Yet Another Write-Optimized DBMS Layer for Flash-based Solid State Storage*

Hongchan Roh
Dept. of Computer Science
Yonsei University
Seoul 120-749, Korea
fallsmal@cs.yonsei.ac.kr

Daewook Lee
Dept. of Computer Science
Yonsei University
Seoul 120-749, Korea
daewook@gmail.com

Sanghyun Park
Dept. of Computer Science
Yonsei University
Seoul 120-749, Korea
sanghyun@cs.yonsei.ac.kr

ABSTRACT

Flash-based Solid State Storage (flashSSS) has write-oriented problems such as low write throughput, and limited lifetime. Especially, flashSSDs have a characteristic vulnerable to random-writes, due to its control logic utilizing parallelism between the flash memory chips. In this paper, we present a write-optimized layer of DBMSs to address the write-oriented problems of flashSSS in on-line transaction processing environments. The layer consists of a write-optimized buffer, a corresponding log space, and an in-memory mapping table, closely associated with a novel logging scheme called InCremental Logging (ICL). The ICL scheme enables DBMSs to reduce page-writes at the least expense of additional page-reads, while replacing random-writes into sequential-writes. Through experiments, our approach demonstrated up-to an order of magnitude performance enhancement in I/O processing time compared to the original DBMS, increasing the longevity of flashSSS by approximately a factor of two.

Categories and Subject Descriptors

H.2.2 [**DATABASE MANAGEMENT**]: Physical Design

General Terms

Design, Algorithms, Performance

1. INTRODUCTION

Flash-memory based Solid State Drives (flashSSDs) have been developed so as to resolve the limitations of the flash memory. FlashSSDs consist of a CPU, a RAM buffer, NAND flash memory chips, ECC (Error Correcting Code) modules, host/flash interfaces, and data/control buses that interconnect between each element of flashSSDs. In general, flashSSDs extend its capacity by embedding a plurality of the flash memory chips, and enhance write-throughput by utilizing parallelism between the flash memory chips. By the definition of Storage Networking Industry Assoication [1], flash-memory based Solid State Storage (flashSSS) includes all kinds of flash memory based storage devices such as flash memory chips, flashSSDs, and RAID solutions including a number of flashSSDs.

We define common problems of flashSSS caused by frequent write-operations as write-oriented problems. Raw flash-memory chips have asymmetric read/write throughput due to the much longer page-write latency regardless of the access pattern. It also has limited lifetime, which is determined by the maximum erase count of a flash-memory block. Consequently, frequent write-operations make the response time of flash memory worse, meanwhile shortening the lifetime of flash memory. For flashSSDs, access pattern to the devices has more significance than raw flash memory chips. FlashSSDs are vulnerable to the random-writes. Random-writes make the write-throughput of flashSSDs decrease, and it is questionable whether the sufficient longevity of a flashSSD device can be assured when a number of write-operations are requested in OLTP (Online Transaction Processing) environments.

Although IPL [4] and AppendPack [5] tried to address the write-oriented problems, each method has the following drawbacks. IPL proposed its own storage manager and buffer manager. In the buffer manager, additional memory space called in-memory log sector is allocated for every DBMS-page. Whenever a DBMS-page is updated, the dirty region is logged into the in-memory log sector. If the DBMS-page is evicted by the buffer manager, only the in-memory log sector is written to the log region, not the entire DBMS-page. Since the log-sector is smaller than the DBMS-page, the IPL storage manager is able to reduce page-writes considerably. However, if multiple log sectors for a DBMS-page are written to the log region, IPL induces additional cost to read the multiple log sectors when reading the DBMS-page. AppendPack replaces random writes into sequential ones through a new data layout. AppendPack adopts a mapping table analogous to the FTL mapping idea of flashSSDs, and contiguously writes the evicted DBMS-pages to the storage space. The written locations of the DBMS-pages are mapped to the DBMS-pages by the mapping table. However, this approach is not enough to

*This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0010689). This work was also supported by Seoul Metropolitan Government 'Seoul R&BD Program(PA090903)'.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

resolve the write-oriented problems. Merely replacing access patterns cannot contribute on lengthening lifetime of flashSSDs.

In this paper, we present a write-optimized layer of DBMSs to address write-oriented problems of flashSSS in OLTP DBMS environments. The layer consists of a log buffer called write-optimized buffer, the corresponding log-space, and an in-memory mapping table. The layer is closely associated with a novel logging scheme called InCremental Logging (ICL). The write-optimized buffer absorbs the evicted DBMS-pages by collecting dirty regions of the DBMS-pages, turning the dirty regions as logs. When it becomes full, it flushes logs into the log-space called ICL log-space. Through the ICL scheme, the log is incremented so that retrieve operation of an up-to-date DBMS-page can read only one log-page including the most recently incremented log for the DBMS-page. The mapping table maintains the links from each DBMS-page to the corresponding log-page.

The contributions of this paper are summarized as follows. We presented a feasible way to resolve write-oriented problems of flashSSS by reducing page-writes with the least compromise of increasing page-reads. Moreover, our approach replaces random-writes into sequential-writes meanwhile reducing page-writes. The write-optimized layer achieves a significant performance enhancement and lengthens the lifetime of flashSSS devices in the OLTP environments.

2. THE WRITE OPTIMIZED LAYER

2.1 The overall architecture

The write-optimized layer, which consists of one write-optimized buffer, the corresponding log space, and an in-memory mapping table, locates between the storage manager and flashSSS. This layer is closely associated with a novel logging scheme called InCremental Logging (ICL).

2.2 Write-optimized buffer

The traditional way of a buffer replacement operation is to choose a victim DBMS-page and write the entire DBMS-page to the corresponding location on hard-disks as presented in Figure 1. Four buffered DBMS-pages are evicted in turn and thus four DBMS-pages are updated on the flashSSS. We appended a buffer for reducing page-writes, called Write-Optimized Buffer (WOB), between the DBMS buffer cache and flashSSS. This buffer absorbs the frequent DBMS-page writes by storing only the dirty regions and turning each of them into a log. The log consists of two parts, the header and the content of the log. The header includes the information to identify the evicted DBMS-page, pairs of the offsets and the sizes of the updated regions inside the DBMS-page. The content is the updated tuples or the updated portions of the DBMS-page.

It is obvious that the total size of produced logs of each evicted DBMS-page is considerably smaller than the entire DBMS-page size. Therefore, the WOB is capable of literally buffering numerous DBMS-page write operations within the limited main memory space. If the main memory space allocated for the WOB becomes full, a flush operation to write the accumulated logs into log pages is performed. In Figure 2, all the logs in the buffer are flushed into one log-page. Four DBMS-page writes are reduced into one log-page write compared to the traditional approach depicted in Figure 1.

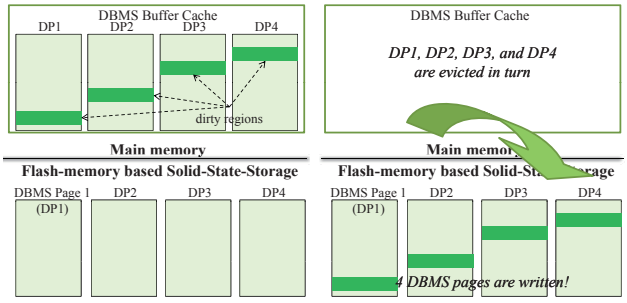


Figure 1: The traditional buffer replacement

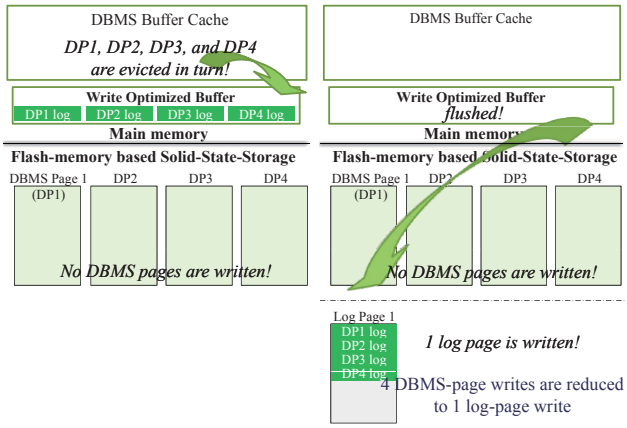


Figure 2: The naive write-optimized method

2.3 InCremental Logging

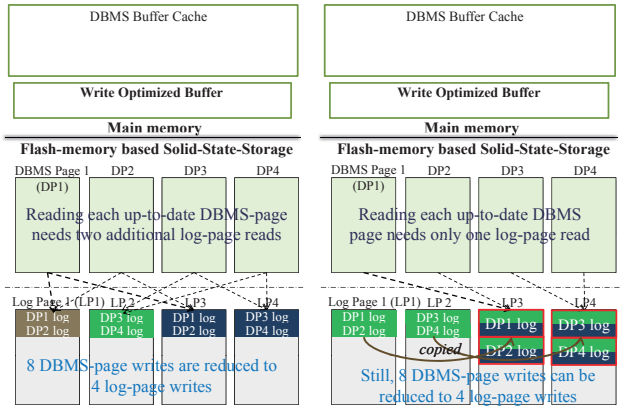
Page-writes can be significantly reduced by the WOB (Write-Optimized Buffer) and its flush operation. Another example of the flush operation is presented in Figure 3, where 8 DBMS-page writes have been replaced into 8 logs, and 4 log-pages are written by the 4 flush operations. This example also demonstrates a representative drawback of the naive write-optimized method. This approach causes excessive page-read operations when a DBMS-page is read. In Figure 3 (a), in order to read up-to-date DP1, the DBMS has to conduct 3 read operations: DP1, LP1, and LP3. Since the log size is significantly small compared to the DBMS-page size, it is possible that the logs corresponding to small portions of the DBMS-page scatter over numerous log-pages.

In order to address this problem, we designed a novel logging scheme named InCremental Logging (ICL). The problem is based on the fact that scattered logs over multiple log-pages incur additional page-reads when an up-to-date DBMS-page is read. Unless the logs to the DBMS-page scatter over multiple log-pages, the additional read-cost can be eliminated. In the ICL scheme, whenever the WOB flushes its logs to a new log-page, it copies the former logs of the DBMS-pages that the current logs belong to, from the previously written log-pages. Then the WOB writes the current logs and previous logs together into a new log-page. This process is represented in Figure 3 (b). The first flush operation of the WOB wrote the log belonging to DP1 and the log belonging to DP2 into LP1. Then, the second flush operation wrote the DP3 and DP4 log into LP2. In the second flush operation, no copy operation of previous logs was involved because the DP3 and DP4 log has no previous logs.

In the third flush operation, however, the write-optimized buffer copied previous logs since it has to write the logs of DP1 and DP2 (marked dark blue) that have the previous logs (marked green) in LP1. Consequently, the copied previous logs were written into LP3 along with the new logs as presented in Figure 3 (b). The same process was iterated in the fourth flush operation. The previous logs, DP3 and DP4 log in LP2, were copied, and were written together with the new DP3 and DP4 log into LP4.

This ICL scheme makes the process reading up-to-date DBMS-pages far less expensive compared to the naive approach described in Figure 2. The process induces only one log-page read in the case of ICL. In Figure 3 (b), each DBMS-page (DP1, DP2, DP3, DP4) needs to read only one log-page (LP3, LP3, LP4, LP4, respectively). Although the naive approach causes two log-page reads in order to retrieve each up-to-date DBMS-page in Figure 3 (a), it induces up-to dozens of log-page reads according to how small the average log-size is compared to the DBMS size. Based on the fact that they are incremented, the logs stored in a log-page are denoted by ICL logs to differentiate them from the logs inside the WOB. The ICL log is a set of logs since it includes more than one log.

Even though the size of an ICL log is incremented, the capability of reducing page-writes still remains. The mechanism reducing page-writes is to process as many evicted DBMS-pages as possible, by converting them into relatively small logs and flushing the logs into a log page at once. By doing so, many write-operations to DBMS-pages are reduced into one log-page write. The only thing changed in ICL is that the log size can be incremented for every flush operation. As long as the size of an ICL log is less than that of a DBMS-page, the benefit is preserved, since another ICL log can be stored in the remaining space of the log-page after storing the ICL log. This is demonstrated in Figure 3 (b), where the write-optimized layer reduced 8 DBMS-page writes into 4 log-page writes.

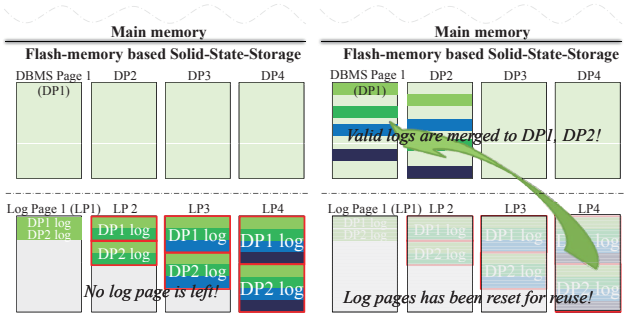


(a) Naive method (b) InCremental Logging

Figure 3: The InCremental Logging approach

The read process of a DBMS-page in the ICL scheme is as follows. First, the write-optimized layer reads the DBMS-page. If the DBMS-page has the corresponding ICL log, it reads the log-page containing the ICL log. Then, it extracts the ICL logs belonging to the DBMS-page from the log-page. Next, the logs inside the WOB are inspected. If the logs that belong to the DBMS-page exist, they are extracted, too. The ICL logs from the log-page and the logs from the WOB are reflected to the DBMS-page. Finally, the DBMS-page is retrieved to the DBMS buffer cache.

Thus far, we explained several features of ICL. A missing point is what happens after an ICL log grows enough to make no benefits any longer. To handle this, the ICL merge operation is performed. The sufficiently incremented logs are merged to the corresponding DBMS-pages. The ICL log-space can be recycled for later use. Figure 4 (a) describes the process while ICL logs for DP1 and DP2 were incremented up-to the half size of a DBMS-page. Therefore, no log page is left in the circumstance in Figure 4 (a). In order to resolve situations like this, the ICL scheme reflects the latest ICL logs into the corresponding DBMS-pages, and the log-pages are reset for reuse, as presented in Figure 4 (b). The reflecting process copies the latest ICL logs (the most incremented log for each DBMS-page) from log-pages to the corresponding DBMS-pages. In Figure 4 (b), the latest ICL log for DP1 is stored in LP4 so the log-contents of the ICL log were copied to the offsets of the update regions inside DP1, and a similar process was performed for the latest ICL log of DP2. The reset process of log-pages requires no actual page-writes since the log-pages can be merely logically invalidated without any physical page rewrite operations.



(a) Before the ICL merge (b) After the ICL merge

Figure 4: The ICL merge operation

Another missing point is how much page-read cost is expected in the copy process of previous logs. In the worst case, each new log to be flushed can have a previous ICL log stored in distinct log-pages. Even in this case, the additional read cost caused by the copy operations is limited to log-page reads as many as the total number of the new logs flushed by the WOB (which is the same as the number of evicted DBMS-pages). The additional read cost of ICL is relatively small compared to the naive write-optimized approach and IPL.

3. EXPERIMENTAL ANALYSIS

In the experiments, we compared the performance of our approach with the original postgresql DBMS, and postgresql DBMS where the AppendPack approach was adopted. We implemented our approach and the AppendPack inside the postgresql DBMS according to the papers. IPL was not compared because a main idea of the IPL approach should be adjusted for a fair comparison. Since it is closely integrated with raw flash memory chips, this feature is not compatible with the flashSSDs where manufacturers embed their own FTLs.

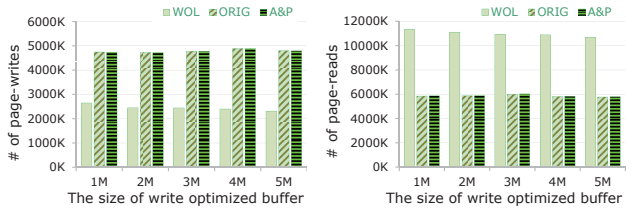
For the experiments, we used the following DBMSs where the three approaches are applied.

- ORIG: the original postgresql DBMS
- WOL: the postgresql DBMS where the Write-Optimized Layer is implemented
- A&P: the postgresql DBMS where the AppendPack approach is implemented

Since even in a single type of flashSSS, the performance varies according to the manufacturer of the device, it seems fairer to provide the invariable measures regardless of the flashSSS type, such as the number of I/O operations. In order to compare the performance of each approach on various types of flashSSS, we measured the number of I/O operations requested by the postgresql where each method is implemented. Then, we estimated the I/O processing time, based on the unit cost of each I/O operation on the various types of flashSSS, which we measured by using IOmeter benchmark [3] through another indepedent experiments.

To assess the performance associated with OLTP workloads, we used dbt2[2] which is an open-source tool generating TPC-C benchmark. The initial DBMS size was set to about 1GB (10 data warehouses). The number of DBMS client connections and the number of terminals per data warehouse were set to 100 and 10, respectively (100 terminals in total). 120,000 transactions were executed in each experiment. The default configurations of postgresql were intact except the buffer-cache size of 100MB. To be fair in main memory utilization, we increased the buffer cache size of ORIG by the WOB size and the maximum size of the ICL mapping tables. For A&P, only the buffer cache size is incremented by the WOB size since it requires the in-memory mapping table as well. A Linux machine equipped with 4 core CPU and 4GB RAM was used.

In the experiments, we compared our approach with the other methods, increasing the WOB size. We first measured the number of I/O operations requested by postgresql DBMSs. Then, the I/O processing time is estimated according to the different types of flashSSS devices.

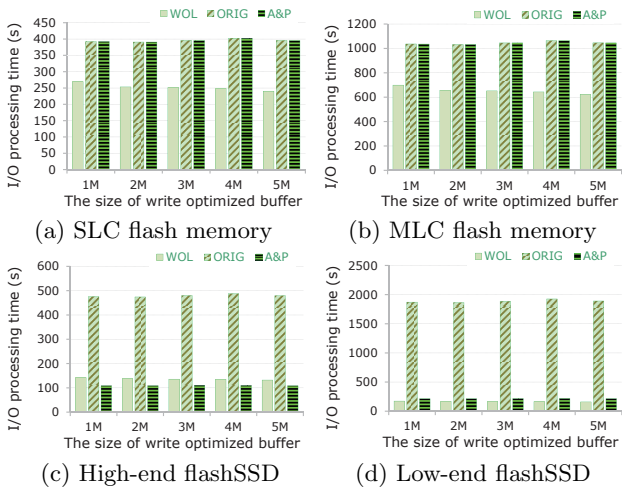


(a) Page-writes (b) Page-reads

Figure 5: I/O operations according to WOB size

Figure 5 shows the I/O performance of the three methods. It can be notified that WOL reduces the page-writes of ORIG by a factor of two. Contrarily, the increased page-reads do not exceed two times of the page-reads caused by ORIG. This is due to the fact that the ICL scheme needs only one log-page read to read an up-to-date DBMS-page. It can be expected that the lifetime of flashSSS will be lengthened by up-to a factor of two, since the lifetime of flashSSS is approximately inverse-proportional to the number of the performed page-writes.

Figure 6 presents the estimated I/O processing time of each method on various types of flashSSS. It can be notified



(a) SLC flash memory (b) MLC flash memory

(c) High-end flashSSD (d) Low-end flashSSD

Figure 6: Estimated I/O processing time

that our approach dominated other methods in all the experiments except the high-end flashSSD case. Even in the high-end flashSSD case, there is no significant gap between the total cost of WOL and A&P. WOL reduced 72% of the I/O processing time of ORIG whereas A&P reduced 77% of the I/O processing time of ORIG, on average. In case of low-end flashSSD, WOL enhanced the I/O processing time of ORIG by an order of magnitude (reduced 91% of ORIG's I/O processing time).

4. CONCLUSIONS

Flash memory has become one of the major storage devices. The manufacturers have been trying to expand the usage spectrum of the flash memory. To achieve this, the problems caused by frequent write-operation should be addressed first. In this paper, we focused on reducing page-writes to enhance the I/O processing time of DBMSs and the longevity of the flash based storage devices. Moreover, we developed a way to convert random-writes into sequential-writes, while reducing page-writes at the same time. Our approach has contributions on all types of flash-memory based storage devices including raw flash memory chips, and flashSSDs. The feasibility of the idea was examined through the implementation inside the postgresql DBMS. In the experiments, we found out that our approach improves the I/O processing time of the original DBMS by up-to an order of magnitude, lengthening the lifetime of flash-memory based storage devices by approximately a factor of two.

5. REFERENCES

[1] Advancing storage & information technology. `http://www.snia.org/home`.

[2] Database test suite. `http://osdldbt.sourceforge.net/`.

[3] Iometer official site. `http://www.iometer.org`.

[4] S.-W. Lee and B. Moon. Design of flash-based dbms: an in-page logging approach. In *SIGMOD Conference*, pages 55–66, 2007.

[5] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki. Evaluating and repairing write performance on flash devices. In *DaMoN*, pages 9–14, 2009.