Improving Write Performance Using Buffer Page On Flash-based DBMS

Mincheol Shin, Hongchan Roh and Sanghyun Park
Department of Computer Science, Yonsei University
134, Shinchon-Dong, Seodaemun-Gu, Seoul 120-749, Korea
{smanioso,fallsmal,sanghyun}@cs.yonsei.ac.kr

In recent years, the advances of flash technology occur to substitute HDD with SSD. The costly access time caused by mechanical properties of HDD makes random access very expensive. On the other hand, SSD has much smaller access time because SSD operates without mechanical principle. The most DBMS works on HDD until now, so the algorithms in DBMS are optimized on HDD and did not make use of fast random access of SSD. In this study, as a way to improve performance, we introduce BPAX(Buffered Partition Attributes Across) that the data layout to use a small unit for I/O on insert operation. The buffer for insert operation is added to PAX Layout.

Key Words: data layout, flash memory, ssd, query processing, buffer management, database

1. INTRODUCTION

Flash memory is more resistant to external shocks, and spend lower power than HDD. Furthermore flash memory has much smaller access time. But it has poor write performance and lower write endurance. SSD consists of flash memory packages and overcomes the drawbacks of a flash package. In the recent years, the capacity per price of SSD significantly increases and performance also improved, and this advances cause to substitute HDD with SSD in many fields.

If according to the trend of recent years SSD have dramatically increased storage capacity and a significant reduction in cost, a sole problem is time for HDDs to be completely replaced by SSDs. But most DBMSs until now have been based on HDD and developed. So many algorithms cannot maximize the benefits of SSD. In this view, there is some research to use benefits of SSD. For example, [4] improved performance of scan operation and join operation by using PAX layout.

In this study, We propose Buffered PAX layout (BPAX) that uses PAX layout for storing tuples and NSM layout for storing temporarily inserted tuples.

2. RELATED WORKS

There are several DBMS data layouts and most hard disk-based DBMSs uses NSM(N-ary storage model)[3]. In the case of NSM layout, tuples are written sequentially to the HDD. Unlike NSM, DSM(Decomposition Storage Model) [2], which is a column-based data layout, generates a sub-relation for each attribute and retrieves required attributes via a join. PAX(Partition Attribute across) is a hybrid method of DSM and NSM. Instead of creating sub-relations, minipages are created in page for an each attribute. Each values of the attribute are stored in the specific minipage.[1]

[4] read a smaller unit than page for PAX layout, using fast random access of flash memory. Because smaller size of read operation is occurred, the performance of projection is increased.

Recently, a special scan method for flashSSDs has been developed. Based on PAX layout, FlashScan [4] reads corresponding minipages rather than reading corresponding pages for selection operations. Therefore reduces the amount of I/Os required for the selection operations.

2.1 NSM Layout

NSM Layout, which is a traditional DBMS page layout commonly used in many commercial relational DBMSs. When every insert operation is occurred, a new tuple is appended to the tuple stored in pages. At the end of each page, a pointer-offset to point the location of each tuple exists.

The disadvantage of NSM layout is cache-misses that frequently occur, because the different attributes could be stored in a same cache block. To solve this, PAX layout has been proposed.

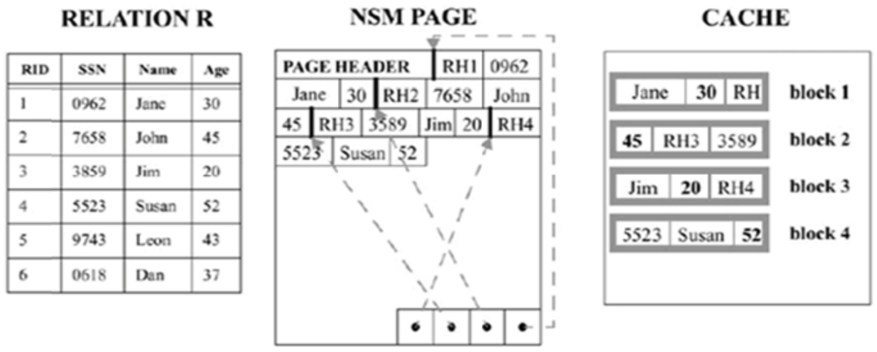

Figure 1. NSM layout.

2.2 PAX Layout

PAX layout has been proposed to improve cache performance. Basically, a data page is divided into several minipages. Each minipage only stores data for a single attribute. The size of each minipage is determined by a percentage of the attribute in the whole tuple. For example, If there is a relation of three attributes and each attribute is 2bytes, 4bytes and 8bytes, the ratio of each minipage size is 2:4:8. In this structure, because value of each attribute are consecutive in a minipage, cache miss are reduced for projection.

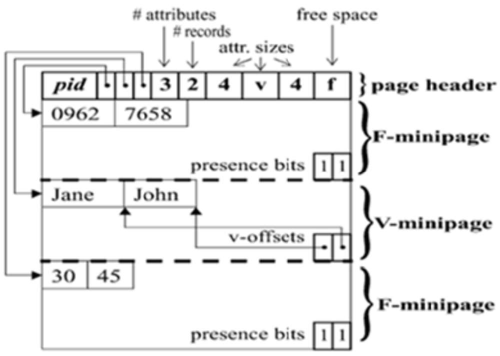

Figure 2. PAX layout.

2.3 Projection on PAX for flash memory

FlashScan [4] is based on PAX layout when data is stored in the flash memory. While the common DBMSs use page based I/O to perform scan algorithm, FlashScan performs read operation as smaller sized unit than page. Because, in the PAX layout, data for the same attribute are stored in the same minipage, we can read only projected attribute. Therefore, this scan operation demonstrates better performance than the existing operation when tuples are projected.

However, we cannot use sequential I/O because the algorithm read a part of the page. Therefore, in conventional magnetic hard-disks, the performance degrades because of the seek time. But onthe flash memory, it can improve performance because the seek time of flash memory is much smaller than magnetic disk.

According to the data presented in [4], the HDD seek time is about 3-4ms and sequential read is 100MB/s. Using these two materials, we can calculate the crossover point that cross performance of a random and sequential read. Assume the situation that DBMS reads sequentially while HDD seeking the position. In this situation, DBMS can read 300 ~ 400kb data. It means that sequential read is better than random read if skipped data between two random read operation is smaller than 300 ~ 400kb. Commonly, because the size of page is 8kb ~ 64kb, the sequential read is better in magnetic disk. Therefore, it is inefficient that data stored on magnetic hard-disks are read with the smaller unit than page size. On flash memory, the efficient size of the skipped data is less than 32kb. Query completion time of the scan operation was measured while increasing the page size by 32kb.

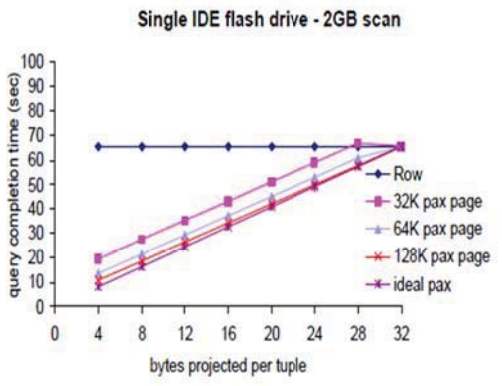

Figure 3. Projection operation on PAX for flash memory.

In Figure 3, the unit size for read operations was 4KB in *order* relation having eight 4-byte integer attributes. "Ideal" is theoretical data when seek time was not considered. Increasing the page size, the read performance was degraded.

Our method improves the performance for read operations based on this previous research. Moreover, our method improves the performance for write operations.

3. BPAX (BUFFERD PAX)

This chapter introduces BPAX, the variant of PAX. BPAX has two kinds of pages, buffer page and data page. Buffer page is structured as NSM layout and a single buffer page exists for each relation. Data page uses PAX layout. Every page is divided into several fragments similarly to minipages in PAX, called I/O blocks, and buffer manager read and write I/O block instead of a page. Consequently the size of each I/O block is less than the size of each page.

3.1 Idea

First, in the case that page layout is structured as only PAX layout, the insert operation make dirty pages. In detail, most of the I/O unit can be made dirty, because the page consists of I/O blocks and the number of I/O blocks are usually smaller than number of attributes. Therefore the whole I/O blocks in the page will be written when the page is selected as a victim by buffer manager. And the number of I/O blocks to be written will be the same when the insert operation occurs once or when the insert operation occurs twice before the page is flushed.

In contrast, the I/O blocks can be dirty partially if the page is structured as NSM layout, because the inserted tuple is not scattered over the page. The number of I/O blocks will be increased by the number of the inserted tuple before flush. The NSM layout makes lesser dirty I/O blocks than PAX layout. However, if the number of inserted tuples before flushing is large, the whole I/O blocks in the page will be dirty in both layouts.

For this reason, BPAX first gathers the inserted tuples in the NSM based buffer page, and then move the inserted tuples to PAX based data pages when the buffer page becomes full. This mechanism makes BPAX layout be able to reduce dirty I/O blocks, while utilizing the advantages of FlashScan [4].

3.2 Design

When read or write operations are occurred, conventional DBMSs load pages into a buffer pool. If the buffer pool becomes full, the buffer manager determines a victim buffer by using a buffer replacement algorithm.

Unlike existing DBMSs, BPAX request I/Os with a smaller sized unit (an I/O block) than page based I/O. A relation file for BPAX consists of one NSM-based buffer page and multiple PAX-based data pages. All the inserted tuples are stored in the buffer page, and when the buffer page becomes full, tuples are converted in the manner of PAX and moved to data buffer. The buffer manager manages buffers as I/O blocks.

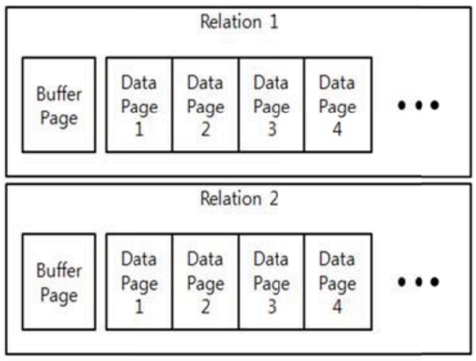

Figure 4. File structure of BPAX.

Algorithm 1 is the insert algorithm to insert tuple to relation and Algorithm 2 is to move tuples in buffer page to data page when buffer page is full.

```
Algorithm 1 Insert(t, R)
  Description
  Insert a tuple to the relation R
  Input   t : A tuple which will be inserted
          R : the relation which the t will be inserted into
  Output  none
  /* get a buffer page of R */
1 bufferPage = getBufferPage(R)
  /* check empty space of bufferPage */
2 if (there are sufficient freespace to insert t in bufferPage) then
3         insert t into bufferPage
4 Else
  /* move tuples in bufferPage to data page*/
5         emptyBuffer(p)
6         insert into bufferPage
7 Endif
8 refresh data pointer
```

```
Algorithm 2 emptyBuffer(R)
  Description
  Move tuples in a buffer page of the relation R to data pages.
  Input   R : the relation which the t will be inserted into
  Output  none
1 bufferPage = getBufferPage(R)
  /* find data page which has sufficient space to store tuples in buffer page */
2 temp = getDataPage(R, bufferPage)
  /* if we did not find it, allocate new data page */
3 if (temp == null) then
4         temp = allocateNewDataPage(R);
5 foreach tuple t in bufferPage
6         insert t into temp
7 delete all tuples in bufferPage
8 refresh pointer pointing end of data
```

4. Performance Prediction

As we studied prior, the performance is varied by the number of inserted tuples before the I/O block is flushed. The number of inserted tuples determines the size of data to be written. We use the factor r which means ratio of dirty part of the buffer page by insert operations.

To predict performance, we assume the situation that n tuples are inserted into a relation. The average size of tuple is notated as $|t|$, and then the total size of tuples is $n|t|$. Other notations used in our prediction are shown Table 1.

Table 1. The notation and its meaning

Notation	Meaning
|PAX|	The size of data page (PAX)
|Buffer|	The size of buffer page (NSM)
|NSM|	The size of data page (NSM)
N	Total number of tuples
|t|	Average size of tuples
R	Ratio of dirty part in the buffer page
|IO|	The size of IO unit

The write operation occurs when the page is dirty and determined victim. We can categorize the write operations as two kinds of flush by the kind of flushed page, buffer page flush and data page flush. First we can calculate approximately the number of buffer page flush and it is given by:

$$the\ number\ of\ buffer\ page\ flush = \frac{n|t|}{r|Buffer|}$$

And the number of I/O blocks to be written is determined by the r|Buffer| in I/O block. So the size of flushed data is:

$$Flush\ Cost\ for\ Buffer\ Page = \frac{n|t|}{r|Buffer|} \times |IO| \left\lceil \frac{r|Buffer|}{|IO|} \right\rceil$$

Data page flush occurs when the buffer page is full. So the number of data page flush is

$$the\ number\ of\ data\ page\ flush = \frac{n|t|}{r|Buffer|}$$

The write cost to write data page is varied by the size of attributes. So we notate it just as WriteCost. Then the cost of data page flush is:

$$Flush\ Cost\ for\ Data\ Page = \frac{n|t|}{r|Buffer|} \times WriteCost$$

We derive the cost model for PAX only data layout and NSM only data layout through similar way. It is shown below, Table 3:

Table 3. Cost model for BPAX, PAX and NSM layout

Data layout	Cost model
BPAX	$\frac{n\|t\|}{r\|Buffer\|} \times \|IO\| \left\lceil \frac{r\|Buffer\|}{\|IO\|} \right\rceil + \frac{n\|t\|}{r\|Buffer\|} \times WriteCost$
PAX only	$\frac{n\|t\|}{r\|Buffer\|} \times WriteCost$
NSM only	$\frac{n\|t\|}{r\|Buffer\|} \times \|IO\| \left\lceil \frac{r\|Buffer\|}{\|IO\|} \right\rceil$

Using the above cost models, the graph was drawn between the "r" and "cost". Parameters used in this study are as follows, Table 2:

Table 2. Parameters for figure 5 and 6.

Parameter	Value
Data page size	8 kb
Buffer page size	4 kb
Data page size (case of NSM)	8 kb
IO Unit size	2 kb
Tuple size	32 byte
Total inserted tuple	100000 tuples

The graphs in the figure 5 and 6 show the size of written data according to r. The graph in figure 5 is the result of prediction when WriteCost equals to |PAX| and The graph in figure 6 is the result of prediction when WriteCost equals to |PAX|/2. In both, BPAX shows better performance than PAX from r=0 to r=0.5. If r is greater than 0.5, the written data of BPAX is larger than PAX. However, r is usually lesser than 0.5 according to our trace of r.

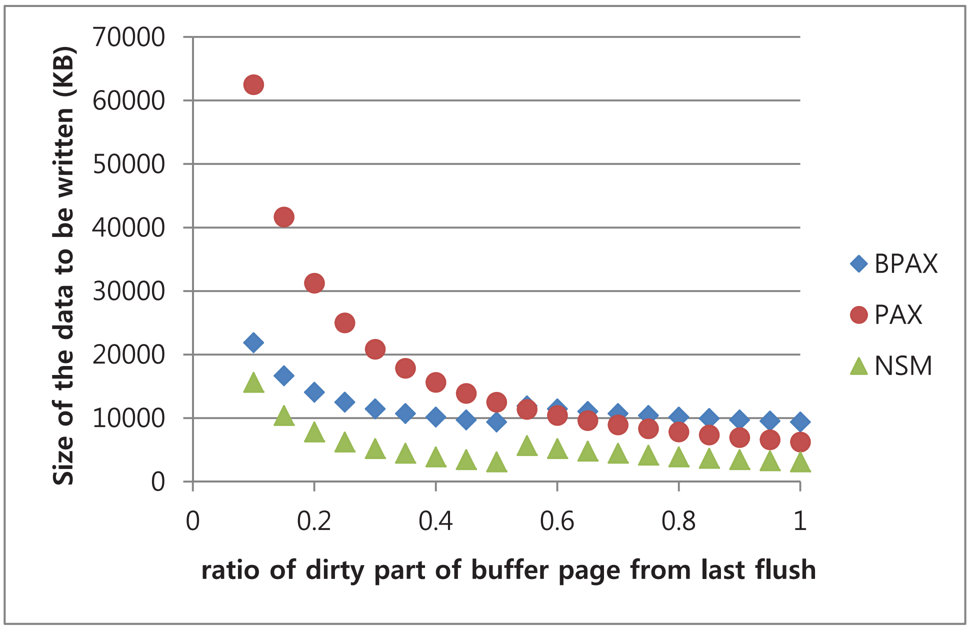

Figure 5. Graph about written data to the disk by r when WriteCost is |PAX|.

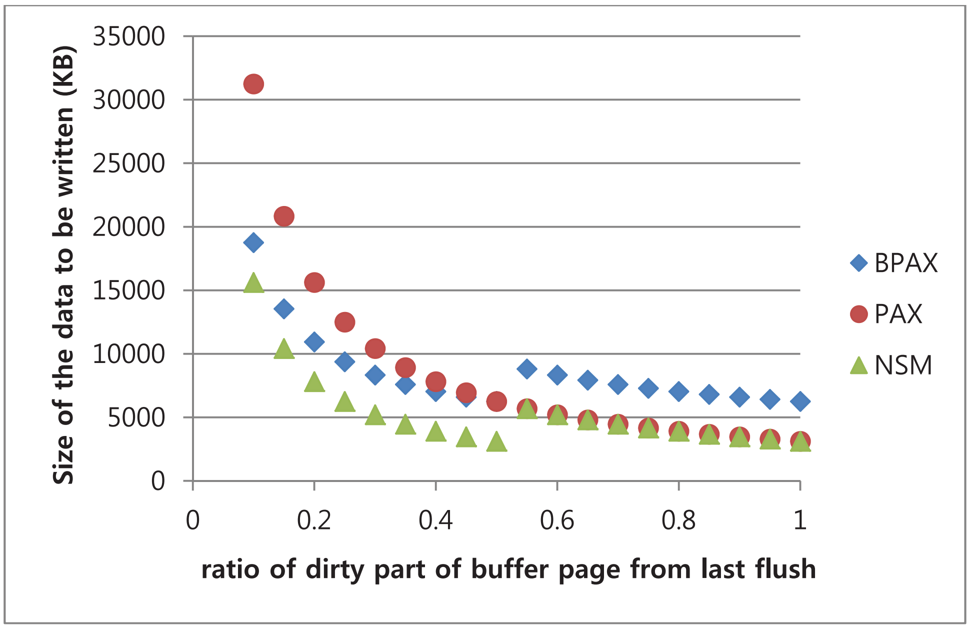

Figure 6. Graph about written data to the disk by r when WriteCost is |PAX|/2.

5. CONCLUSION

SSD is gradually spreading from the consumer market to the enterprise DBMS market. BPAX is a novel hybrid data layout for DBMSs that provides high performance for read and write operations both. BPAX gathers the inserted tuples in a NSM-based buffer page, and later moves the tuples to PAX-based data pages when the buffer page becomes full.. Consequently, BPAX demonstrates better performance for write operations than PAX layout while showing better performance for scan operations than NSM layout.

ACKNOWLEDGEMENTS

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(2012-0001997).

REFERENCES

[1] Ailamaki, A., DeWitt, D. J. and Hill, M. D.. "Data page layouts for relational databases on deep memory hierarchies," In *The VLDB Journal*, pp. 198-215, 2002.

[2] Copeland, G. P., "A decomposition storage model," In Proc. ACM SIGMOD International Conference on Management of Data, pp 268 - 279, May. 1985.

[3] Ramakrishnan, R. and Gehrke, J., "Database management systems," WCB/McGraw-Hill, New York, 2000.

[4] Shah, M. A., Harizopoulos, S., Wiener, J. L. and Graefe, G.. "Fast Scans and Joins using Flash Drives," In *Data Management on New Hardware DaMoN*, 2008.