DOI: 10.3745/KIPSTD.2010.17D.6.395

PRMS: SSDs에서의 Page 재배치 방법

이 동 현[†] · 노 홍 찬[††] · 박 상 현[†††]

요 약

Solid-State Disks (SSDs)는 빠른 접근 시간, 적은 전력소모, 전기 충격에의 내성과 같은 장점으로 인해 하드 디스크를 대체 할 것으로 기대되고 있다. 그러나 SSDs는 임의 쓰기(random write)로 인한 수명 단축이란 단점이 있으며 이는 SSDs 컨트롤러의 구조와는 별개로 나타나고 있다. SSDs와 관련한 기존 연구는 컨트롤러의 더 나은 디자인과 쓰기 연산의 감소에 주력하였다. 본 연구는 동시에 쓰여지는 경향이 있는 여러 데이터 페이지를 연속적인 블록에 배치하는 방법을 제시한다. 이 방식은 우선 특정 기한 동안 쓰기 연산에 대한 정보를 수집한 후 상기 쓰기 연산에 대한 정보를 트랜잭션화 하여 frequent itemset을 추출하고 이를 연속적인 블록에 재배치하는 과정으로 이루어진다. 또한 본 연구는 frequent itemset의 page를 재배치할 수 있는 알고리즘을 소개한다. TPC-C 기반 실험에 있어 본 연구가 제안한 재배치를 수행한 결과 저장 기기 접근 횟수를 평균 6 % 감소시킬 수 있었다.

키워드 : SSDs, 플래시 메모리, Frequent Itemset Mining

PRMS: Page Reallocation Method for SSDs

DongHyun Lee[†] · Hongchan Roh[††] · SangHyun Park[†††]

ABSTRACT

Solid-State Disks (SSDs) have been currently considered as a promising candidate to replace hard disks, due to their significantly short access time, low power consumption, and shock resistance. SSDs, however, have drawbacks such that their write throughput and life span are decreased by random-writes, nearly regardless of SSDs controller designs. Previous studies have mostly focused on better designs of SSDs controller and reducing the number of write operations to SSDs. We suggest another method that reallocates data pages that tend to be simultaneously written to contiguous blocks. Our method gathers write operations during a period of time and generates write traces. After transforming each trace to a set of transactions, our method mines frequent itemsets from the transactions and reallocates the pages of the frequent itemsets. In addition, we introduce an algorithm that reallocates the pages of the frequent itemsets with moderate time complexity. Experiments using TPC-C workload demonstrated that our method successfully reduce 6% of total logical block access.

Keywords : SSDs, Flash Memory, Frequent Itemset Mining

1. Introduction

Solid state disk is now expected to play a key role in storage device system. Due to fast access time of NAND flash memory and SSD's own internal mechanism, the throughput of SSDs is superior to that of hard disks [1-4]. Hence, low power consumption and outstanding physical durability of SSDs accelerate the spread of SSDs for various areas to adopt them as major storage device. Despite of such merits, several drawbacks of SSDs exist since SSDs inherit shortcomings of NAND flash memory. Unlike hard disks, NAND flash memory is unable to overwrite a page, whereas writing an empty page is available. NAND flash memory must erase a block, a set of contiguous pages, before updating specific data located in a page. Such an inefficiency of NAND

※ This work was supported by National Research Foundation of Korea Grant funded by the Korean Goverment(KRF-2008-313-D00849).
This research was also 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)
† 준 회 원 : 연세대학교 컴퓨터과학과 공학석사
†† 준 회 원 : 연세대학교 컴퓨터과학과 박사과정
††† 종신회원 : 연세대학교 컴퓨터과학과 부교수(교신저자)
논문접수 : 2010년 4월 20일
수 정 일 : 1차 2010년 6월 3일, 2차 2010년 7월 19일
심사완료 : 2010년 7월 28일

flash memory has been issued early and many studies have focused on improvement of update mechanism [5, 6] or design of SSDs-specialized data structures [7, 8]. The life span of NAND flash memory is another drawback. A block becomes unusable after the number of erase operation on the block exceeds a certain number. Several wear-leveling algorithms were proposed to even erase counts of NAND flash memory block [9, 10]. Infeasibility of inplace-update and limited life span imply that random write is a major factor of performance degradation of SSDs. Random writes cause more erase operation on blocks and shorten the life span of NAND flash memory. Moreover, frequent random writes bring the internal fragmentation, decentralization of logically related data, which worsens the throughput of SSDs [11].

In this paper, we suggest a method called PRMS (Page Reallocation Method for SSDs) to reduce random writes without any modification of internal structure of SSDs. Since the internal structure of SSDs has substantial influence on the performance of SSDs, many SSDs manufacturers conceal their internal structure of SSDs. Our method is based on two assumptions. First, most workloads have I/O patterns, the non-uniform distribution of stored data. Second, many systems such as OS and DBMS mostly request a set of write operations rather than a single write operation in order to avoid I/O bottlenecks. If we can find write operations that are both simultaneously and frequently requested, pages of SSDs modified by those write operations can be reallocated in logically contiguous regions (i.e. contiguous logical blocks of SSD). We define the pages modified by write operations that are both simultaneously and frequently requested as the frequent page set. Reallocating the frequent page sets to continuous logical blocks enhance the performance of SSDs. In more detail, incoming writes to the frequent page set can be efficiently processed. After the reallocation, write operations which write the frequent page set can be processed as if those were sequential writes other than random writes. Write operations to contiguous logical blocks are probably faster than write operations to several random logical blocks.

The remainder of this paper is organized as follows. In the next section, we introduce the background of this paper. Section 3 introduces the overview of our method. The details of our method are described in sub-section 3.1, 3.2, and 3.3. Section 4 presents experimental results. Section 5 concludes the paper.

2. Background

2.1 Frequent Itemset Mining

Let $I = \{i1, i2 \cdots in\}$ be a set of items. Let $D = \{t1, t2, t3 \cdots tm\}$ is transaction-based database and each transaction is denoted as ti. Every transaction of D is a subset of I. Set S is called itemset if S is a subset of I and is called k-itemset if the number of elements of S is equal to k. The support of set S is the number of transactions including S as a subset. If the support of S is greater than a given threshold (i.e. minimum support count threshold) then the itemset is a frequent itemset. An itemset S is called maximal frequent itemset if no superset of S is a frequent itemset.

2.2 Related Works

The concept of association rule, which includes the concepts of the frequent itemset, was first introduced in [12]. A classic algorithm for mining frequent itemset is Apriori algorithm [13]. After the Apriori algorithm has been proposed, many related algorithms have been suggested [14-16]. Specifically, FP-growth is known as one of the fastest algorithm for mining frequent itemset. After the appearance of SSDs, some researchers are interested in the internal structure of SSDs [1, 11]. [1] has surveyed the known internal structures of SSDs and suggested basic policies to enhance the performance of SSDs. Other study suspected the internal structure of SSDs by analyzing the result of benchmark executed on SSDs [11]. This study also discussed the factors that degrade the performance of SSDs. Tracing kernel operation has been used by several studies. [17] designed the kernel level tracing tools for analyzing kernel activity. [18] implemented the system call monitoring tools for detecting write operations.

The fields of applying data mining in system are various. The well-known studies are [19, 20], adopting data mining to detecting abnormal behavior on networks. Several studies have been focused on applying data mining to storage device fields. [21] suggested a concept called pre-fetching. Pre-fetching reduces the latency of storage device and predicates the incoming read request and reads the data before the actual demands. Recently, [22] proposed the more sophisticated pre-fetching method. The pre-fetching, however, only enhance the read performance of the storage device.

3. The Overall Process of PRMS

The overall process of PRMS is twofold. The first process is finding the frequent page sets. PRMS generates write traces, the set of write operations, while the workloads are executed. After the generation of write traces, PRMS finds frequent page sets from write traces by applying frequent itemset mining. Like other frequent itemset issues, PRMS examines itemsets and transactions from input data and find frequent page sets. Different from other frequent itemset issues, PRMS handles the overlapping of certain items over several frequent page sets. The second process is the reallocation of frequent page sets. In this paper, PRMS reallocates frequent page set by using Parallel ATA (PATA) commands, the standard interface for the connection of storage. As PATA commands directly handle the logical sector of storage devices, PRMS can reallocate the pages corresponding to the logical sector. Reallocating algorithm of PRMS considers the size of logical block in SSDs and the I/O efficiency.

3.1 Generation of Write Traces

PRMS maintains information of requested write operations on main memory and periodically flushes the information to storage devices. <Table 1> shows an example of write traces. There are two ways to tracing write operations. One is recognizing the write operation immediately when the workloads deploy it. The system called Amino implemented this method by utilizing a monitoring process in Linux systems [18]. When a system call is invoked, Amino catches a system call and check out this system call incurring write operations. This method can directly fetch the write operations. The other method is recognizing write operations when data are written to storage devices (i.e. when the disk cache is flushed into storage devices). Since generated write traces from the second method are strongly related to real write patterns, we adopted the second method in catching write operations.

Tracing write operations can be additional overhead to upper layers of SSDs. Main memory space has to be spared for maintaining information of requested I/O operations. Flushing the write traces to storage devices causes another write cost. From the results of our observation, the memory space and additional write cost revealed to be manageable. For 20,000 write operations

〈Table 1〉 An example of write traces

Time point	Sector offset	# of sectors to write
PM 8:49:13.921	100	8
PM 8:49:13.921	56	4
PM 8:49:13.921	110	16
PM 8:49:13.921	35	8
PM 8:49:14.687	100	8
PM 8:49:14.687	35	8
PM 8:49:14.734	104	16
PM 8:49:14.797	56	4
PM 8:49:14.797	110	16

which are generated from TPC-C benchmarks, the size of the write traces is approximately 8 MB. Writing 8MB data to SSDs used in our experiments took 0.015 seconds. This additional write cost is relatively small, compared to 635 seconds, which was taken for 20,000 write operations to be processed on SSDs. Therefore, the cost of generating write traces will be tolerable if the upper layers of SSDs save the main memory space for maintaining write traces and flush the memory space to SSDs.

3.2 Extraction of Frequent Page Sets from Write Traces

Finding frequent page sets is identical to frequent itemset mining. Each write operation of write traces can be treated as an item. The information such as sector offset and number of sectors to be written distinguishes each write operation from one another. We define the range deduced from sector offset and number of sectors to be written as **write range**. <Table 2> shows extracted items from write traces of <Table 1>.

Unlike frequent itemset mining issues, finding frequent page sets has the overlapping problem. Specifically, write operations have different write ranges, due to different offset and different amount of data to write. Each write range can overlaps one another. In <Table 2>, write range of item *I1* is sector 100~107 and write range of item *I5* is sector 104~119. Thus, two write ranges overlaps within sector 104~107. In frequent itemset mining issue, each item does not overlap to others. Therefore, the way to cope with the overlapped write range problem should be considered. One intuitive approach is decomposing write operation into items each of which has only one sector per item. From upper example, 8 items can be identified from *I1*. This approach, 〈Table 2〉 Extracted items from write traces shown in Table 1

Time point	Sector offset	# of sectors to write	Item Id
PM 8:49:13.921	100	8	I1
PM 8:49:13.921	56	4	I2
PM 8:49:13.921	110	16	I3
PM 8:49:13.921	35	8	I4
PM 8:49:14.687	100	8	I1
PM 8:49:14.687	35	8	I4
PM 8:49:14.734	104	16	I5
PM 8:49:14.797	56	4	I2
PM 8:49:14.797	110	16	I3

however, generates a substantial number of items. In this paper, we strictly regard each write operation as an item even if other items nearly overlap the item. The policy, one item per write operation, alleviates the difficulty of mining process. Nevertheless, the overlapping problem cannot be completely resolved. We address the remaining problem in reallocating phase. By comparing time points of write operations with one another, a set of write operations can be converted to a transaction. As mentioned in section 3.1, PRMS recognizes write operations when data are written to storage devices. If a set of write operations has the same time point, these write operations were written to storage devices at once. In this paper, we define a set of write operations whose time points are the same as a transaction.

To define the appropriate minimum support count, PRMS observes the cumulative percentage of support count of certain write trace. In detail, PRMS groups the page sets by its support counts. PRMS considers the lowest support count of groups within top *n* % page sets (decreasing order by its support count) as the minimum support count. For example, if we choose the parameter *n* to 10 %, the minimum support count in <Table 3> will be 4. If the parameter n is 15 %, then the minimum support count in <Table 3> will be 3 and so on.

The other concepts applied to PRMS are follows. PRMS finds the maximal frequent itemsets. If a frequent page set denoted as *S1* is a subset of other frequent page set denoted as *S2*, there is no need to reallocate both *S1* and *S2*. The frequent page set *S1* can be eliminated from the mining result. For its popularity and efficiency, we adopt FP-growth to our frequent itemset mining process [14].

〈Table 3〉 Cumulative percentage of support count of write trace

	Cumulative count of page sets	Cumulative percentage (%)
Support count ≥ 6	1,054	1.922
Support count ≥ 5	2,021	3.685
Support count ≥ 4	3,593	6.552
Support count ≥ 3	7,622	13.899
Support count ≥ 2	54,838	100
Total	54,838	100

〈Table 4〉 Identified transactions and maximal frequent itemsets from write traces shown in 〈Table 1〉

Transaction ID	Itemset
T1	I1,I2,I3,I4
T2	I1,I4
T3	I5
T4	I2,I3

Frequent itemset	Itemset	Support count
S1	{I1,I4}	2
S2	{I2,I3}	2

<Table 4> shows the identified transaction and maximal frequent itemsets where the minimum support count is 2.

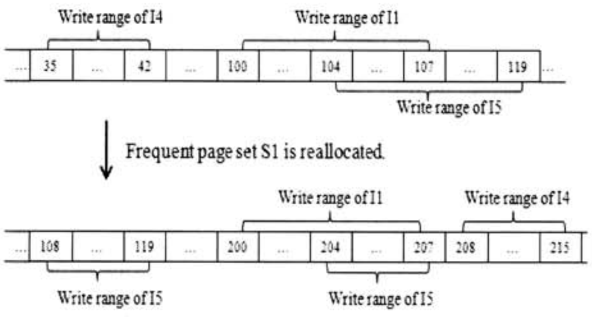
3.3 Reallocation of Frequent Page Sets to Logical Blocks

3.3.1 Expansion Process

Reallocation without considering overlapped write range causes additional write operations. An update operation depends on data to be written. Therefore, reallocation of data changes the write ranges of the related update operations. Assume that a certain part of write range of an update operation is reallocated. After the reallocation, the incoming update operations should update both the reallocated write range and un-reallocated write range. As two updates cannot be handled in one write operation, such a case generates additional write operations. (Fig. 1) is an example of additional write operations when frequent page set *S1* is reallocated to sector 200~215(sector 200~207 for *I1* and sector 208~215 for *I4*). If the write operation *I5* is an update operation, the incoming write operation *I5* should write data to sector 204~207 and 108~119.

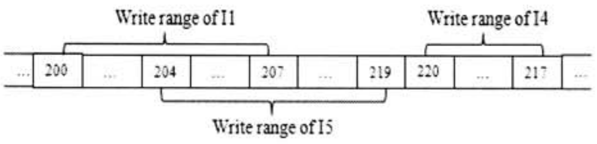
PRMS handles the overlapping problem by expanding

(Fig. 1) Additional write operations by the reallocation. Sector 100~107 is moved to sector 200~207 and sector 35~42 is moved to sector 208~215



the write ranges of overlapped frequent page sets. In brief, if PRMS reallocates write range A overlapping write range B, PRMS expands write range A to the union of A and B ($A \cup B$). (Fig. 2) is an example of expansion process. Since *I1* and *I5* overlaps each other, PRMS expands write range of *I1* to sector 100~119, the write range of the union of *I1* and *I5*. The reallocation of sector 100~119 prevents the additional write operation of *I5*.

(Fig. 2) The reallocation of expanded frequent page set. Write range of I1 is expanded to sector 100~119. For the reallocation, sector 100~119 is moved to 200~219 and sector 35~42 is moved to sector 22 0~217



[Algorithm 1] Expansion algorithm

```
Expansion algorithm
Input :   write traces $WT$, frequent page sets $F$
Output:  expanded frequent page sets $EF$
Expansion algorithm($WT$,$F$)
  1:  for each frequent page set $S$ of $F$ do
  2:    for each write range $w$ of $S$ do
  3:      Set $ws \leftarrow$ start offset of $w$
  4.      Set $we \leftarrow$ end offset of $w$
  5:      for each write range $t$ of $WT$ do
  6:        if $t$ overlaps with $w$ then
  7:          Set $ws \leftarrow$ start offset of $w \cup t$
  8:          Set $we \leftarrow$ end offset of $w \cup t$
  9:    Set start offset of $w \leftarrow ws$ and end offset of $w$
        $\leftarrow we$
 10:  for each frequent page set $S$ of $F$ do
 11:    for each write range $p$ of $S$ do
 12:        for each frequent page set $R$ of $F$ do
 13:          for each write range $q$ of $R$ do
 14:            if $p$ overlaps with $q$ then
 15:              set $p \leftarrow p \cup q$ and $q \leftarrow p \cup q$
 16:  $EF \leftarrow F$ and return $EF$
```

If several write ranges *R1*, *R2*, *R3*, ... ,*Rn* are overlapped with a write range R to be reallocated, PRMS expands the write range R to union of all the overlapped write ranges and R ($R \cup R1 \cup R2 \cup R3 \cdots \cup Rn$). Lines 1 to 9 in algorithm 1 describe the expansion process. For each write range of all frequent page sets, PRMS locates overlapped write ranges (line 6) and calculates the write range of the frequent page set to be expanded (line 7 and 8). After finding all the overlapped write ranges, PRMS finally set the write range to the calculated write range (line 9). This scheme is not capable of eliminating all the expected additional writes. There are overlapped write operations such that none of them are members of frequent page sets. To remove all the expected additional writes, all the write operations of write traces have to be sorted and the overlapped write ranges have to be located. This process is not applied to PRMS since high time complexity is expected. There are still chances of additional write costs since it is possible for each expanded frequent page to overlap others. <Table 5> presents write ranges of expanded frequent page sets shown in <Table 4>. The expanded write range of *I1* in *S1* overlaps expanded write range of *I3* in *S2*. Therefore, reallocation of *S1* causes the fragmentation of *S2*. To prevent the additional write operation, PRMS expands the write range of the frequent page again (line 10-15 in algorithm 1). Differently from former expansion, PRMS compares the write range of the frequent set to the write range of other frequent page set.

The time complexity of expansion algorithm can be induced as follows. In line 1 to 9, expansion algorithm scans a write trace for each frequent page set. If we

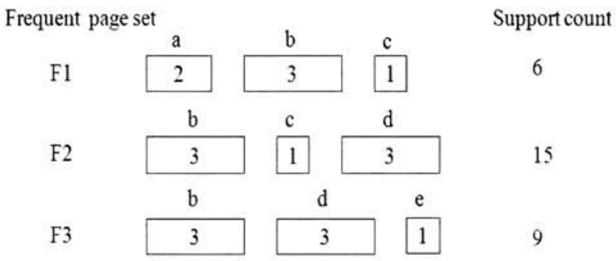
<Table 5> Write range of each item of expanded frequent page sets S1 and S2

Frequent page set	Write operation	Original write range	Overlapped Write operations	Expanded write range
S1	I1	100~107	I5	100~119
	I4	35~42	None	35~42
S2	I2	56~59	None	56~59
	I3	110~125	I5	104~125

denotes the number of write operation of write trace as M and the average number of write range in a frequent page set as C and the number of frequent page sets as N, the time complexity of line 1 to 9 is $O(CMN)$. In line 10 to 15, algorithm 1 scans whole frequent page sets at each frequent page set. With the same notations, the time complexity of line 10 to 15 can be described as $O(C^2N^2)$. Thus, the whole time complexity of expansion algorithm is $O(CMN + C^2N^2)$.

3.3.2 Reallocation Process

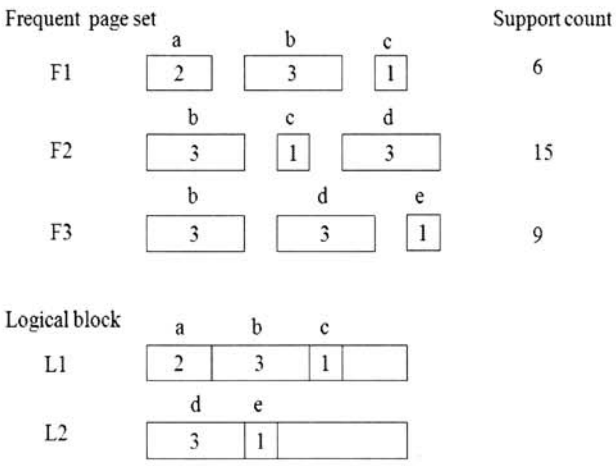
The order of reallocating frequent page sets determines the efficiency of reallocation process. To verify the efficiency of each reallocation process, we compute the total number of logical block accesses before the reallocation. (Fig. 3) is an example of frequent page sets. Since each write range is separated to others, the number of logical block accesses per frequent page set is the same as the number of write operations in frequent page set. When each frequent page set has been requested with the same support count, the total number of logical block accesses is 90(=6·3+15·3+9·3).



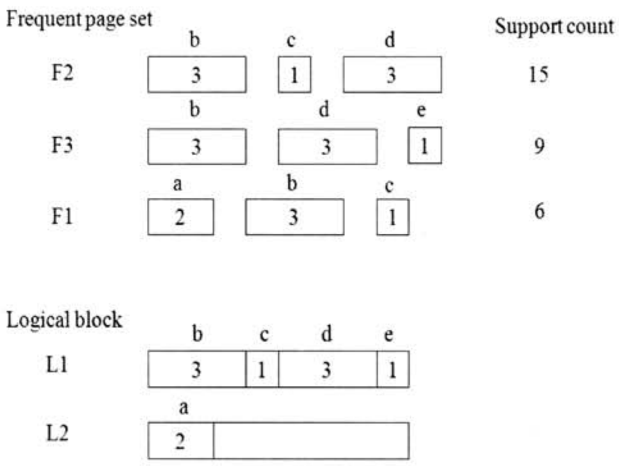
(Fig. 3) An example of frequent page sets.

(Fig. 4) is an intuitive reallocation process of frequent page sets. In (Fig. 4), *F1*, *F2* and *F3* are reallocated sequentially. The reallocation of *F1*, *F2* and *F3* is processed as follows. First, *F1*(write range *a*, *b* and *c*) is reallocated to logical block *L1*. Second, because write ranges *b* and *c* are already reallocated to *L1*, write range *d* of *F2* is reallocated. Write range *d* of *F2* is reallocated to the logical block *L2* since there is not enough space for *d* in *L1*. Third, during reallocation of *F3*, write range *e* of *F3* is reallocated since write range *b* and *d* are already reallocated in *L1* and *L2*. There are enough space for write range *e* in both *L1* and *L2*. We randomly choose *L2* as a logical block in order to reallocate write range *e*. After the intuitive reallocation, logical block accesses of frequent page set are reduced. For example, we need only one logical block access to handle the frequent page set *F1* after the intuitive reallocation. If each frequent page set has been requested with the same support counts, the total number of logical block accesses is 54(=6+15·2+9·2). Compared to 90(i.e. the total number of logical block accesses before the intuitive reallocation) intuitive reallocation definitely decreases the number of logical block accesses. However, this intuitive reallocation scheme ignores the priority of frequent pages.

PRMS considers the priority of each frequent page set. PRMS reallocates first frequent page set with higher support count. In more detail, PRMS sorts the frequent page sets in decreasing order of support count and reallocates sorted frequent page set sequentially. The way of reallocating each frequent page set is identical to intuitive reallocation. (Fig. 5) is an example of



(Fig. 4) Intuitive reallocation process of frequent page sets. The logical block size is 8 sectors



(Fig. 5) Reallocation process of frequent page sets in decreasing order of support counts

[Algorithm 2] Reallocation algorithm

```
Reallocation algorithm
Input : Expanded frequent page set $EF$, logical block set $LBS$
Reallocation algorithm($EF$,$LBS$)
1: sort each frequent page set in decreasing order of support counts
2: for each frequent page set $P$ of $EF$ do
3:   for each write range $w$ of $P$ do
4:     if $w$ is not reallocated then
5:       for each logical block $LB$ of $LBS$ do
6:       if remained space of $LB$ > data size of $w$ then
7:           move data of $w$ to $LB$ and break the iteration 5~7
```

reallocating frequent page sets in decreasing order. After this reallocation, the total number of logical block accesses is 36(=6·2+ 15 + 9). The reallocation algorithm is described below.

Similar to expansion algorithm, the time complexity of reallocation algorithm can be defined as follows. Denoting the average number of write range in a expanded frequent page set as C, the number of frequent page sets as N, the number of logical block as L and the average time for moving data of a write range to logical block as T, the time complexity of reallocation algorithm is $O(N(logN + C(L+T)))$.

4. Experiments

Our experiments were based on common environment, equipped with 2.3 GHz CPU and 2GB RAM. As a storage device, we used a commercial SSDs, MTRON MSP-SATA7035. The maximal performance of the used SSDs is 120MB/sec for read and 90MB/sec for write. Logical block size of SSDs used in our experiment is 1MB. We used 16GB space for the benchmark test. The operating system was Windows XP and DBMS was MySql 5.0.

For the benchmark test, we used a specified toolkit [23]. To reduce the interference, we modified TPC-C parameters. TPC-C defines the behavior of each emulated users, not the whole group of emulated users [24]. Therefore, not enough quantity of emulated users may randomize the write patterns. For regulation of the side effects of several emulated users and the limits of our experiment environments, we set the number of emulated user to 1. TPC-C parameters such as key time and think time were set to 5 seconds each. We run several benchmark tests and choose 4 tests for mining and reallocating process.

〈Table 6〉 TPC-C benchmark tests

TPC-C test	Executed transaction	Logical block accesses	Executed time(sec)
T1	10,000	37,411	2,233
T2	10,000	36,824	2,196
T3	20,000	73,970	4,631
T4	20,000	74,656	4,272

As a preliminary process, PRMS calculated the appropriate minimum support count. We consider the support count of top 5 % highly-accessed page sets as the minimum support count. By observing the cumulative percentage of support count of certain write trace (i.e. test $T1$), we choose 5 as the minimum support count.

Since the real benefit of reallocation is hard to estimate, we planned a simulation for the estimation. The related work revealed the performance degradation of SSDs after the initial data distribution of SSDs is randomized by random writes [11]. During our experiments, we observed the performance of write operation was decreased after the several benchmark tests were executed. Erasing all blocks of SSDs and rebuilding the data may recover the write performance of SSDs, however, such process changes the whole data distribution of SSDs. From this reason, we did not actually reallocate frequent page sets to SSDs. We virtually reallocated frequent page sets to main memory spaces. We assumed a benchmark test was occurred after the reallocation and counted the logical block access of benchmark tests.

<Table 7> shows the measured mining time and reallocation time of the experiments. <Table 7> also lists the factors to determine the reallocation time. In our experiments, the measured reallocation time matches the time complexity of reallocation process. For example, the logical block access (denoted as M in section 3.3) and the total number of write range of frequent page sets (denoted as $M{\cdot}C$ in section 3.3) of $T3$ are twice than that of $T1$. According to the time complexity, the reallocation time of $T3$ will be four times more than reallocation time of $T1$. Shown in <Table 7>, the estimated reallocation times matched the assumption. The main time-consumed task was mining process. The

〈Table 7〉 The measured mining time and reallocation time

TPC-C test	Logical block accesses	Mining time	Reallocation time	Total time
T1	37,411	6.687	1.595	8.282
T2	36,824	7.798	1.377	9.175
T3	73,970	9.556	5.993	15.549
T4	74,656	10.941	4.707	15.648

TPC-C test	Logical block accesses	Number of frequent page sets	Average number of write range in a frequent page set
T1	37,411	949	8.173
T2	36,824	996	7.700
T3	73,970	2,989	5.156
T4	74,656	3,042	5.003

〈Table 8〉 Simulated result of the reallocation. The reduced percentage is the ratio of reduced logical block accesses to the original logical block accesses of the benchmark

Test for mining	Test for simulation	Logical block accesses after reallocation	Reduced percentage (%)	logical blocks for reallocation
T1	T2	34,362	6.69	5
T2	T3	70,317	4.94	5
T3	T4	69,501	6.91	15
T4	T1	34,972	6.52	15

mining time was approximately proportional to the size of write traces.

In our experiments, 6.2% of logical block accesses were reduced after the reallocation of frequent page sets. The reduced access of logical block means that more pages can be written in one logical block. Such an effect saves the time for searching the logical blocks and ease the update mechanism in block based SSDs.

Though the percentage of reduced logical block accesses seems small, there are three contributions. First, PRMS enhanced the write performance without modifying any internal structure of SSDs. Applying PRMS would increase the performance of system regardless of any SSDs. Second, the benefit of reallocation is sustained in period of time. Each four tests were selected randomly among several benchmark tests. This selection regulates the temporal data correlation of each reallocation process. From this point of view, PRMS can be executed in cycle and soften the cost of PRMS. Third, PRMS only needs small number of logical blocks for reallocation. All four reallocation simulations use 5MB or 15 MB memory space for the reallocation. The the simulation denoted on third row, test *T3* in <Table 6>, writes approximately 1280MB on SSDs. Since average request data size per one logical block accesses in our experiments is 16 KB and reduced percentage is 4.94%, total size of 58MB write operations were serialized with other write operations. And such effect was taken by using only 5MB for reallocation.

5. Conclusions and Future Work

We proposed a novel way of improving the performance of SSDs. In this paper, we suggested that the migration of frequent data could reduce the overall performance degradation of SSDs which is mainly caused by frequent random writes. Due to the fact that internal structure of SSDs is completely different from that of hard disks, we designed a basic algorithm for the reallocation process. Our experiment results confirmed that classic frequent mining algorithm, FP-growth, could be applied to mining frequent page sets with reasonable time complexity. In simulation experiment, PRMS reduced the number of logical block accesses by approximately 6.2%.

Several unsolved problems exist in our research. First, PRMS was experimented with synthetic benchmarks. We have plans to apply our method to several real workloads. In addition, we are eager to analyze the appropriate execution cycle and performance evaluation of PRMS in real workloads. Second, we used common monitoring tools for generating write traces. Implementing PRMS on open source OS or building own monitoring module is possible as a direction of our future research. Third, the mining algorithm is imperfect for mining frequent page sets due to the overlapping problem. Besides, the recycle of former frequent page set in next frequent page set mining was not considered. We are willing to design an appropriate algorithm for fining frequent page sets in SSDs. Forth, we are planned to apply PRMS in SSDs based on the page mapping FTL. As our method reallocates frequent page set into one logical block, PRMS does not enhance the write performance of SSDs based on the page mapping FTL.

References

[1] Nitin Agrawal, et al., "Design tradeoffs for SSDs performance", USENIX 2008, p.57-70, 2008.

[2] www.intel.com/design/flash/nand/extreme/index.htm

[3] 정승국, 고대식, "차세대 스토리지 SSD 기술 동향", 정보통신연구진흥원 report, 2008.

[4] 장성원, "차세대 저장장치 SSD의 부상과 시사점", SERI report, pp.1-15, 2008.

[5] S. W. Lee, et al., "A log buffer-based flash translation layer using fully-associative sector translation", ACM Transactions on Embedded Computing Systems (TECS), 6(3), pp. 18-es, July 2007.

[6] Kim, J., Kim, J. M., Noh, S. H., Min, S. L., and Cho, Y, "A space-efficient flash translation layer for compact-flash systems", IEEE Transactions on Consumer Electronics, 48(2), 2002.

[7] 나갑주, 이상원 "플래쉬 메모리 기반의 B+트리 알고리즘", 한국인터넷정보학회 춘계학술발표대회 논문집, 제 7권, 제 1호, pp.167-172, 2006.

[8] 김성탄, 김상우, 이상원, "Flash SSD 상에서 인덱스 기반 질의 처리", 한국컴퓨터종합학술대회 논문집, 제 1권, 제 35호, pp. 33-34, 2008.

[9] L. Chang. "On efficient wear leveling for large-scale flash-memory storage systems", SAC'07, pp.1126-1130, 2007.

[10] K. M. J. Lofgren, R. D. Norman, G B. Thelin, and A. Gupta, "Wear Leveling Techniques for Flash EEPROM", In United States Patent, No 6,850,443, 2005.

[11] Feng Chen, et al., "Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives", ACM SIGMETRICS, 2009.

[12] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases", Proceedings of the ACM SIGMOD Int'l Conferenceon Management of Data, 1993.

[13] Agrawal R, Srikant R. "Fast Algorithms for Mining Association Rules", VLDB, pp.487-99. 1994.

[14] J. Han, H. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation", In Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX), ACM Press, New York, NY, USA 2000.

[15] Pei, J., Han, J., and Mao, R., "CLOSET: An efficient algorithm for mining frequent closed itemsets", In Proc. ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX, pp.11-20, 2000.

[16] D. Burdick, M. Calimlim, and J. Gehrke, "MAFIA: a maximal frequent itemset algorithm for transactional databases", In Intl. Conf. on Data Engineering, April 2001.

[17] C. LaRosa and et al. "Frequent pattern mining for kernel trace data", In Proc. of ACM SAC'08, 2008.

[18] Charles Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok, "Amino: Extending ACID Semantics to the File System." In FAST '05 Conference on File and Storage Technologies, December 2005.

[19] Clifton. C, and Gengo. G., "Developing custom intrusion detection filters using data mining", In Proceedings of the 2000 Military Communications International Symposium, 2000.

[20] Lane, T. and Brodley, C. E. "Sequence matching and learning in anomaly detection for computer security", In AAAI Workshop: AI approaches to Fraud Detection and Risk Management, pp.43-49, 1997.

[21] Griffioen, J. and Appleton, R. "Reducing file system latency using a predictive approach", In Proceedings of the USENIX Summer 1994 TechnicFal Conference, pp.197-207, 1994.

[22] Li, Z., Chen, Z., and Zhou, Y. "Mining Block Correlations to Improve Storage Performance", ACM Transactions on Storage 1, 2, pp. 213-245, 2005.

[23] www. hammerora.sourceforge.net/

[24] www.tpc.org/tpcc/

이 동 현

e-mail : ldh@cs.yonsei.ac.kr

2008년 2월 연세대학교 컴퓨터과학과 (공학사)

2010년 2월 연세대학교 컴퓨터과학과 (공학석사)

관심 분야 : 데이터 마이닝, SSD

노 홍 찬

e-mail : fallsmal@cs.yonsei.ac.kr

2006년 2월 연세대학교 컴퓨터과학과 (공학사)

2008년 2월 연세대학교 컴퓨터과학과 (공학석사)

2008년 3월~현 재 연세대학교 컴퓨터과학과 박사과정

관심 분야 : 플래쉬메모리 인덱스, SSD, 데이터 마이닝

박 상 현

e-mail : sanghyun@cs.yonsei.ac.kr
1989년 2월 서울대학교 컴퓨터공학과
(공학사)
1991년 2월 서울대학교 컴퓨터공학과
(공학석사)
2001년 2월 UCLA 대학교 전산학과
(공학박사)
2001년 2월~2002년 6월 IBM T. J Watson Research Center
Post-Doctoral Fellow.
2002월 8월~2003년 8월 포항공과대학교 컴퓨터공학과 조교수
2003년 9월~2006년 8월 연세대학교 컴퓨터과학과 조교수
2006년 9월~현　재 연세대학교 컴퓨터과학과 부교수
관심분야 : 데이터베이스 보안, 데이터 마이닝, 바이오인포매틱스,
XML