A New Cache Management Approach for Transaction Processing on Flash-based Database

Da Zhou
Information school
Renmin University of China
Beijing, Country
e-mail:cadizhou@gmail.com

Xiaofeng Meng
Information school
Renmin University of China
Beijing, Country
e-mail: xfmeng@ruc.edu.cn

Zhichao Liang
Information school
Renmin University of China
Beijing, Country
e-mail: frankey0802@gmail.com

***Abstract[1]*—Database is an important part of web information system, the IO performance of which is a key factor of the responsiveness of the whole system. Existing databases usually adopt magnetic disks as storage devices. Now, low IO performance of magnetic disks becomes the bottle neck to the performance of the entire web information system. Flash memory, as a new electronic storage device, has high read/write performance when compared with magnetic disk. However existing disk-based databases can not take full advantage of flash memory, especially transaction processing because of the special characteristics of flash memory. Based on analysis of mismatch between "stealing page" and "not force page" in traditional transaction processing and flash memory, this paper proposes a new cache management mechanism for transaction processing. The average access time (AAT) and log-based write of our method are proposed to enhance the IO performance. Our method can not only eliminate the problems of traditional databases on SSD, but also make full use of flash memory to achieve higher IO performance for transaction processing. Besides the improvement of IO performance, our method can also extend the life span of flash memory and maintain wear-levering. The experiments show high efficiency of our method.**

Keywords-flash-based database; transaction processing; cache management;

I. INTRODUCTION

Database is an important part of web information system, the IO performance of which is a key factor of the responsiveness of the whole information system. With the rapid development of web information system, IO performance of databases becomes the bottle neck to the entire information systems. One of important reason is low IO performance of magnetic disks which are the storage media of databases. As we know, magnetic disks access data by magnetic head. During the course of access, mechanical latency is unavoidable, and then leads to the low performance of databases and web information systems.

Flash memory, as a new electronic storage device, has high IO performance compared with magnetic disk. The read speed of flash memory is more than 100 times of that of magnetic disk. There are two types of flash memory: NAND Flash and NOR Flash. Compared to NOR flash, NAND Flash has higher data density, less volume etc. Therefore most of recent research works are based on NAND flash. So does our work. In recent years, the capacity of flash memory chip doubled each year[1]. Now, Samsung has launched 256G flash SSD which is made up of 16G NAND flash chip. On the other hand, the price dropped 50 percentages each year[2] since 1995. With the increase of capacity and decrease of price, flash memory is regarded as the ideal data storage device for databases instead of magnetic disk in order to improve the IO performance of database.

But the traditional disk-based database can not provide high performance, especially transaction processing, on flash memory. The main reason is that traditional disk-based database can not take full advantage of flash memory due to the physical difference between magnetic disk and flash memory. Compared with magnetic disk, flash memory mainly has five different characteristics [3]: 1) no mechanical arms; 2) a two-level hierarchical structure: block/page; 3) three kinds of asymmetric operation: read/write/erase; 4) erase before overwrite; 5) limited erase times.

These physical characteristics lead different IO mechanism of flash memory. For example, flash memory adopts out-of-place update, while in-place update for magnetic disk. In order to apply the IO mechanism of disk-based database on flash memory, Intel develops Flash translation layer (FTL) technology [4] which re-maps physical addresses into logical addresses. Therefore traditional database can run on flash memory without any modifications by FTL. However, it has very poor performance because it leads to lots of write and erase operations.

In order to improve the performance of flash-based database, log-structured index technology [5] is proposed to reduce the number of writes. Log-structured index records updates as logs and flushes them into flash memory. Only at checkpoints, the data and its logs are merged as a new item. Log-structured index reduces the number of write and then enhances the write performance of database. However, when the database runs in long time, the log becomes longer and longer, and it would take long time to search the logs in order to get the latest data. Therefore the read performance is very low and lots of merge operations are unavoidable. Based on this problem, IPL [6] proposes a novel method to manage the logs by storing data and its logs in the same block. This work improves the write performance obviously. But as for hot data, the erase operations are triggered frequently and the space utilization is very low.

This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60833005, 60573091; National High-Tech Research and Development Plan of China (No:2007AA01Z155，2009AA011904)

After analyzing the IO mechanism of transaction processing on magnetic disk and the characteristics of the flash memory, we first propose a new cache management mechanism for transaction processing on flash-based database. The key contributions of this paper are summarized as follows.

1). We first analyze the challenges of disk-based transaction IO mechanism on flash memory in detail. When disk-based transaction processing is applied on flash memory, special data access interface of flash memory leads to new IO challenges.

2). We propose a new cache management mechanism for transaction processing on flash memory. Our method gives the uncommitted transactions higher priority than committed transaction in main memory. The Average Access Time (AAT) is used to decide which uncommitted transaction will be flushed into flash memory.

3). A Log-based write is proposed to reduce the times of writes. Although read performance decreases, the total IO performance is enhanced greatly. Besides this, it also decreases the quantity of garbage and extends the lifespan of flash memory due to the decrease of write times.

The rest of this paper is organized as follows. Section 2 explains the characteristics of flash memory and their challenges on disk-based database transaction processing. Section 3 surveys the related works. Then, in section 4, the basic concepts and algorithms of our method are described. Section 5 shows the experiment results. Finally we summarize this paper in section 6.

II. Problem Statement

This section first analyzes the characteristics of flash memory. Secondly the cache management mechanism of traditional disk-based transaction processing is introduced and analyzed. Finally the problems of applying traditional database on flash memory are shown.

A. NAND Flash memory characteristics

1) No Mechanical Latency: NAND flash memory is a pure electronic device and has no mechanical magnetic head like magnetic disk. Flash memory reads data by circuit, so random read almost has the same speed with sequential read because of no mechanical latency.

2) Two-level Hierarchical storage Structure: Unlike magnetic disks are made of sectors, NAND flash memory has a two-level hierarchical storage structure: block/page. A flash memory chip usually contains a number of blocks which commonly have 32 or 64 pages. Page is the basic access unit, the size of which usually is 512 bytes for Single-Level Cell (SLC) or 2K bytes for Multi-Level Cell (MLC).

3) Three Kinds of Asymmetric operations: Flash memory has three kinds of data operations: read/ write/erase. The granularity of read and write is page, while that of erase is block. For magnetic disk, the speed of read is the same with that of write. On the contrary, the time of read/write/erase of flash is 25 μs/200μs/1.5ms as shown in table 1. Therefore the cost of write is higher and that of erase is highest. Traditional disk-based database supposes that read has the same throughput with write. However this principle is not suitable for flash-based database and leads to low performance of flash-based database further.

4) Erasing before rewriting: Unlike rewrite of magnetic disk, erase must be executed before rewrite. Besides this, the unit of erase is block, but the unit of write is page. Therefore flash memory adopts out-of-place update. In this case, the original data becomes garbage, and needs to be collected.

5) Limited Erase Times: Flash memory has limited erased times, commonly from 106 to 107 times. In order to improve the lifespan of flash memory, wear-levering must be adopted to avoid that some blocks are worn out in advance. Large number of small writes, especially caused by transaction processing, will lead to too many erase operations.

B. Design of disk-based database

ARIES[7,8] algorithm is widely adopted by transaction processing of traditional disk-based DBMS. Most of existing commercial databases also adopt this algorithm. “Stealing frame” and “not forcing pages” are two important algorithms to manage the IO of transaction processing.

TABLE I
THE ACCESS SPEED OF FLASH MEMORY AND MAGNETIC DISK

	Flash Memory	Magnetic Disk
Read	25μs	9ms (Rand.)
Write	200μs	9ms (Rand.)
Erase	1.5ms	N/A

1) “Stealing frame” algorithm: As shown in figure 1(a), T1 reads A and updates A, then T2 reads B. At this time, although T1 has not committed, the memory has no more space to load B. At this scenario, the updated A is written to disk, and B is loaded.

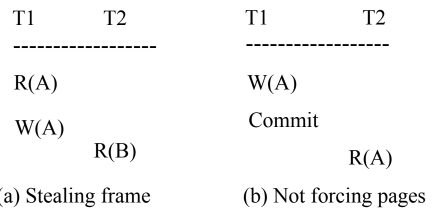

Fig. 1. “stealing frame” and “not forcing pages”algorithm

2) “Not forcing pages” algorithm: As shown in figure 1(b), T1 reads A and updates A, then T1 commits. Although T1 has committed, main memory still has free space, so the updated A is still in memory. Another scenario is next transaction T2 also needs to read A after T1 is committed. In this case, the updated A of T1 will not be written to disk.

Most of existing databases adopt magnetic disks as storage. The physical characteristics of magnetic disk make it easy to adopt the “stealing frame” and “not forcing pages” algorithm on magnetic disk. The reasons are as follows: 1) In-place update. As to “stealing frame”, rewrite is unavoidable. Because the magnetic disk adopts in-place updates, DBMS just need to overwrite the value at the cost of an IO operation. 2) High cost of IO. When large quantity of transactions run at the same time, the result of one transaction is probably the data another transaction need to load. After adopting “not forcing page” algorithm, a number of the IO operations which are used to access preceding data can be avoided, and then the performance will be enhanced.

C. *Problem with conventional designs*

When ARIES is applied on flash-based database, the performance is very low due to the special physical characteristics. The reason is analyzed as follows.

1) Out-of-place update: Flash memory usually adopts out-of-place update mechanism, so the largest challenge comes from random and small writes. When "stealing frame" algorithm is adopted, temporary data are probably written to flash memory. If the transaction aborts, the original value should be written back to the flash. The two operations above will lead to random updates. The cost of random update is very high for flash memory, so all write policies try to reduce the number of writes. Besides this, the temporary data on flash will become garbage after the transaction aborts. The garbage collection is a high cost operation for flash memory because database systems need to identify garbage data, clean data, move clean data to new block and erase old block. All of these characteristics show that disk-based "stealing frame" algorithm lowers IO performance.

2) High read/write speed: The main reason of using "not forcing pages" algorithm in magnetic disk is the lower read/write speed. When using flash memory, the IO cost will be reduced. According to Samsung datasheet as shown in table 2, the read/write speed of Samsung's 64GB flash-SSD is 100/80MB/s, while that of 80GB HDD is 59/60MB/s. If the flash memory chip and magnetic disk are used, the read/write speed of flash memory is 360/45 times of that of magnetic disk according to table 1. Therefore "not forcing pages" does not take full advantage of the characteristics of flash memory.

III. RELATED WORK

Our method draws on the concepts and principles in flash-based database, flash-based file system and log structured file system. In this section, we will discuss relevant examples of prior work in each of these systems.

A. *Flash-based Database*

TABLE II
THE CHARACTERISTICS OF SSD AND HDD

	SSD	HDD
Interfaces	2.5” SATA-Ⅱ	2.5” SATA-Ⅱ
Density	64GB	80GB
Weight	73g	365g
Read	100MB/s	59MB/s
Write	80MB/s	60MB/s

Flash-based databases are divided into block-emulate flash database and native flash database. Existing mobile or embedded databases, such as Microsoft SQL Server CE[2], are mainly block-emulate flash databases. FTL(Flash Translation Layer)[4] is used to emulate the flash memory as magnetic disk. Physical addresses are mapped into logical addresses by FTL, so the out-of-place update of physical addresses is mapped into in-place update of logical addresses. The design principle of this type of databases is to reduce write times as many as possible, even add more reads. These databases have advanced functions. Almost all the functions on disk-based databases are supported. However, the disk-based architecture of these databases can not take full advantage of the high IO performance of flash memory, such as random read.

On the contrary, native flash database is designed according to physical characteristics of flash memory, such as IPL, LGeDBMS[9]. Most of these databases use logs to record update, which take design principle of the PostgreSQL and log-structured file system. IPL organizes the data and its related logs in the same block, so the performance of update and merge is enhanced. These databases have excellent read and write performance. The wear levering and garbage collection are also implemented. However, only simple function is supported, such as primary key query. IPL can just support simple transaction. Of course, there are some databases based other kinds of flash memory, for example, TinyDB[10] on sensor nodes, PicoDBMS[11] on EEPROM. But these databases do not support transaction processing.

All of the existing databases, such as block-emulate flash database and native flash database, adopt the IO mechanism of disk-based database for transaction processing. Our method first designs transaction IO mechanism according to the characteristics of flash.

B. *Flash-based File System*

Flash-based file systems also have two design methods: block-device emulate approaches and native design approaches. Intel FTL, BFTL[12] and IBSF belong to the former approach. All of them are trying to offer the same interface with magnetic disk by flash translation layer. For example, BFTL efficiently implements a B-tree on NAND flash memory. On the contrary, native flash file system accesses the pages and blocks of flash memory directly by physical addresses. The native file systems mainly uses logs to record modifications, such as JFFS3[5]. The logs are kept in buffer firstly. At the appropriate time, the logs are written to flash in batch. Garbage collection is used to merge the logs with their data. Compared to block-device emulate approach, native file systems efficiently reduces the write times and extends the lifespan of the flash memory. Besides these, the IO bandwidth is also enhanced. Our method also uses logs to reduce the write times like the native file systems.

C. *Log Structure File System*

Sprite LFS[13] introduces the design principles of log based file system. All the data are represented as logs which are written to disk sequentially in order to reduce time of seek and rotation. BSD-LFS[14] enhances the performance of log structure file system by implementing block allocation method during the merge operations. BSD-LFS delays write and merges overwrite by making full use of RAM, and reduce the write times by writing logs in batch.

IV. OUR METHOD

In this section, the overall architecture of our method is firstly introduced. Secondly the detail replacement algorithm about committed and uncommitted transactions is explained. Log-based write method is introduced finally.

A. *Overall architecture*

Our design is to improve the IO performance of transaction processing by reducing the write times. The uncommitted transactions have higher priority than the committed transactions in memory when data of transactions must be written to flash memory. As shown in figure 2, the number of committed transactions reduces and that of uncommitted transactions with stealing pages increases when the number of total transactions increases. When only uncommitted transactions are in memory, AAT is used to decide which data will be replaced and written into flash memory. When the data need to be written into flash memory, logs, instead of the data, are written to flash memory.

B. Committed transactions IO mechanism

When RAM has large free space, the results of the committed transactions will still stay in RAM, so the following transactions will not need to access it from flash memory if they just need the data. Of course the temporary results of the uncommitted transaction will not be outputted to flash memory. When the number of the transactions increases, the free space of the memory becomes less and less. At last there is no free space for the new transactions. In this scenario, the results of committed transactions will be outputted to flash memory. In other words, the uncommitted transactions have higher priority than the committed transactions in memory.

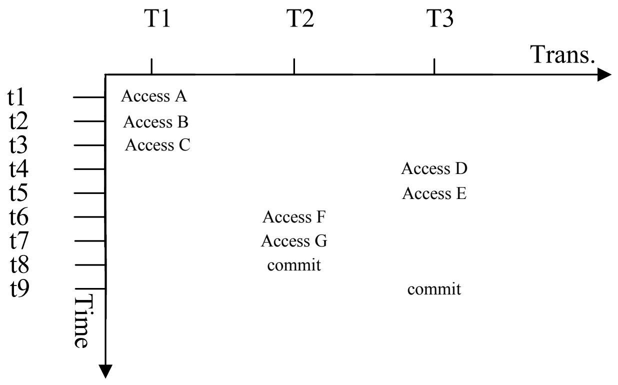

Fig. 3. Committed transaction replacement algorithm

The design object is to reduce unnecessary writes. If the temporary data are written to flash memory and the uncommitted transactions which associate those temporary data will possibly abort in the future, the temporary data written to flash memory will become garbage, and must be collected. In order to free this space, a block of data must be moved and the block must be erased. The garbage collection not only leads to large quantity of IO but also reduces the lifetime of flash memory because of write and erases operation. However, if the result of the committed transaction is written to flash memory, although it is possible that the data will be used by the following transactions, only some additional read operations are paid to load the data again. Because the flash memory has excellent read performance, so our algorithm is very suitable for flash memory.

As shown in figure 3, although T2 accesses F later than T3 accesses D, T2 will be replaced from memory earlier than T3 because T2 commits earlier than T3. According to our design, data of T3 is outputted earlier than that of T1.

C. Uncommitted transactions IO mechanism

When all data of the committed transactions are replaced from RAM, and there still are data should be loaded into RAM, the temporary data of the uncommitted transaction must be written to the flash memory. The temporary data of uncommitted transaction in RAM should be deleted because the temporary result leads to garbage collection which is a high cost operation.

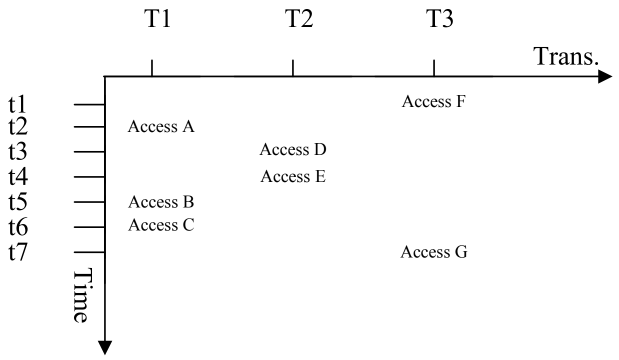

Fig. 4. Uncommitted transaction replacement algorithm

Our method uses Average Access Time (AAT) to decide which transaction should be replaced from RAM. Suppose a transaction has n pages data as P1, P2… Pn. Each page has an access time. For example, the access time of Pi is Ai. Ai is defined as the last access to the Pi. If the page is replaced, the access time will be set to the current time. Therefore the access time of each page is A1, A2…An. Our method defines the AAT as the average of the sum of all the access time. Each transaction has an AAT. Suppose there are n uncommitted transactions as T1, T2… Tn, so the associated AAT is AAT1,

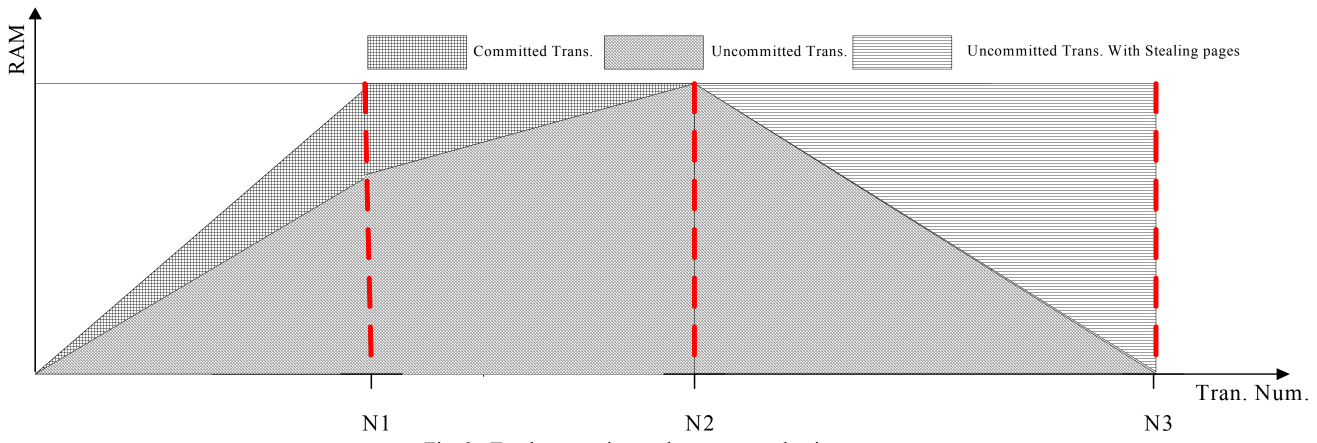

Fig. 2. Total transaction replacement mechanism

AAT2…AATn. According to AAT, the temporary data of associated transaction which has earlier AAT will be written to flash memory in higher priority. Every access will update the access time of the page as the current time, and then the AAT of the transaction will be changed. The memory will record AAT of every transaction.

If a transaction is created earlier, the AAT is less and the transaction is more possibly to be replaced from RAM. On the other hand a transaction is accessed more recently, the AAT of it is larger and the transaction will not be less possibly replaced from the RAM. As shown in figure 4, the AAT of T1, T2, T3 is 6.5, 3.5, 4, so the output order is T2, T3 and T1. Therefore our method reflects the real situation of the transaction replacement mechanism of the RAM.

D. *Log-based write mechanism*

In the disk-based database, when the transaction is replaced from the RAM, all related data are written to magnetic disk. According to the previous analysis, this mechanism is not suitable for flash-based database. In order to reduce the quantity of data to be written to flash memory, our method records the modifications as logs. When the transaction is replaced from the RAM, the modifications of transaction are organized as logs. The logs are written to the flash memory instead of data. With the reduction of the quantity of data to be written to flash memory, the IO performance is improved. On the same time, when the transaction aborts, the garbage which comes from the logs will be smaller than that of data written to flash memory, and then the garbage collection is easier.

There is a problem which must be concerned. If the transaction needs to read the data again, we must read the logs and the original data, merge the data and its log to get the latest data. Compared with reading the data that is directly written to flash memory, more reads are paid, and then the read cost is increased. According to the physical characters of flash memory, we know the write cost is eight times of that of read, so we just get better IO performance by reducing the write times at the cost of adding read times. The experiments show that this mechanism improves the whole IO performance.

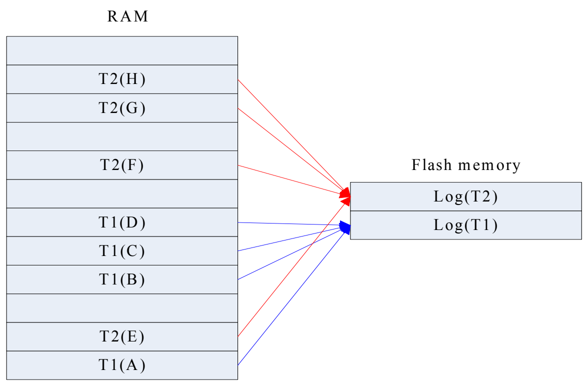

Fig. 5. Log-based write algorithm

In order to improve the read performance, based on the log-based write concept, our method adopts lazy replace method. When the transaction is replaced from RAM, the logs are written to flash memory, and the data is not replaced in RAM, only when new data is loaded into RAM, the data will be replaced. For example, if a transaction T is replaced from RAM and T has n pages. After the logs are written to flash memory, the n pages are still in RAM. When data is loaded into RAM, the n pages will be replaced one by one. In some scenarios, this mechanism will improve the system IO performance significantly. After the first page of transaction T is deleted and the system needs to load all data of transaction T, we just need to load the first page. The remaining data is still in RAM and does not need to be loaded, so quantity of IO is reduced and the performance is improved.

The logs of a page are usually less than a page. According to the physical characteristics of flash memory, a page can be written only once. Therefore the space utilization will be very low if we write the log of a page one time. In order to improve the space utilization, our method uses a log to record the modification of multiple data pages as shown in figure 5. Finally the length of a log reaches the size of a page.

V. Performance Evaluation

In this section, we implement simulation experiments by running disk-based transaction processing and our method on flash memory. The comparison experiments show the efficiency of our method.

A. *Setup for Experiment*

We run flash memory simulator on a computer with CPU 3.2GHZ and 1G DDR 667 RAM. The famous flash simulator, the NANDSIM of Linux, is used in our experiments. We suppose the flash memory has 1024 blocks, each block contains 64 pages which is 2Kbytes. The RAM for transactions is set to 16Mbytes, so 128 blocks of data can be maintained in RAM, and every transaction contains 16 pages. We record every accesses of the simulator, and get the total read/write/erase times. According to the Samsung datasheet [3] and table 2, we can get the IO performance.

B. *Transactions IO Performance*

In experiment of figure 6(a), 768 transactions run in the RAM. When 256 transactions are committed, each of other 512 uncommitted transactions only loads half of the data. Figure 6(a) shows the IO performance of ARIES on HDD, ARIES on SSD, and our method on SSD. According to the result, our method reduces by 63% IO time compared with the ARIES on SSD, even more on the write time. In the experiment 6(b), 1024 transactions run in the RAM at the same time. Every transaction loads a page of data in turn. Figure 6(b) shows the ARIES on SSD has poor write performance. After the temporary data are written to flash memory, additional erase operations must be paid to collect the garbage. Our AAT-based method can enhance the replace right ratio, reduce the temporary data, and then improve by 11% performance. As shown in Figure 6(c), we record the modifications of data as logs, and write the logs instead of data. Although the performance of read decreases 25%, the total performance enhances by 34% because write performance improves 75%. Therefore, log-based write mechanism can improve the whole IO performance. According to the experiments, the disk-based transaction mechanism has poor performance on flash memory, but our method make full use of flash memory and get high IO performance.

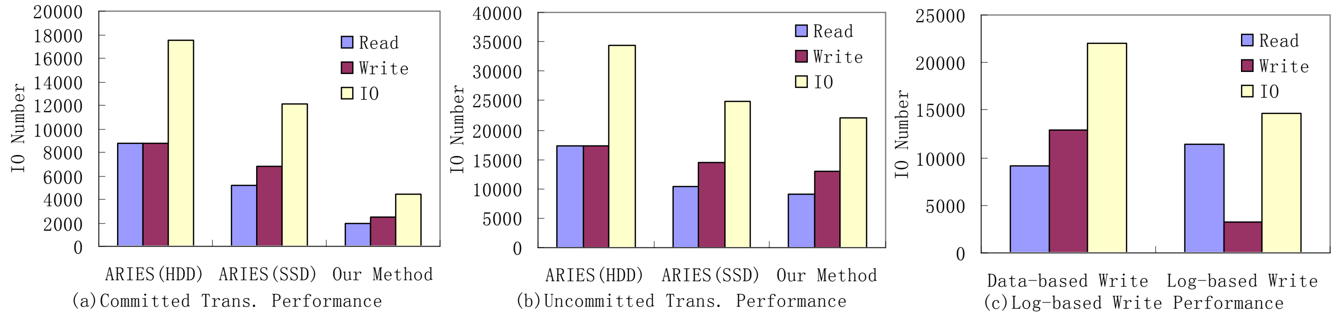

Fig. 6. the IO performance of ARIES on HDD and SSD, and our method on SSD

VI. CONCLUSION

Database is one of important parts in web information system. However the IO performance of database is limited by the IO characteristics of magnetic disk. With the development of flash memory, it brings a new chance for database to improve the IO performance basically. But the cache management of transaction processing in existing disk-based databases is not suit to flash memory. This paper analyses the challenges of IO mechanism of traditional transaction processing on flash memory. A new cache management mechanism for transaction processing is proposed. Our method can take full advantage of the characteristics of flash memory. We use Average Access Time (AAT) and log-based write to improve the IO performance. The experiments show the high efficiency of our method.

REFERENCES

[1] SUMSANG. 128G and 16G nand flash chip [Online]. Available: http://www.samsung.com/global/business/semiconductor/productList.do?fmly_id=161

[2] J. Gray. (2007, January). FLASH Opportunity for Server-Applications. [Online]. Available: http://research.microsoft.com/~Gray/papers/FlashDiskPublic.doc

[3] Samsung. (2006, November). 1G x 8 Bits / 2G x 8 Bits / 4G x 8 Bits NAND Flash Memory [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/139788/SAMSUNG/K9WAG08U1A.html

[4] Intel. "Understanding the Flash Translation Layer (FTL) Specification (Report style)," Intel Corporation, Application Note AP-684, December 1998.

[5] A. B. Bityutskiy, "JFFS3 Design Issues (Report style)," Tech. report, November 2005.

[6] S. W. Lee, B. Moon, "Design of flash-based DBMS: an in-page logging approach (Published Conference Proceedings style)," in Proc. of the ACM SIGMOD International Conference Management of Data, Beijing, China, June 12-14, 2007, pp. 55-66.

[7] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* (Book style). Morgan Kaufmann, 1993, pp.504-530.

[8] C. Mohan, D. J. Haderle, B. G. Lindsay, H. Pirahesh, P. M. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. Database Syst*, Vol. 17, no. 1, 1992, pp. 94-162.

[9] G. J. Kim, S. C. Baek, H. S. Lee, H. D. Lee, M. J. Joe, "LGeDBMS: A Small DBMS for Embedded System with Flash Memory (Published Conference Proceedings style)," In Proc. of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006, pp. 1255-1258.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," ACM Trans. Database Syst. Vol. 30, no. 1, 2005, pp. 122-173.

[11] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, "PicoDBMS: Validation and Experience (Published Conference Proceedings style)," In Proc. of the 27th International Conference on Very large Data Bases, Roma, Italy, September 11-14, 2001, pp. 709-710.

[12] C. H. Wu, L. P. Chang, T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems (Published Conference Proceedings style)," In Proc. the 9th International Conference, RTCSA, Tainan, Taiwan, February 18-20, 2003, pp.409-430.

[13] M. Rosenblum, J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Trans. Comput. Syst. Vol. 10, no. 1, 1992, pp. 26-52.

［14］ M. I. Seltzer, K. Bostic, M. K. McKusick, C. Staelin, "An Implementation of a Log-Structured File System for UNIX (Published Conference Proceedings style)," In Proc. USENIX Winter, 1993, pp. 307-326.