND flash memories are, by contrast, block addressed, but offer a higher bit density and a lower cost and provide good performance for large operations. Those properties make them more suitable for storing data [15]. This study only considers NAND memories.

Flash memory is structured as follows: it is composed of one or more chips; each chip is divided into multiple planes. A plane is composed of a fixed number of blocks; each of them encloses a fixed number of pages (multiple of 32). Actual versions have between 128 and 1024 KB blocks of pages of 2-4 KB. A page actually consists of user data space and a small metadata area [2][4].

Three key operations are possible on flash memories: read, write and erase. Read and write operations are performed on pages, whilst erase operations are performed on blocks. NAND flash does not support in-place data modification. In order to modify a page, it must be erased and rewritten in the same location or completely copied to another page to avoid additional latency, and its corresponding logical to physical translation map entry is updated.

One fundamental constraint on flash memories is the limited number of write/erase cycles (from 10^4 to 10^6) [2]. Once this number is reached, a given memory cell can no more retain data. Some spare cells exist on the chip to cope with such an issue. Due to data locality (temporal and/or spatial), causing the write operations to be performed on the same blocks, some of those memory blocks tend to wear out more quickly. This very central problem pushed the researchers to design wear leveling techniques, implemented through the FTL, to even out the wearing over the whole memory area even though these techniques reduce dramatically the performance.

2.1 Basic FTL Schemes

FTL systems can be classified into 3 main classes, depending on the way logical to physical mapping is performed: page, block, and hybrid mappings.

Page-mapping. It is a flexible system in which each logical page is independently mapped on a physical one. When a write request addressing a page (already containing data) is issued, the physical page is invalidated, and the logical page is mapped to another free one, avoiding a block erase operation. Page mapping shows high performance in terms of response time and number of erase operations. The main drawback of the page-mapping scheme is the big size of the mapping table [1].

Block-mapping. It considers block rather than page granularity for mapping. The mapping table contains the translation between logical and physical blocks while the offset inside one block remains invariable. As compared with the page-mapping, the size of the mapping table is drastically decreased. With block mapping, problems occur when a write request addresses a physical page which already contains data. In this case, the system must copy the entire logical block to another physical one. These operations are extremely costly.

Hybrid-mapping. It mixes both preceding techniques. Flash memory is divided into two parts [6]: data and log blocks. The first are mapped by block while the second are mapped by page granularity. As the page-mapping method requires a large space for metadata, the number of log blocks is limited. They maintain most frequently

accessed data. When the number of free log blocks becomes too small, part of them are merged to be moved to data blocks. This operation is extremely costly.

2.2 Advanced FTL Schemes

We compare the cache system proposed in this paper with some efficient state of the art FTL systems named Fully Associative Sector Translation (FAST) and Demand-based Flash Translation Layer (DFTL):

FAST [6] is a hybrid-based FTL. It uses a block-based mapping table, and the log/data blocks partitioning previously described. FAST separates log blocks (0.07 % of the total space) into two spaces: a sequential and a random write one. DFTL [2] is a pure page mapping FTL. It resolves the mapping table size issue by placing part of it in the media itself. To improve response times, the most accessed pages are stored in the SRAM; it takes into consideration temporal locality. DFTL divides the flash space into 2 parts: the translation space (0.2% of the total space) which contains the flash metadata, and the data space which contains the user data.

Many other FTL systems have been developed, each with its pros and cons: a convertible (page/block) flash translation layer (CFTL) [17], a state transition fast based FTL (STAFF) [16] Mitsubishi [20], SSL [21], NFTL [18], other log block schemes such as BAST [6], BFTL (a B-Tree FTL) [19], etc.

2.3 Buffering Systems for Flash Storage Media

Even though designed FTL techniques are more efficient, performance of write operations are still very poor. Buffering systems, using an amount of RAM upstream of the FTL, have been designed to cope with this issue. They reorganize non-sequential requests streams before sending them to the FTL. There are many examples of such systems in the literature: CFLRU [7], FAB [3], BPLRU [5], BPAC [8], CLC [4], etc.

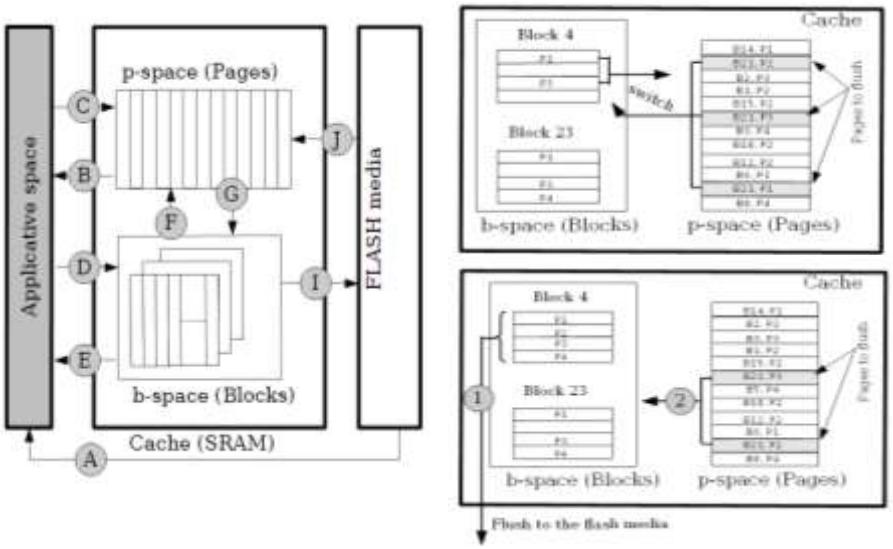
These systems are very different from the C-lash approach because they do not operate at the same level. They use large RAM buffers, sometimes larger than 100 MB, and are used in addition to the FTL. C-lash fits in a less than 1 MB RAM memory and is designed to replace some FTL services.

3 C-lash System Architecture and Design

3.1 C-lash Architecture

In C-lash system, the cache area is partitioned into two distinct spaces, a page space (p-space) and a block space (b-space). P-space consists of a set of pages that can come from different blocks in the flash memory while b-space is composed of blocks that can be directly mapped on those of the flash media as one can see in Fig. 1-a.

This dual cache system allows to represent, at the cache level, both granularities on which operations can be performed on the flash memory: reads and writes on pages and erases on blocks. The pages in the p-space and blocks of the b-space have,



respectively the same size as those of the underlying flash memory pages and blocks.

Fig.1. a) at the left hand side: structure of the C-lash cache for flash system. b) at the right hand side: examples describing different scenarios of the eviction policy for both spaces.

C-lash is also hierarchical, it has two levels of eviction policies: one which evicts pages from p-space to b-space (G in Fig. 1-a) and another level in which blocks from b-space are evicted into flash (I in Fig. 1-a). With this scheme, we insure that blocks are always flushed to the media rather than pages, which causes less erase operations.

P-space and b-space always contain either valid or free pages or blocks respectively. Consequently, when a read request arrives, the requested data are searched in both spaces. If we get a cache hit, data are read from the cache (B or E in Fig. 1-a), otherwise, they are read from the flash memory (A of Fig. 1-a). In any case, a read miss does generate a copy from the flash memory to the cache. When a write operation is issued, if the written data are present in the p-space or in the b-space, they are overwritten (respectively C or D in Fig. 1-a) with no impact on the flash media which avoids a costly erase and write operations (and merge if data has been modified). If data are not in the cache, they can only be written in the first cache level which is the p-space (C in Fig. 1-a). If enough pages are available, we use them to write the data. If not, we choose some pages to flush from the p-space to b-space (G in Fig. 1-a) and copy the new data in the freed space (see the next section for details).

3.2 Write Cache Eviction Policies

Two eviction policies are implemented in C-lash, one for each of the p-space and b-space.

P-space Eviction Policy. P-space contains written pages that come from different blocks in the flash memory. When a write request is issued and the considered page is neither present in the p-space nor in the b-space, a new page is allocated in the cache. If a free page is available, the new page is written and data in the corresponding location in the flash invalidated. If no space is available, the system chooses one or more pages to evict into the b-space (not into the flash media). The choice of the pages to evict is achieved in two steps. First, the system searches the p-space for the largest set of pages belonging to the same block. Then, we have two different cases: 1) A free block is available in the b-space and so the subset of pages found in the first step is copied into one free block. 2) No free block is available and then, the set of victim pages is compared to the number of valid pages contained in each block of the b-space area: if there is a block containing less valid pages than the previously found subset, a switch operation in the SRAM is performed (F and G in Fig. 1). This means that pages in the victim block are moved in the p-space while the subset of victim pages is moved in the freed block. This induces no impact on the flash memory. The second possibility arises when all the blocks in the b-space contain more valid pages than the subset of victim pages to evict from the p-space. In this case the b-space eviction policy is executed to flush a block into the flash media (See next section).

In the upper illustration of Fig. 1-b, we have an example of a p-space eviction; the chosen pages are those belonging to block B21 containing the biggest number of pages. One block in the b-space contains 2 valid pages which is less than the 3 pages to evict. Thus, C-lash switches between the block B4 and the 3 pages subset. The system, therefore, frees 1 page without flushing data into the flash.

B-space Eviction Policy. The b-space eviction algorithm is called whenever the system needs to write a subset of pages form the p-space to the b-space while all blocks are full. Therefore, C-lash writes an entire victim block from the b-space into the flash memory media in order to free one block in the cache. An LRU algorithm is used for this eviction. Throughout this algorithm, the system takes into consideration, in the b-space, the temporal locality exhibited by many workloads.

When a block eviction is performed, the whole corresponding block in the flash memory is erased before being replaced by the one in the cache. In case the flash media still contains some valid pages of the block in the cache to evict, a merge operation (J in Fig. 1-a) is achieved. This consists in reading the still valid pages in the flash memory and copying them into the cache before flushing (erase then write) the whole block. This read operation can be done either during a p-space eviction, we call it *early merge*, or just before flushing the block on the flash media and we call it *late merge*. Early merge is more advantageous if the workload is read intensive and shows temporal and/or spatial locality because we have more chances to make benefit from the cached data. If the workload is write-intensive, we gain no benefit in doing the early merge, we would prefer to postpone the merge operation using late merge. By doing so, we insure two main optimizations: 1) we read a minimum number of

pages since between the moment the pages are evicted into the b-space and the moment it is evicted into the flash, many pages can be written and so invalidated from the flash (no need to read them). 2) Since it is possible for a block in b-space to be moved to the p-space during a p-space eviction, it may not be worth doing the merge operation too early. This can cause extra flash read operations.

We restrict the scope of the presented study to the use of a late merge scheme.

An example of block eviction is shown in Fig. 1-b. In the lower illustration, we describe a p-space eviction leading to a b-space flush to the flash media. In the p-space eviction phase, 2 pages of the B21 block are chosen to be evicted. The blocks in the b-space contain more valid pages than the pages' amount to be evicted from the p-space. So, the system needs to evict a block into the flash beforehand. After that, the system copies both pages of the B21 into the freed block. In this specific example, no merge operation occurs because the flushed block is full of valid pages.

4 Performance Evaluation

We compare the performance of the C-lash system with the state-of-the-art FTLs because it has been built in order to replace part of those FTL services.

Fig.2. The simulated global storage system architecture.

4.1 Simulation Framework and Methodology

FlashSim is a simulator based on Disksim [14] which is the most popular disk drive storage system simulator in both academia and industry. Disksim is an event driven simulator. It simulates the whole storage system going from the device driver until the detailed disk movement. It also integrates a detailed synthetic I/O workload generator used in part of our experimentations. Disksim does not natively include flash memory support. FlashSim integrates modules specific to flash memories in Disksim. It is able to simulate basic flash device infrastructure: read, write and erase operations, in addition to logical to physical address translation mechanisms and garbage collection policies. FlashSim implements FAST, DFTL and an idealized page map FTLs.

We have increased the functionality of FlashSim to allow the simulation of a dual cache subsystem placed on the top of the flash media (see Fig. 2). The used cache is configurable and many cache policies can be simulated (FIFO, LRU, LFU).

4.2 Storage System and Performance Metrics

We rely on two main performance metrics: the average request response time and the number of performed erase operations. The response time is captured from the I/O driver point of view (see Fig. 2) including all the intermediate delays: caches, controllers, I/O queues, etc. We tried to minimize the intermediate elements impact to focus on the flash memory subsystem behavior. The second metric we capture is the number of performed erase operations. It indicates the global wear out of the memory.

We also discuss the write amplification phenomenon for each FTL/cache solution for all performed tests. Write amplification is the ratio of the number of writes, in terms of pages (of 2KB in our case), coming from the applicative layer and the number of writes really occurring on the flash media. The write amplification metric gives some clues on the efficiency of FTL algorithms.

In fact, the write amplification metric was first introduced by Intel in [25] as a new challenge resulting from the growing complexity of SSD wear levelers and performance. In our opinion, this metric alone is not sufficient to evaluate the efficiency of a given FTL or cache solution for a write intensive workload, but coupled with the number of erase operations, it can be more relevant.

4.3 Simulated I/O Workloads

We used different sets of real and synthetic workloads to study the impact of the C-lash system as compared to the chosen FTLs. We focused our research on write intensive applications for both sequential and random workloads, but we also show some results based on some read dominant workloads.

The simulated synthetic traces have been produced with Disksim I/O workload generator that is based on the sequentiality request rates, read/write operation rates, spatial locality, request sizes and address space variation. We explored the performance of the C-lash subsystems as compared to other FTLs. Synthetic workloads are in fact more flexible in exploring the limitations and strengths of a given solution (see Table 1). Request sizes follow a normal distribution with a mean of 2 and a variance of 2KB. This ensures that no sequentiality is caused by applied large request sizes, but only by the sequentiality and spatial locality rate parameters.

Table 1. Synthetic workloads tested with C-lash, FAST, DFTL and page map.

Seq. rate		Spatial locality		Write rate	
Default value	Other values	Default value	Other values	Default value	Other values
80%	50%, 65%, 90%	40%	5%, 10%, 30%,	80%	50%, 65%, 90%

Address space size		Request sizes	
Default value	Other values (GB)	Default value	Other values
1GB	0.1, 0.5, 2GB	nor (2, 2)	1, 4, 8, 16, 32KB

We performed more extensive simulations by varying two important parameters that are sequential and spatial locality rates from 20% to 80% by steps of 20 with a larger interval of request sizes (see Table 2) proving that C-lash is good performing for some random I/O workloads provided that request sizes are not extremely small. Request sizes follow a normal distribution with a mean of 1KB and a variance of 64KB and the inter-arrival times follow an exponential distribution from 0 to 200ms. The request size distribution allows to deal with a more representative workload containing a majority of small request sizes with a decreasing probability for bigger ones.

Table 2. Synthetic workloads with a focus on sequentiality and spatial locality.

Seq. rate	Spatial locality	Write rate	Address space	Inter arrival times	Request sizes
20%; 40%; 60%; 80%	20%; 40%; 60%; 80%	80%	2 GB	exp (0, 200ms)	nor (1, 64KB)

Table 3. OLTP workloads tested with C-lash, FAST, DFTL and page map.

Workloads	Write rate	Seq. write rate	Mean req. size (KB)
Financial 1	99%	83%	4
Financial 1	81%	81%	4
Financial 1	18%	13%	11.5
Financial 2	95%	30%	44
Financial 2	24%	1%	2.5
Financial 2	0%	-	7.5
Financial 1	71%	11%	2
Financial 1	94%	6%	8

For real workloads performance tests, we used different enterprise I/O traces from OLTP applications running in financial institutions [22] available thanks to the Storage Performance Council (SPC) [23]. Those traces are extracted from a 5 to 20 disks' storage subsystems. We isolated the I/O trace of each disk and applied each to our one-flash-SSD. In this paper, we did not take into consideration all the disks; we chose 3 disks from each, revealing different characteristics (see Table 3). In addition to that, for the two last disks (in Table 3.) we studied the impact of increasing the C-lash size by growing either the b-space or the p-space to analyze its behavior and compare it to state-of-the-art FTLs.

4.4 Performed Tests

We simulated a NAND flash memory with a page size of 2KB and a block size of 128KB (+4KB for metadata). The request addressing size interval varied between 100MB and 2GB. The three operations have the following values: a page read: 130.9µs, a page write: 405.9µs and a block erase: 2ms. This configuration is based on a Micron NAND flash memory [24]; its values are representative of the read/write/erase performance ratio.

In the performed set of tests, we compare one C-lash configuration with different FTLs: FAST, DFTL (a very efficient FTL) and the idealized page map. As seen earlier, page map FTL consumes a huge amount of memory but gives ideal performances as compared to other FTLs, as in [2], we use it as a baseline. The chosen C-lash configuration has 2 blocks (128KB each) and 128 pages which constitutes a total small size of 512KB. The hybrid (block and page) mapping table used with C-lash is very small as compared to the cache size; for a 512KB cache, its size is about 320 bytes. Simulations are performed with synthetic and real workloads (see Table 1, 2 and 3). All the performed simulations begun with the same initial state, the flash media was supposed to be completely dirty (or at least the addressed area). Each new flash write in a given block generated an erase operation.

5 Results and Discussion

Synthetic I/O Workload Comparison. Fig. 3 shows the behavior of both mean response time and number of erase operations when varying the write rate and sequential rate of the I/O workload. We can see, from the left hand side illustrations, that the more we increase the write rate, the more the C-lash system outperforms the other FTLs and the more we approach from the baseline page map performance. For an 80% write rate, we enhance DFTL mean response time by more than 43% and reduce the number of erase operations by more than 47%. For this experimentation, we always outperform both FAST and DFTL and closely approach the page map performance. Another observation one can draw is that FAST FTL gives very low performance for the tested sequential write intensive workloads.

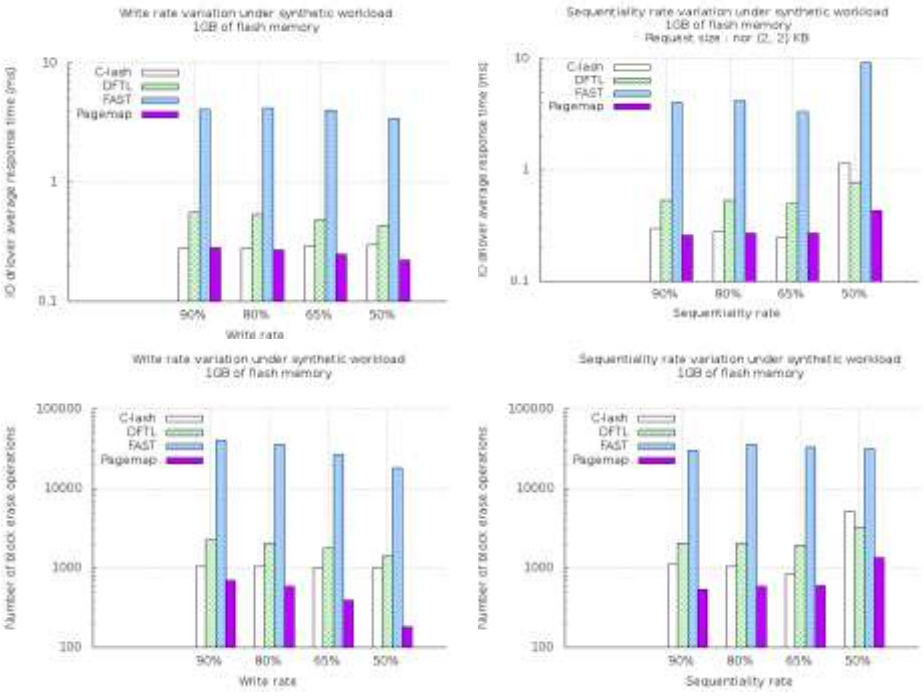
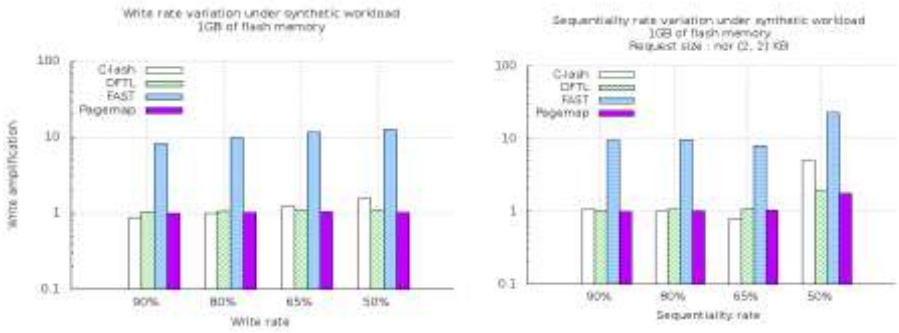


Fig.3. I/O driver mean response time (in milliseconds) and number of erases for the synthetic workload configuration of Table 1: variation of the write and the sequential rate.

The second tuned workload parameter is the sequential rate as one can see in the right hand side graphics in Fig. 3. For the mean response time, C-lash is outperforming both FAST and DFTL for sequential rates greater than 65% (up to 48% improvement in response time and 54% in number of erase operations) but performs poorly when sequential rates are less than 60%. In these simulations, request sizes were very small (nor (2KB, 32KB)) which negatively impacts C-lash performance



and spatial locality was fixed to 40%. We show, latter, in more details, the implications of sequential rate and spatial locality. We can also observe that for a sequential rate of 65%, C-lash gives a better mean response time while generating more erase operations than DFTL. The decreasing sequential rate generates more flush operations on the flash memory, especially for very small request sizes, which increases the number of erase operations, but we still benefit from the low response times because writes are reported on the cache.

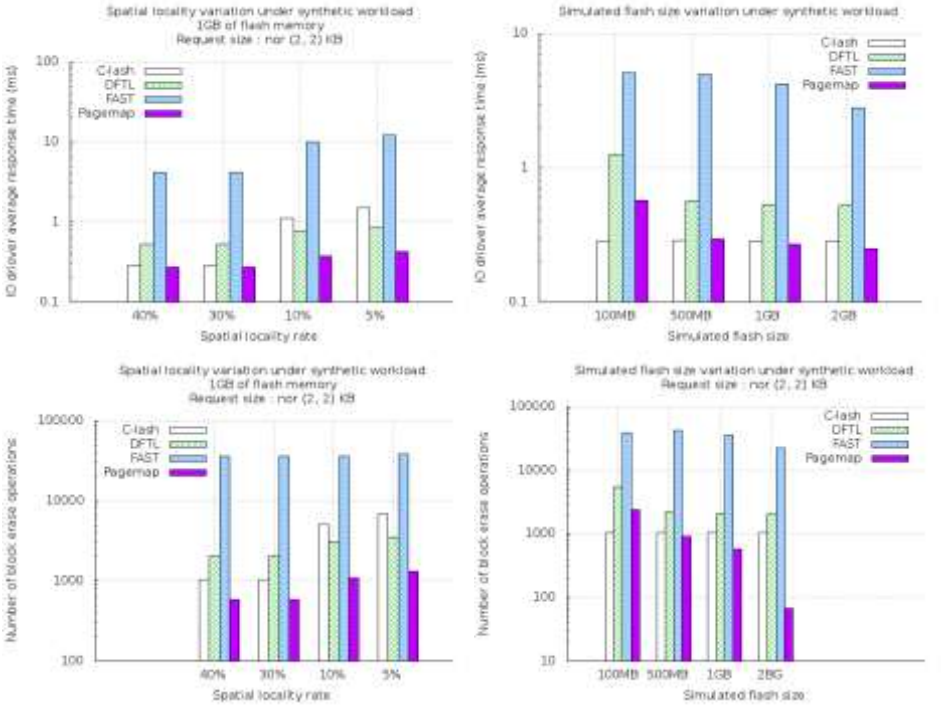
Fig.4. Write amplification phenomenon for synthetic workload configuration of Table 1: variation of write and sequential rate

Fig. 4 shows the write amplification (WA) for both sequential rate and write rate variation. As specified earlier, the WA metric has been calculated as the ratio between the number of pages written by the applicative layer and the number of pages written effectively on the flash media. We can observe from Fig. 4 that the more we increase the sequential rate or/and the write, the better is the WA of C-lash. Better WA means a smaller value. We also can see from Fig. 4 that C-lash is the only mechanism delivering a write amplification lower than 1 in some cases. This is because C-lash absorbs some write operations and does perform neither garbage collection nor wear leveling which increases the WA metric. However, merge operations have a real negative impact on the C-lash WA; if we only modify one page in block from the B-space to be flushed, the whole block's pages will be flushed (after being read from the flash media).

Fig. 5 shows the variation of both performance metrics when changing the spatial locality and the flash size. We can observe that for spatial localities greater than 30%,

C-lash outperforms the other FTLs (more than 40% enhancement) while it gives poor performances for 10% rate. Spatial locality is different from sequentiality from the simulator point of view. Sequentiality means strictly contiguous request addresses while spatial locality means requests with neighbor addresses.

Graphics on the right hand side of Fig. 5 depict the variation of performance according to the simulated size of request address space. We can notice that the mean response time of the C-lash system is always better than other FTLs' with a ratio

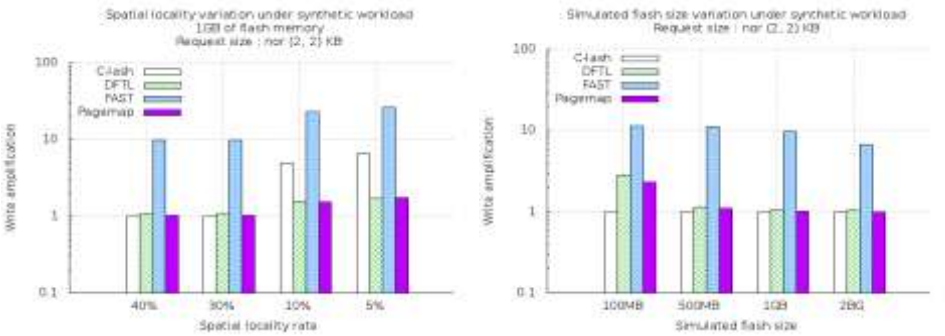


going from 35% to 63%. C-lash always generates the same number of erase operations independently from the flash size while some other FTLs are less stable. For instance, DFTL outperforms all the other FTLs by generating no erase operations for a flash size of 500MB, we could not explain such a result. More tests were performed with more extensive sizes (10, 20GB, etc.), C-lash showed better performances for all tested sizes. We noticed that for large address spaces, FAST performed better, but it still lagging behind DFTL and C-lash.

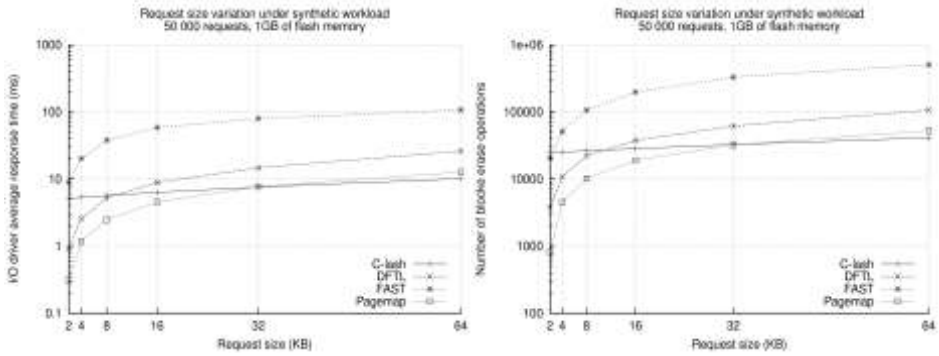
Fig.5. I/O driver mean response times (in milliseconds) and number of erase operations for the synthetic workload configuration of Table 1: variation of spatial locality and flash size.

Fig.6. Write amplification phenomenon for synthetic workload configuration of Table 1: variation of spatial locality and flash size.

In Fig. 6, we can observe the WA when varying spatial locality and flash size parameters. We can notice that the bigger is the spatial locality, the better is the WA



for C-lash. For small spatial locality values, WA of C-lash is big; as seen earlier, this is due to more generated merge operations. For the flash size variation, C-lash WA does not vary according to the flash size while other FTLs do. In fact, DFTL, FAST and page map FTLs use the flash free space as a provisioning space to post pone erase operations by simply copying modified data into the available free space without



performing additional erase operations.

Fig.7. I/O driver mean response times (in milliseconds) and number of erase operations for the synthetic workload configuration of Table 1: variation of request sizes.

In Fig. 7, we depict the response time and number of blocks erasures when varying the request sizes with fixed sequentiality and spatial locality values to 20%. We have intentionally chosen small values for those parameters in order to better observe the request size impact. In fact, the request size is just another aspect of sequentiality rate, the bigger the request size, the greater the sequentiality as seen by the flash device. We can observe that C-lash performs better than DFTL and FAST for request sizes exceeding 8KB (4 pages) and even performs better than page map for more than 32KB (16 pages). The same observation can be drawn for the number of erase operations. In fact, the larger the request sizes, the bigger the sequentiality as seen by the flash device, and the better C-lash is performing.

Fig. 8 shows the WA when varying the request size. We can notice that C-lash performs a better WA than DFTL for request sizes larger than 16KB and better than page map for request sizes larger than 32KB. We can notice that C-lash performs better than DFTL in terms of response time and number of erase operations earlier than it does for WA. This is because C-lash generates less erase operations by writing more pages (merge operations) on a given flash block. Instead of that DFTL use the free space to spread pages on the whole surface without merging the blocks,

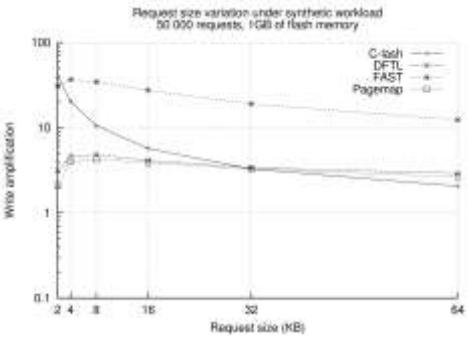


Fig. 8 Write amplification according to request size.

though generating more erase operations but less writes.

We conclude that for synthetic workloads described in Table 1, the C-lash system gives better performance than the tested FTLs for workloads showing a sequential rate of more than 60%, and at least 20% spatial locality for small request sizes, or for request sizes larger than 8KB. We can have an impressive performance increase of up to 65% on response times and number of erase operations as compared to the 2nd best performing FTL (DFTL).

Table 4. shows a more thorough study of the sequential rate and spatial locality variation (see Table 2 for configurations). We observe that for those configurations, whether the workloads are random or sequential, C-lash always outperforms the other FTLs. It is better than DFTL by at least 59%, while it outperforms page mapping FTL by at least 13%. Table 4. shows a summary of obtained results. Results in bold represent cases for which C-lash outperforms the page map idealized FTL.

Table 4. Synthetic workload results' summary for spatial locality and sequential rate varitation.

Spatial locality (%)		**20**				**40**				**60**				**80**			
Sequentiality(%)		***20***	***40***	***60***	***80***	***20***	***40***	***60***	***80***	***20***	***40***	***60***	***80***	***20***	***40***	***60***	***80***
DFTL	*Resp. t. %*	59	60	61	63	62	64	67	63	65	70	67	63	69	70	67	63
	Erase %	57	59	62	67	62	66	72	67	68	78	72	67	84	78	72	67
FAST	*Resp. t. %*	91	91	90	87	91	91	91	87	91	91	91	87	90	91	91	87
	Erase %	91	91	92	91	92	93	95	91	94	96	95	91	97	96	95	91
P. map	*Resp. t. %*	**13**	**17**	**27**	**23**	**25**	**33**	**43**	**23**	**41**	**55**	**43**	**23**	**62**	**55**	**43**	**22**
	Erase %	**11**	**18**	**31**	**28**	**26**	**38**	**52**	**28**	**47**	**68**	**52**	**28**	**79**	**68**	**52**	**28**

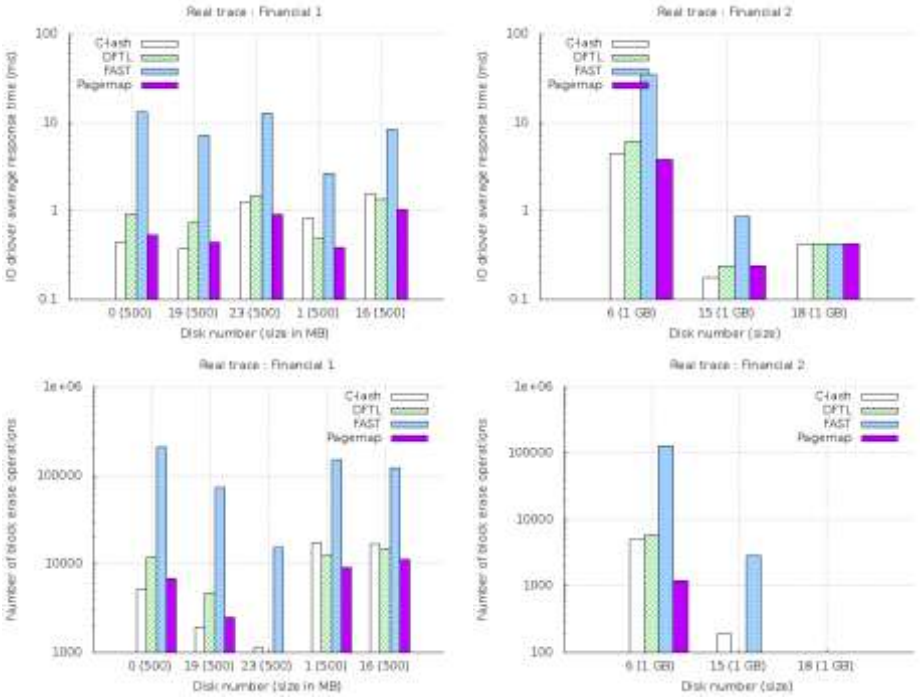
Results of Table 4. do not contradict those of Fig. 5. In fact, C-lash does not perform well for very random workloads coupled with very small request sizes (< 4KB), but still the performance does not fall dramatically. We are working to improve the C-lash performance for those kinds of workloads.

Real OLTP Performance Evaluation. We depict, in this part, the results of comparison with real enterprise workloads described in Table 3.

Fig. 9 shows that for all the tested disks (except disks 1 and 16 that will be discussed latter), C-lash performance is better than the other FTLs. It even achieves better results than those of the page mapping FTL for the two first disks. From the mean response time point of view, C-lash enhances the FAST FTL performance by more than 95%, while improving the DFTL by 53%, 51% and 13% respectively for the three simulated disks. For the erase operation metric, C-lash generates less erase operations than all the other FTLs for the first disk, while it reduces the DFTL number of erasures by 57% for the second disk. For the third one, despite the fact that C-lash gives better response times, it produces more erase operations. These results confirm that C-lash outperforms the tested FTLs for highly sequential workloads (disk 0: 83% and disk 19: 81%).

The performance increase of C-lash on the Financial 2 workload, as shown in Fig. 9, is less impressive, even if it surpasses all the tested FTLs for the two last disks. We observed an improvement of less than 10% on response times as compared to DFTL.

The main reason behind this small growth is the write operation sequential rate,



spatial locality and request sizes (except for the first disk) that are small in this case.

Fig.9 I/O driver mean response time (in milliseconds) and number of erase operations for the OLTP real workload Financial 1 and Financial 2 configurations of Table 3.

Fig. 10 shows the WA corresponding to real traces of Financial 1 and 2. As seen earlier, C-lash is the only system for which the WA is smaller than 1 in some cases. Except disks 23, 1 and 16 of Financial 1 workload (the two last ones will be discussed later), C-lash shows a better WA in all other cases.

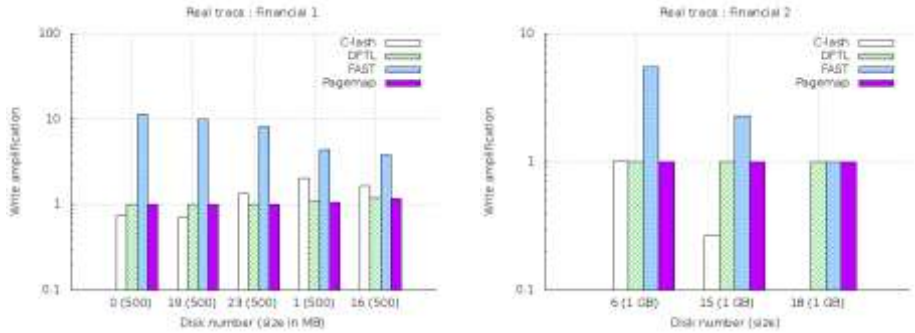


Fig.10 Write Amplification phenomenon for the OLTP real workload Financial 1 and Financial 2 configurations of Table 3.

Table 5. Real OLTP simulation results' summary.

		Financial 1			Financial 2		
	Disks	1	2	3	1	2	3
DFTL	Resp. times	53%	51%	13%	9%	4%	2%
	Erase ops.	57%	59%	-	*-12%*	-	0%
FAST	Resp. times	97%	95%	93%	86%	80%	2%
	Erase ops.	97%	97%	92%	95%	87%	0%
P.map	Resp. times	**18%**	**17%**	*-31%*	*-34%*	**4%**	**2%**
	Erase ops.	**26%**	**23%**	-	*-79%*	-	-

A summary of the results of the simulation is shown in Table 5. All gray cells are those for which C-lash shows better performances than DFTL and FAST. All the results in bold represent cases where C-lash outperforms the page map idealized FTL. Italic results are traces for which C-lash does not perform better for one or both performance metrics, most of those values appear for the page map idealized FTL. The cells containing a (-) sign mean that the FTLs does not show any erase operations while C-lash exhibits a small number. The 0% cells mean that neither C-lash nor the FTLs exhibit erase operations.

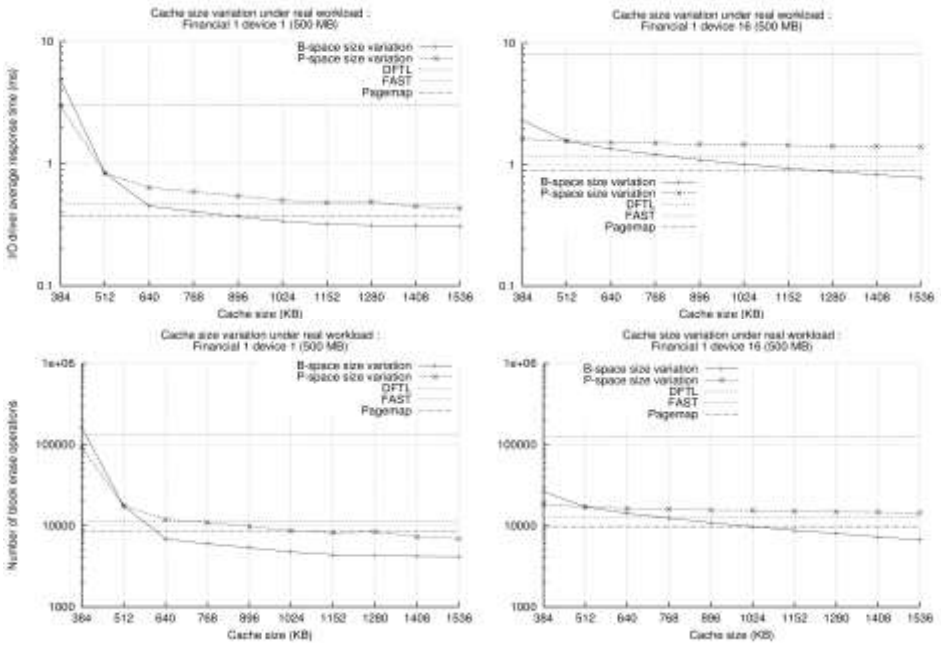
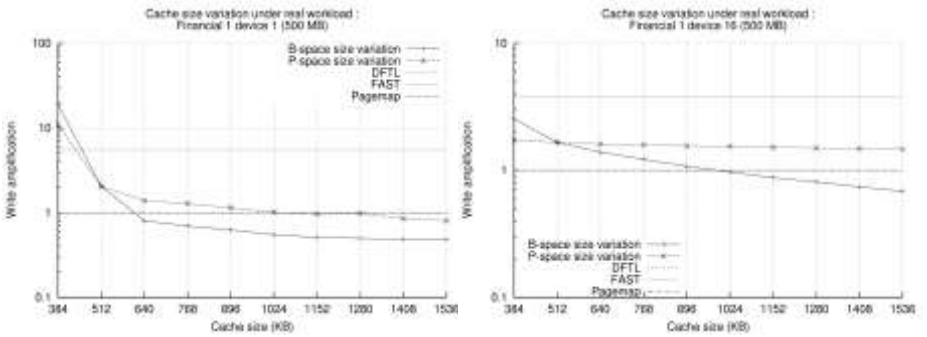


Fig.11 I/O driver mean response time (in milliseconds) and number of erase operations for disks 1 and 16 of the OLTP real workload Financial 1 with variation of the C-lash cache size.

In Fig. 9, we have shown results of simulation with a very small cache of 512KB and we observed that C-lash performed a little bit less than DFTL and page mapping FTLs. We varied the C-lash cache size in two ways: 1) by increasing the p-space (number of available pages) by steps of 64 pages (128KB), and 2) by increasing the b-space (number of blocks) by steps of 1 (128KB). When varying the p-space, the b-space was fixed to contain 2 blocks. The results are shown in Fig. 11 where they are compared to the three other FTLs. We can see that for disk 1, if we increase the cache size up to 640KB and 896KB by increasing the b-space, we respectively outperform DFTL and page mapping FTLs. Increasing the p-space by the same size does not result in the same performance increase. We can notice a similar behavior for disk 16, C-lash outperforms DFTL for 640KB cache size while it outperforms page mapping FTL for 1MB, and in both cases by increasing the b-space. Depending on the workload characteristics, it can be more relevant to increase the p-space or the b-space to gain maximum performance. In our case, we would prefer increasing the b-space, this should be due to the presence of interleaved sequential streams into the



trace file which requires getting extra blocks to separately store each sequential stream. Adding only pages in that case is not optimal because it does not guarantee that sequentially accessed data are preserved in the cache before being flushed as entire blocks.

Fig.12 Write Amplification phenomenon for disks 1 and 16 of the OLTP real workload Financial 1 with variation of the C-lash cache size.

Fig. 12 shows the WA variation when tuning the C-lash size, we can observe that the behavior is same as the one of response time and number of erase operations.

Other Performance and Cost Considerations:

- In the experimental evaluation part, the simulator does not consider the whole address translation and garbage collection latencies for the different FTLs. This parameter can negatively impact their performances as compared to the simple direct block mapping used in C-lash.

- The main disadvantage from reducing the number of erase operations by absorbing them through the cache is the data loss that can be induced by a sudden power failure. This issue is very important to consider. Different hardware solutions may be proposed. A small battery may delay the effective shutdown until the whole valid data are flushed [5], in the worst case and with the cache configuration tested in this paper (2 blocks with just one valid page and 128 pages from different blocks), this operation takes only 4 seconds. In C-lash system, we thought about this problem and consequently, we did implement the LRU algorithm on the b-space (small number of blocks) and not on the p-space which would have provoked more flush operations on the media. This ensures a more regular data flush operations, and so less data loss, at the expense of performance (increase of response time).
- From a cost point of view, the C-lash solution is very attractive; it does not use a lot of cache memory and simplifies the FTL implementation by removing some of its services. We can still reduce the C-lash size since the pages of the blocks in the b-space are not always entirely used. One solution consists in virtualizing the pages' addressing of the b-space region to be sure of getting good memory usage ratio. This is particularly true when dealing with weakly sequential workloads.
- For space considerations, we did not show the distributions of the erase operations over the flash area for each simulation. In addition to a decrease of the total erase operation, the performed tests showed a good distribution because the repetitive erase operations (on the same blocks) were absorbed by the cache. There are still some cases for which cache could not absorb such erases, but this rarely happens.

7 Conclusion and Future Work

We proposed, in this study, a paradigm shift as we introduced C-lash (Cache for Flash), a cache system for managing flash media supposed to replace the wear leveling and garbage collection parts of FTLs. We showed that C-lash system drastically improves the performance for a large spectrum of real enterprise and synthetic workloads in terms of response times and number of erase operations.

Beyond the C-lash solution, we think that it is more and more interesting to begin considering the migration from FTL wear leveling and garbage collection solutions towards cache based solutions (without the preceding services) for a large set of applications (there are still exceptions). The growing reliability and efficiency of flash memories reinforces us in this position.

C-lash is flash-aware because it takes into account the cost of the different flash memory operations (read, write and erase) in terms of access time and lifetime by flushing onto the media only blocks containing the maximum number of valid pages. This tends to severely reduce the number of erase operations and so improves the performances. C-lash takes also into account the temporal and spatial locality of data via its b-space LRU eviction policy.

C-lash performance was tested on a large set of workloads either synthetically generated or extracted from well established enterprise trace repositories. The

achieved experimentations proved the relevance and efficiency of C-lash for a large set of workloads. In fact, we compared our system with an efficient FTL scheme named DFTL, and we enhanced the performance on the tested workloads by more than 65% in many cases. We also frequently performed better than the idealized page mapping strategy which consumes more than an order of magnitude additional memory.

C-lash system performs poorly on very random workloads with very small request sizes if the cache size is not big enough. This is mainly caused by the bad use of the b-space in that case. We will carry on tests on random workloads to finely identify the small parameters' windows where C-lash does not perform well. We propose, then, to study the possibility to virtualize the access to b-space pages in order to better exploit the whole cache size when facing random workload. Another solution to investigate consists in dynamically reconfigure/adapt the dual cache sizes according to the applied workload. The latter consumes less metadata (mapping tables).

We expect to extend our simulation framework to multi-SSD systems to explore in more details the interactions of the different disks in larger enterprise storage systems.

A patch to the Disksim simulator including the C-lash support will be available online with the bunch of performed tests.

References

1. Chung, T., Park, D., Park, S., Lee, D., Lee, S., Song, H.: A Survey of Flash Translation Layer. J. Syst. Archit. 55, 332--343 (2009)
2. Gupta, A., Kim, Y., Urgaonkar, B. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In Proceeding of the 14th international Conference on Architectural Support For Programming Languages and Operating Systems, pp. 229--240. New York (2009)
3. Jo H., Kang J., Park S., Kim J., Lee J.: FAB: Flash-Aware Buffer Management Policy for Portable Media Players. IEEE Trans. on Consumer Electronics, 52, pp. 485- 493 (2006)
4. Kang S., Park S., Jung H., Shim H., Cha J.: Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices. In IEEE Transactions on Computers, 58, no.6, pp. 744--758 (2009)
5. Kim, H. and Ahn, S.: BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In Proceedings of the 6th USENIX Conference on File and Storage Technologies. M. Baker and E. Riedel, Eds, CA, 1-14 (2008)
6. Lee, S., Park, D., Chung, T., Lee, D., Park, S., Song, H.: A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation. ACM Trans. Embed. Comput. Syst. 6, 3. (2007)
7. Park, S., Jung, D., Kang, J., Kim, J., Lee, J.: CFLRU: a Replacement Algorithm for Flash Memory. In CASES '06: Proceedings of the 2006 international Conference on Compilers, Architecture and Synthesis For Embedded Systems, pp. 234--241. Seoul, Korea (2006)
8. Wu G., Eckart B., He X.: BPAC: An Adaptive Write Buffer Management Scheme for Flash-Based Solid State Drives. In Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (2010)
9. Myers, D.: On the Use of NAND Flash Memory in High-Performance Relational Databases, Master of Sc. Tech. report, Massachussets Institute of technology, (2008)

10. Caulfield, A.M. , Grupp, L. M., Swanson, S.: Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications, ACM Architectural Support for Programming Languages and Operating Systems, Washington, DC, (2009)
11. Stoica, R., Athanassoulis, M., Johnson, R.: Evaluating and Repairing Write Performance on Flash Devices, Fifth International Workshop on Data Management on New Hardware (DaMoN 2009), Providence, Rode-Island (2009).
12. Leventhal, A.: Flash Storage, ACM Queue, (2008)
13. Kim, Y., Tauras, B., Gupta, A., Nistor, D.M., Urgaonkar, B.: FlashSim: A Simulator for NAND Flash-based Solid-State Drives, Tech. Report CSE-09-008, Pensylvania (2009)
14. Ganger, G.R., Worthington, B.L, Patt, Y.N.: The Disksim Simulation Environment Version 3.0 Reference Manual, Tech. Report CMU-CS-03-102, Pittsburgh (2003)
15. Forni, G., Ong, C., Rice, C., McKee, K., Bauer, R.J.: Flash Memory Applications, In Nonvolatile Memory Technologies with emphasis on Flash, edited by Brewer, J.E. and Gill, M., IEEE Press Series on Microlelectronic Systems, USA (2007)
16. Chung, T.S., Park, H.S.: STAFF: A Flash Driver Algorithm Minimizing Block Erasures. Journal of Systems Architectures, 53(12): 889-901, (2007).
17. Park, D., Debnath, B., Du, D.: CFTL: A Convertible Flash Translation Layer with Consideration of Data Access Pattern, Tech Report, University of Minnesota (2009)
18. M-Systems: Flash Memory Translation Layer for NAND Flash (NFTL), (1998)
19. Wu, C.H., Chang, L.P., Kuo, T.W.: An Efficient B-Tree Layer for Flash-Memory Storage Systems, ACM Trans. On Embedded Computing, Vol 6, Issue 3 (2007)
20. Shinohara, T.: Flash Memory Card with Block Memory Address, United States Patent, No. 5,905,993 (1999)
21. Kim, B.S., Lee, G.Y.: Method of Driving Remapping in Flash Memory and Flash Memory Architecture Suitable Therfor, United States Patent, No. 6,381,176 (2002)
22. OLTP Traces, UMass Trace Rep. http://traces.cs.umass.edu/index.php/Storage /Storage
23. Storage Performance Council, http://www.storageperformance.org
24. Micron: Small Block vs. Large Block NAND Flash Devices, Micron Technical Report TN-29-07, http://download.micron.com/pdf/technotes/nand/tn2907.pdf (2007)
25. Lucchesi, R., "SSD Flash drives enter the enterprise". Silverton Consulting. accessed on 2011-08, (2008).