An FTL-agnostic Layer to Improve Random Write on Flash Memory

Brice Chardin, Olivier Pasteur, Jean-Marc Petit

The original publication is available at www.springerlink.com:
https://doi.org/10.1007/978-3-642-20244-5_21

An FTL-agnostic Layer to Improve Random Write on Flash Memory

Brice Chardin[1,2], Olivier Pasteur[1], and Jean-Marc Petit[2]

[1] EDF R&D, France
[2] Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France

Abstract. Flash memories are considered a competitive alternative to rotating disks as non-volatile data storage for database management systems. However, even if the Flash Translation Layer – or FTL – allows both technologies to share the same block interface, they have different preferred access patterns. Database management systems could potentially benefit from flash memories as they provide fast random access for read operations although random writes are generally not as efficient as sequential writes. In this paper, we propose a simple data placement algorithm designed for flash memories, to reorganize inefficient random writes in a quasi-sequential access pattern. This access pattern is first established encouraging for a subset of flash devices by identifying a strong correlation between spatial locality and write performances, with a distance being defined to quantify this effect. This design is then validated by a formalization with a mathematical model, along with experimental results. With this optimization, random write potentially become as efficient as sequential write, improving random write speed by up to two orders of magnitude.

1 Introduction

For the sake of interchangeability, many flash memories include a Flash Translation Layer – abbreviated as FTL – to comply with the block interface, a rotating disk legacy. In addition to providing block write and read operations, the FTL manages flash chips complex writing mechanism. However, this layer is implemented with proprietary and undocumented software, which makes flash devices appear as "black boxes" from a system's point of view [3].

Advantageously, this FTL allows a straightforward substitution between both storage technologies. Yet, most database management systems include rotating disks-oriented optimizations, which are not relevant for flash memories. Even if both technologies use the same block interface, they have different preferred access patterns. Database management systems could potentially benefit from flash memories as they provide fast random access for read operations. Still, for FTL-based devices, random writes are generally not as efficient as sequential writes [5] and most optimization techniques for flash memories relate to this specific issue.

In this paper, we identify a strong correlation between write performances and spatial locality for a subset of FTL-based devices; and define a distance to quantify this effect. From this property, we propose a simple data placement algorithm, which trades flash memory space for random write performances. Its efficiency is validated by a formalization with a mathematical model, along with experimental results. With this optimization, random write potentially become as efficient as sequential write, improving random write speed by up to two orders of magnitude.

The rest of this paper is organized as follow. Section 2 introduces NAND flash memories and different types of mapping used in the FTL. Section 3 emphasizes the importance of locality on these devices for write performances and defines a distance between consecutive writes to quantify this effect. In section 4, we derive from this property an optimization technique for random writes, using an indirection layer to minimize this distance, thus avoiding scattered writes. In section 5, we present an approximate model for this algorithm. The results of both our experiments and model are reported in section 6. Related works are described in section 7. Then, section 8 summarizes the contributions of this paper.

2 NAND Flash Memories

NAND flash memory is a non-volatile storage technology, which allows three low-level data-access operations: read, write (or *program*) and erase. Still, erasing is performed at a different granularity than reading or writing: NAND flash chips are divided into blocks that can be erased independently, each block containing a fixed number of pages, each of which being individually accessible for reading or writing. As overwrites are not allowed, a full block must be erased prior to writing on one of its already used page. Additionally, pages within a block must be written sequentially.

To handle this complex writing mechanism, most flash memories include a Flash Translation Layer (FTL) that redirects writes on available (erased) pages and stores the associations between the logical sector identifier and its physical location in an address translation table.

In most cases, this translation operates on a page-level basis or on a block-level basis [6]. With a page mapping FTL, each logical page has its associated physical page. After an overwrite, the translation table is updated with the new physical location and the old physical location is marked as obsolete to be reclaimed by a garbage collection mechanism. With a block mapping FTL, each logical block has its associated physical block and an additional logging area, which consists of log blocks. When a page is overwritten, new data are appended to the last log block. Garbage collection merges a data block with all its associated log blocks by copying every valid page on a new (erased) data block and updating the translation table.

Each mapping granularity has its own drawbacks. Page mapping has a higher memory overhead because of its larger address translation table, while block

mapping performances are highly dependent on empty blocks availability, to serve as log blocks.

3 Write Spatial Locality for FTL-based Devices

As FTL enclosed in flash devices are usually proprietary and undocumented, studies have been conducted to identify preferred write access patterns for such devices. In [2], Birrell et al. identify a strong correlation between the average latency of a write operation and the gap between writes, as long as this gap is less than the size of two flash blocks. They conclude that write performance varies with the likelihood that multiple writes will fall into the same flash block, which is a manifestation of an underlying block or hybrid-mapping FTL. As a result, a fine-grained mapping is mandatory for high performance flash memories, but we believe that such a mapping can be efficiently provided by an additional layer, distinct from the FTL. Indeed, Wang et al. study in [12] the effectiveness of log-structured file systems for flash-based DBMS, since these file systems tend to write large data blocs in sequence. Their experiments validate potential benefits as they achieve up to x6.6 performance improvement.

uFLIP [3] is a component benchmark designed to quantify the behavior of flash-memories when confronted to defined I/O patterns. Some of these patterns relate to locality and increments between consecutive writes. Their results confirm that localizing random writes greatly improve efficiency and large increments lead to performances which could be even worse than random writes.

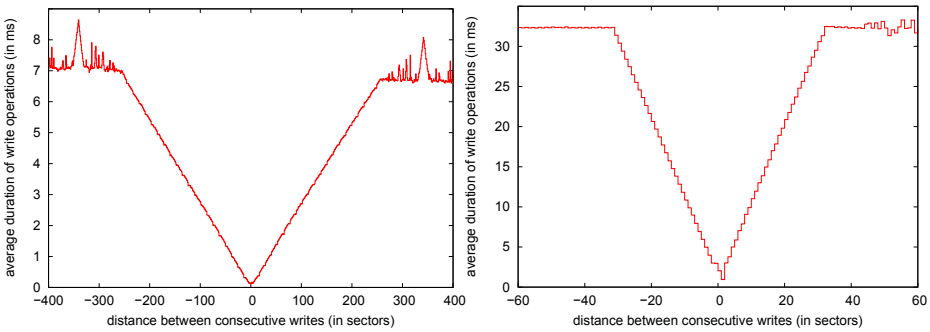
We propose a similar approach to quantify the effect of spatial locality on FTL-based devices, by introducing a notion of distance between consecutive writes. In our experiments, the average write duration for each distance d is evaluated by skipping $|d|-1$ sectors between consecutive writes. This metric can be negative to discriminate between increasing and decreasing address values. From the results of previous works, we conjecture a usual behavior where, up to a distance $d_{\max}$, the average cost of a write operation $cost(d)$ is approximately proportional to d.

To validate this assumption, we measured the effect of distance on a variety of flash devices. Although individual write durations are erratic, their average value converge when this access pattern is sustained. Figure 1 shows that our assumption is verified for a flash-based SSD[3] and a USB flash drive[4].

Scattered writes (ie. $d \geq d_{\max}$) are typically 20 to 100 times slower than sequential writes for flash memories with a block-mapping FTL[3]. Consequently, and because of this proportional performance pattern, reducing the average distance between consecutive writes can significantly improve efficiency, even if strict sequential access (d=1) is not achieved. The optimization described in the following section focuses on this access pattern, skipping as little sectors as possible.

[3] SSD Mtron MSD SATA3035-032, sector size 4 KiB

[4] Flash chip HYNIX HY27UG088G5B with an ALCOR AU6983HL controller, sector size 4 KiB



a) SSD b) USB flash drive

Fig. 1. Influence of distance on write duration

4 Gathering Random Writes

Online transaction processing usually have a part of its workload constituted of small random writes. The optimization described in this section converts these random writes into sequential writes, while skipping sectors containing valid data, as this access pattern should increase performances on flash memories. With this optimization, sectors containing valid data (used sectors) and free (unused) sectors are mixed on the device. An additional indirection layer is used to redirect logical writes to unused sectors by minimizing the distance between consecutive writes.

To allow data retrieval, correspondences between physical and logical locations are stored in an address translation table, with every logical sector being associated with a physical sector. Unused physical sectors – not associated with any logical sectors in the address translation table – do not contain any useful data, and therefore constitute a pool of free sectors available for writing.

To overwrite a logical sector, data are assigned to a pool sector adjacent to the previous write. Then the logical-physical association stored in the address translation table is updated, the previously associated sector therefore being freed and added to the pool. Figure 2 illustrates how logical writes are assigned to physical locations, when writing successively on logical sectors 0, 3 and 0.

This optimization does not require garbage collection, as the size of the pool remains constant: physical sectors containing obsolete data are immediately added to the pool, and can be overwritten. Yet, as an independent and internal mechanism, the FTL might still use garbage collection to handle flash erasures.

Any logical access pattern, whether sequential or random, will lead to a quasi-sequential physical access pattern. Consequently, the average distance (and thus write efficiency) is determined exclusively by the proportion of pool sectors. As increasing pool size requires additional non-volatile memory space, this characteristic can be adjusted to obtain an expected efficiency. As a downside, sequential reads are also transformed into random reads. However, this behavior is not

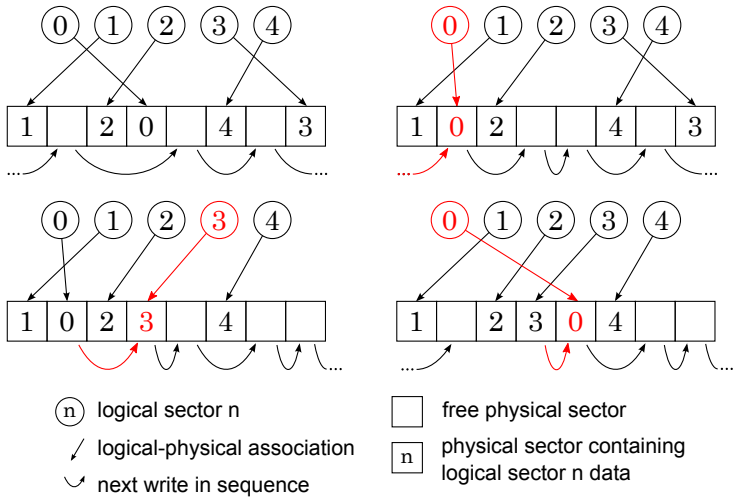


Fig. 2. Optimization overview

an issue for flash devices, as random reads are as efficient as sequential reads [3]. Read operations only induce lookups in the address translation table, which is a negligible overhead.

To prevent revisiting regions of the memory recently accessed, where pool sectors should have been exhausted, only positive distances are considered in this optimization. Additionally, the addressable space is assumed to be circular, in order to avoid handling edges differently.

The first data structure used by the redirection algorithm is the address translation table. This table – named T – binds every logical sector to a flash sector. As this optimization aims at minimizing the average distance between consecutive writes, the most recently written sector is referred to as $f_{\curvearrowleft}$. A simple version of the redirection algorithm involves the following operations when writing a logical sector l:

1: $f \leftarrow$ closest pool sector from $f_{\curvearrowleft}$
2: write data on f
3: $T(l) \leftarrow f$ {update the translation table}
4: $f_{\curvearrowleft} \leftarrow f$ {keep the reference of the most recently written sector}

Operation (1) – searching the pool sector closest to the previous write – has to be implemented carefully with an adequate data structure. In our implementation, to hasten lookups of this sector, we keep references of every sector in the pool in an ordered list P, where sectors are arranged by increasing distances from $f_{\curvearrowleft}$. $P(0)$ is thus the closest free sector from $f_{\curvearrowleft}$, followed by $P(1)$, etc.

Including this ordered list of pool sectors allows efficient retrievals of closest pool sectors. Nevertheless, this list has to be updated with each newly freed sector, whenever the translation table is altered. With this additional data structure, writing on a logical sector l implies the following operations:

1: $f \leftarrow T(l)$
2: write data on $P(0)$
3: $T(l) \leftarrow P(0)$ {update the translation table}
4: remove $P(0)$ from P {update the list of pool sectors}
5: add f in P

Operations (4) and (5) can be done asynchronously (ie. during the subsequent write in a write-intensive environment) without any consistency issue, as the list of pool sectors P can be rebuild from the translation table T. Consequently, P might not be up-to-date for each write request, which results in a slight increase of the average distance between consecutive writes if the closest pool sector from $f_{\curvearrowright}$ is not yet referenced in this list. However, this case appears infrequently for large pools and can be neglected.

5 Model

To estimate write speed improvement provided by this algorithm, we propose to model its behavior by evaluating the average cost of a write operation. This model is based on the simplifying assumption that pool sectors are uniformly distributed within flash sectors. This state is also supposed to be stable with occurring writes. Additionally, writing cost is expected to be determined exclusively by its physical distance from the previous write.

Under these approximations, the overall speed improvement can be evaluated given the probability to obtain each possible distance, and their associated costs. For this model, the following notations are used :

- $\mathbb{F}$ is the set of sectors accessible from the device (flash sectors),
- $\mathcal{D}(a, b)$ is the distance between two sectors a and b,
- $\mathbb{L}$ is the set of logical sectors,
- $\mathbb{P}$ is the set of pool sectors; by definition, $|\mathbb{P}| = |\mathbb{F}| - |\mathbb{L}|$.

Definition 1. *Let $p(d)$ be the probability that the sector $f_i \in \mathbb{P}$ which minimizes $\mathcal{D}(f_{\curvearrowright}, f_i)$ also verifies $\mathcal{D}(f_{\curvearrowright}, f_i) = d$, namely having a distance d between two consecutive writes.*

With the uniform distribution assumption, the probability $p(d)$ to get a distance d between two consecutive writes can be estimated as the ratio between favorable and possible distributions :

$$p(d) = \frac{\binom{|\mathbb{F}|-d-1}{|\mathbb{P}|-1}}{\binom{|\mathbb{F}|-1}{|\mathbb{P}|}} \tag{1}$$

The cost of a write operation conditioned by its distance from preceding write, $cost(d)$, can be approximated but also measured from the device, as shown in Fig. 1. For our evaluations, the latter is believed to be more accurate.

Given these two parameters, $p(d)$ and $cost(d)$, the average cost of a write operation, named $cost_{\text{avg}}$, amounts to :

$$cost_{\text{avg}} = \sum_{d=1}^{|\mathbb{L}|} p(d) \times cost(d) \tag{2}$$

Estimations from this model are reported in Sect. 6, together with experimental results. In addition to theoretical performance gains, resource usage can be quantified as this optimization trades server CPU and RAM, as well as flash memory space for writing speed.

CPU overheads occur when handling the translation table and the list of pool sectors during a write operation. These overheads relate to the following operations :

- search for the closest pool sector, which is $\mathcal{O}(1)$ when pool sectors references are stored in an ordered list,
- update the translation table, which is $\mathcal{O}(1)$,
- update the list of pool sectors, which is $\mathcal{O}(\log |\mathbb{P}|)$ with optimized data structures, such as skip-lists.

Updating the list of pool sectors is the only operation with significant CPU cost. However, as stated in Sect. 4, this update can be asynchronous. For read operations, looking up correspondences between logical and flash sectors in the address translation table results in constant CPU overhead, which is negligible compared to a flash sector read duration.

Server RAM overheads are caused by the translation table and the list of pool sectors maintained in main memory. These overheads amounts to $\mathcal{O}(|\mathbb{L}| \times \log |\mathbb{F}|)$ for the translation table, and $\mathcal{O}(|\mathbb{P}| \times \log |\mathbb{F}|)$ for the pool. Total RAM overhead adds up to $\mathcal{O}(|\mathbb{F}| \times \log |\mathbb{F}|)$.

As pool sectors are stored on the device, and do not hold any useful data, flash memory space overhead amounts to $|\mathbb{P}|$ sectors.

The last significant trade-off involves sequential writes. Since, with this algorithm, performances do not depend on the access pattern, logical sequential writes have the same performances as logical random writes. Existing attempts to sequentialize accesses would not bring any additional performance gain and should be discarded.

6 Results

To validate this optimization together with the model detailed in the previous section, the data placement algorithm is tested on both devices mentioned in section 3. These tests consist in evaluating the average cost of logical random writes for varying sizes of the pool.

Figure 3 shows experimental results and the model expectations for the USB flash drive. To compare with conventional access patterns, random write and sequential write iops (respectively 30 and 1060) are also reported on this figure.

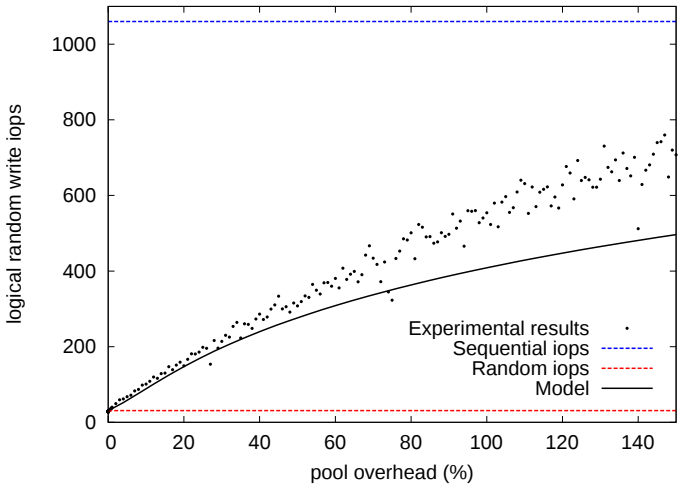


Fig. 3. Logical random write performance for 100,000 logical sectors

To obtain performances equivalent to sequential writes, consequent sacrifices have to be made in terms of flash memory space. In our experiment, 95% of sequential write efficiency is achieved when the pool is about three times larger than the logical address space. However, we achieved significant improvements over random writes with acceptable trade-offs, as we have a ten times improvement with 50% flash memory space overhead.

Contrastingly, writing speed on the SSD is improved with distances below $d_{\max} = 256$, instead of $d_{\max} = 32$ for the USB flash drive. As a result, notable improvements are achieved on the SSD with relatively lower sacrifices. Experimental results for this device are reported on Fig. 4.

Another noticeable difference was suggested by measurements obtained in Sect. 3: Fig. 5 focuses on small distance values. Remarkably, a quasi-sequential access pattern with a distance of 4 sectors between consecutive writes shows relatively good performances. Highest iops are achieved with a pool size of about 29,000 sectors, which results in an average – but still random – distance of 4. This property allows optimal performances to be achieved with much less overhead. Indeed, our optimization reaches 7796 average iops for 4KB logical random writes at a cost of only 687 KB of RAM, and 14.5% flash overhead for 800 MB of usable data space. Compared to physical random writes 134 average iops, performances are improved by $\times 58$.

Determining this optimal pool size is not straightforward, and depends on the sector size. With 16 KiB sectors, experiments give an optimum distance of 2; and about 1.6 for 32 KiB sectors. A possible explanation for this behavior is that interleaving favors non-zero sized skips to access multiple internal flash chips in parallel [13].

Unfortunately, this "peak" behavior might not be representative of flash solid-state drives. Among the twelve SSD with uFLIP results available, only

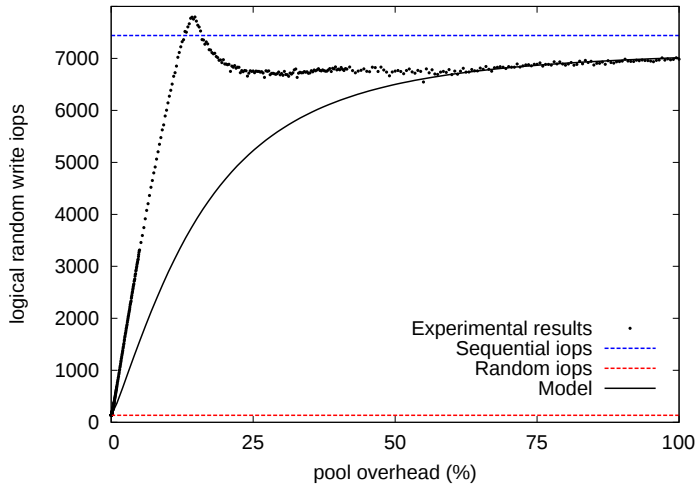


Fig. 4. Logical random write performance for 200,000 logical sectors

one (by the same manufacturer as ours) expose the same characteristic. This singularity is only a facultative additional benefit as it was not part of our initial assumptions. Our model might be closer of what we would expect with a regular SSD, and still shows considerable performances gains, such as a $\times 40$ improvement over random writes with 25% flash memory overhead.

Still, these results reveal some limitations of our model. One of its simplifying assumptions is that writing cost is determined exclusively by its physical distance from the previous write, which might not be accurate.

Moreover, experiments outperform our model as memory is accessed in only one direction – increasing physical addresses – to prevent revisiting regions of the memory where pool sectors should have been exhausted. This improvement over our theoretical uniform distribution reduces the average distance between consecutive writes.

7 Related Works

Many optimizations have been conceived with flash chips characteristics in mind. A frequent design avoids in place updates with log-based methods.

The In-Page Logging Approach [8] allocates a portion of each bloc to write updates of its pages. This optimization improves writing speed, as updates are written sequentially inside the erase unit, at the expense of more read operations. Garbage collection consists in merging data pages with their log sectors on a new empty block.

Page-Differential Logging [7] uses a similar approach, except page differential are logged. Writing is improved as differentials of multiple pages can be combined to fit in a single page. Also, differentials are recomputed from the original page

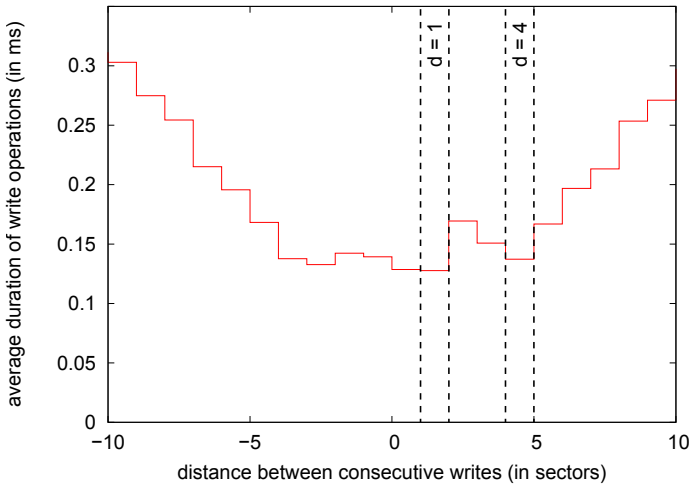


Fig. 5. Small variations of gaps for quasi-sequential writes

for each overwrite, which implies that reading a logical page involves at most two physical read (the original page and its last differential). The garbage collection mechanism is also improved, as merging a page with its current differential is not required (both can be copied separately).

Log-structured file systems, with a distinction between file systems designed for raw flash chips (without FTL) – like YAFFS, LogFS, JFFS – and those designed for block devices – like LFS – use methods comparable to the in-page logging approach, and therefore provide similar benefits and drawbacks. Additionally, I/O patterns of log-structured file systems for block devices when accessing multiple files tend to be of small size and scattered.

Regarding more specific use cases, B-File [10] is an abstraction layer for self-expiring items on flash memories. Depending on their expiration date, items are written sequentially in appropriate erase units to avoid copying valid data on deletion. Another approach defines an Append and Pack Layout [11], which divide the database in two separate datasets, respectively write-hot and write-cold. These datasets are written sequentially in multiples of the erase block size, with space reclamation when the memory is full.

The main differences between these approaches and our optimization are the necessity of a garbage collection mechanism and decreased read performances as a logical read rely on multiple physical reads. In contrast with these works, we optimize data access exclusively over the FTL. As a result, our approach is not applicable to raw flash chips.

RS-Wrapper [13] is a simple conversion between random writes and sequential writes for FTL-based devices. When random writes are adequately dense, their experiments show that reading the missing pages to overwrite sequentially the entire data range outperforms overwriting exclusively modified pages. However,

reorganizing these random writes in a quasi-sequential access pattern has not been tested.

FlashLogging [4] is an efficient mechanism for synchronous logging on multiple low capacity flash devices. While the use case differs from our proposition, this approach could be used to address the non-volatile issue of our current optimization. Indeed, data written to the device are volatile, since the address translation table is stored in RAM, and is needed to rebuild the database. Logging its modifications on additional flash devices could provide an efficient solution. This issue could also be managed by writing logical addresses together with data, similarly to the FTL internal functioning.

On a different but related subject, enterprise class SSD can provide better random write performance at the cost of additional RAM, processing power and spare blocks (not accessible from the host) [1, 9]. However, these designs focus on random write and provide invariably good performances for the entire device. Most database applications mix random and sequential accesses and do not require such homogeneous random write efficiency. By adding a software layer, our optimization permit using less expensive personal-class SSD with good, yet spatially limited, random write performances. This is also applicable to removable flash media, which have lessened hardware capabilities.

8 Conclusion

In this paper, we first introduced a notion of distance, and described its impact on flash memories write performances. Based on this property, we proposed a data placement algorithm, which significantly improves random write performances. Our contributions emphasize the importance of locality for these FTL-based devices, and we believe quasi-sequential access patterns to be of use for future data placement optimizations.

Compared to native write operations over the FTL, our optimization benefit from the host available RAM and processing power to improve random write efficiency on portions of the device. This method support localized performances adjustment, while flash memories offer homogeneous behaviors.

For the SSD used in our experiments, we achieved an improvement of up to $\times 58$ at a cost of only 14.5% flash overhead. In this best case scenario, this technique even caused random write to perform slightly better than sequential write, by 3.5%. This optimization is, to some extent, also applicable on flash memories with less capacity, as results with a USB flash drive show a $\times 10$ improvement with 50% flash overhead. Conjointly, we proposed a model to predict write performances, however, future works are needed to enhance its accuracy and help tradeoffs adjustments.

Another perspective relate to data volatility, which might be addressed in future works with solutions proposed in Sect. 7. Still, this optimization is already applicable for indexation or temporary tables, where volatility is acceptable.

References

1. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahy, R.: Design Tradeoffs for SSD Performance. In: 2008 USENIX Annual Technical Conference. pp. 57–70. USENIX Association (2008)
2. Birrell, A., Isard, M., Thacker, C., Wobber, T.: A Design for High-Performance Flash Disks. SIGOPS Operating Systems Review 41(2), 88–93 (2007)
3. Bouganim, L., Jónsson, B.T., Bonnet, P.: uFLIP: Understanding Flash IO Patterns. In: 4th Biennial Conference on Innovative Data Systems Research (2009)
4. Chen, S.: FlashLogging: Exploiting Flash Devices for Synchronous Logging Performance. In: 35th International Conference on Management of Data. pp. 73–86. ACM (2009)
5. Gray, J., Fitzgerald, B.: Flash Disk Opportunity for Server Applications. Queue 6(4), 18–23 (2008)
6. Kim, J., Oh, Y., Kim, E., Choi, J., Lee, D., Noh, S.H.: Disk Schedulers for Solid State Drives. In: 7th International Conference on Embedded Software. pp. 295–304. ACM (2009)
7. Kim, Y.R., Whang, K.Y., Song, I.Y.: Page-Differential Logging: An Efficient and DBMS-independent Approach for Storing Data into Flash Memory. In: 36th International Conference on Management of Data. pp. 363–374. ACM (2010)
8. Lee, S.W., Moon, B.: Design of Flash-Based DBMS: An In-Page Logging Approach. In: 33th International Conference on Management of Data. pp. 55–66. ACM (2007)
9. Lee, S.W., Moon, B., Park, C.: Advances in Flash Memory SSD Technology for Enterprise Database Applications. In: 35th International Conference on Management of Data. pp. 863–870. ACM (2009)
10. Nath, S., Gibbons, P.B.: Online Maintenance of Very Large Random Samples on Flash Storage. PVLDB 1(1), 970–983 (2008)
11. Stoica, R., Athanassoulis, M., Johnson, R., Ailamaki, A.: Evaluating and Repairing Write Performance on Flash Devices. In: 5th International Workshop on Data Management on New Hardware. pp. 9–14. ACM (2009)
12. Wang, Y., Goda, K., Kitsuregawa, M.: Evaluating Non-In-Place Update Techniques for Flash-Based Transaction Processing Systems. In: 20th International Conference on Database and Expert Systems Applications. pp. 777–791. Springer-Verlag (2009)
13. Zhou, D., Meng, X.: RS-Wrapper: Random Write Optimization for Solid State Drive. In: 18th Conference on Information and Knowledge Management. pp. 1457–1460. ACM (2009)