A Case for Flash Memory SSD in Enterprise Database Applications

Sang-Won Lee[†] Bongki Moon[‡] Chanik Park[§] Jae-Myung Kim[¶] Sang-Woo Kim[†]

[†]School of Information & Communications Engr.
Sungkyunkwan University
Suwon 440-746, Korea
{wonlee,swkim}@ece.skku.ac.kr

[‡]Department of Computer Science
University of Arizona
Tucson, AZ 85721, U.S.A.
bkmoon@cs.arizona.edu

[§]Samsung Electronics Co., Ltd.
San #16 Banwol-Ri
Hwasung-City 445-701, Korea
ci.park@samsung.com

[¶]Altibase Corp.
182-13, Guro-dong, Guro-Gu
Seoul, 152-790, Korea
jmkim@altibase.com

ABSTRACT

Due to its superiority such as low access latency, low energy consumption, light weight, and shock resistance, the success of flash memory as a storage alternative for mobile computing devices has been steadily expanded into personal computer and enterprise server markets with ever increasing capacity of its storage. However, since flash memory exhibits poor performance for small-to-moderate sized writes requested in a random order, existing database systems may not be able to take full advantage of flash memory without elaborate flash-aware data structures and algorithms. The objective of this work is to understand the applicability and potential impact that flash memory SSD (Solid State Drive) has for certain type of storage spaces of a database server where sequential writes and random reads are prevalent. We show empirically that up to more than an order of magnitude improvement can be achieved in transaction processing by replacing magnetic disk with flash memory SSD for transaction log, rollback segments, and temporary table spaces.

Categories and Subject Descriptors

H. Information Systems [**H.2 DATABASE MANAGEMENT**]: H.2.2 Physical Design

General Terms

Design, Algorithms, Performance, Reliability

*This work was partly supported by the IT R&D program of MIC/IITA [2006-S-040-01] and MIC, Korea under ITRC IITA-2008-(C1090-0801-0046). The authors assume all responsibility for the contents of the paper.

Keywords

Flash-Memory Database Server, Flash-Memory SSD

1. INTRODUCTION

Due to its superiority such as low access latency, low energy consumption, light weight, and shock resistance, the success of flash memory as a storage alternative for mobile computing devices has been steadily expanded into personal computer and enterprise server markets with ever increasing capacity of its storage. As it has been witnessed in the past several years, two-fold annual increase in the density of NAND flash memory is expected to continue until year 2012 [11]. Flash-based storage devices are now considered to have tremendous potential as a new storage medium that can replace magnetic disk and achieve much higher performance for enterprise database servers [10].

The trend in market is also very clear. Computer hardware manufacturers have already launched new lines of mobile personal computers that did away with disk drives altogether, replacing them with flash memory SSD (Solid State Drive). Storage system vendors have started lining up their flash-based solutions in Terabyte-scale targeting large-scale database servers as one of the main applications.

Adoption of a new technology, however, is often deterred by lack of in-depth analysis on its applicability and cost-effectiveness, and is even considered risky when it comes to mission critical applications. The objective of this work is to evaluate flash memory SSD as stable storage for database workloads and identify the areas where flash memory SSD can be best utilized, thereby accelerating its adoption as an alternative to magnetic disk and maximizing the benefit from this new technology.

Most of the contemporary database systems are configured to have separate storage spaces for database tables and indexes, log data and temporary data. Whenever a transaction updates a data object, its log record is created and stored in stable storage for recoverability and durability of the transaction execution. Temporary table space stores temporary data required for performing operations such as sorts or joins. If multiversion read consistency is supported, another separate storage area called rollback segments is created to store previous versions of data objects.

For the purpose of performance tuning as well as recoverability, these distinct storage spaces are often created on physically separate storage devices, so that I/O throughput can increase, and I/O bottlenecks can be detected and addressed with more ease. While it is commonly known that accessing data stored in secondary storage is the main source of bottlenecks in database processing, high throughput of a database system cannot be achieved by addressing the bottlenecks only in spaces for tables and indexes but also in spaces for log, temporary and rollback data.

Recent studies on database availability and architecture report that writing log records to stable storage is almost guaranteed to be a significant performance bottleneck [13, 21]. In on-line transaction processing (OLTP) applications, for example, when a transaction commits, all the log records created by the transaction have to be force-written to stable storage. If a large number of concurrent transactions commit at a rapid rate, the log tail will be requested to be flushed to disk very often. This will then lengthen the average wait time of committing transactions and delay the release of locks further, and eventually increase the overall runtime overhead substantially.

Accessing data stored in temporary table spaces and rollback segments also takes up a significant portion of total I/O activities. For example, queries performing a table scan, join, sort or hash operation are very common in a data warehousing application, and processing those queries (except simple table scans) will require a potentially large amount of intermediate data to be written to and read from temporary table spaces. Thus, to maximize the throughput of a database system, it is critical to speed up accessing data stored in those areas as well as in the data space for tables and indexes.

Previous work has reported that flash memory exhibits poor performance for small-to-moderate sized writes requested in a random order [2] and the best attainable performance may not be obtained from database servers without elaborate flash-aware data structures and algorithms [14]. In this paper, in contrast, we demonstrate that flash memory SSD can help improve the performance of transaction processing significantly, particularly as a storage alternative for transaction log, rollback segments and temporary table spaces. To accomplish this, we trace quite distinct data access patterns observed from these three different types of data spaces, and analyze how magnetic disk and flash memory SSD devices handle such I/O requests, and show how the overall performance of transaction processing is affected by them.

While the previous work on *in-page logging* is targeted at regular table spaces for database tables and indexes where small random writes are dominant [14], the objective of this work is to understand the applicability and potential impact that flash memory SSD has for the other data spaces where sequential writes and random reads are prevalent. The key contributions of this work are summarized as follows.

- Based on a detailed analysis of data accesses that are traced from a commercial database server, this paper provides an understanding of I/O behaviors that are dominant in transaction log, rollback segments, and temporary table spaces. It also shows that this I/O pattern is a good match for the dual-channel, super-block design of flash memory SSD as well as the characteristics of flash memory itself.

- This paper presents a quantitative and comparative analysis of magnetic disk and flash memory SSD with respect to performance impacts they have on transactional database workloads. We observed more than an order of magnitude improvement in transaction throughput and response time by replacing magnetic disk with flash memory SSD as storage media for transaction log or rollback segments. In addition, more than a factor of two improvement in response time was observed in processing a sort-merge or hash join query by adopting flash memory SSD instead of magnetic disk for temporary table spaces.

- The empirical study carried out in this paper demonstrates that low latency of flash memory SSD can alleviate drastically the log bottleneck at commit time and the problem of increased random reads for multiversion read consistency. With flash memory SSD, I/O processing speed may no longer be as serious a bottleneck as it used be, and the overall performance of query processing can be much less sensitive to tuning parameters such as the unit size of physical I/O. The superior performance of flash memory SSD demonstrated in this work will help accelerate adoption of flash memory SSD for database applications in the enterprise market, and help us revisit requirements of database design and tuning guidelines for database servers.

The rest of this paper is organized as follows. Section 2 presents a few key features and architecture of Samsung flash memory SSD, and discusses its performance characteristics with respect to transactional database workloads. Section 3 describes the experimental settings that will be used in the following sections. In Section 4, we analyze the performance gain that can be obtained by adopting flash memory SSD as stable storage for transaction log. Section 5 analyzes the patterns in which old versions of data objects are written to and read from rollback segments, and shows how flash memory SSD can take advantage of the access patterns to improve access speed for rollback segments and the average response time of transactions. In Section 6, we analyze the I/O patterns of sort-based and hash-based algorithms, and discuss the impact of flash memory SSD on the algorithms. Lastly, Section 7 summarizes the contributions of this paper.

2. DESIGN OF SAMSUNG FLASH SSD

The flash memory SSD (Solid State Drive) of Samsung Electronics is a non-volatile storage device based on NAND-type flash memory, which is being marketed as a replacement of traditional hard disk drives for a wide range of computing platforms. In this section, we first briefly summarize the characteristics of flash memory as a storage medium for databases. We then present the architecture and a few key features of Samsung flash memory SSD, and discuss its performance implications on transactional database workloads.

2.1 Characteristics of Flash Memory

Flash memory is a purely electronic device with no mechanically moving parts like disk arms in a magnetic disk drive. Therefore, flash memory can provide uniform random access speed. Unlike magnetic disks whose seek and rotational delay often becomes the dominant cost of reading or writing a sector, the time to access data in flash memory is almost linearly proportional to the amount of data irrespective of their physical locations in flash memory. The ability of flash memory to quickly perform a sector read or a sector (clean) write located anywhere in flash memory is one of the key characteristics we can take advantage of.

On the other hand, with flash memory, no data item (or a sector containing the data item) can be updated in place just by overwriting it. In order to update an existing data item stored in flash memory, a time-consuming erase operation must be performed before overwriting. The erase operation cannot be performed selectively on a particular data item or sector, but can only be done for an entire block of flash memory called *erase unit* containing the data item, which is much larger (typically 128 KBytes) than a sector. To avoid performance degradation caused by this erase-before-write limitation, some of the data structures and algorithms of existing database systems may well be reconsidered [14].

The read and write speed of flash memory is asymmetric, simply because it takes longer to write (or inject charge into) a cell until reaching a stable status than to read the status from a cell. As will be shown later in this section (Table 1), the sustained speed of read is almost twice faster than that of write. This property of asymmetric speed should also be considered when reviewing existing techniques for database system implementations.

2.2 Architecture and Key Features

High bandwidth is one of the critical requirements for the design of flash memory SSD. The dual-channel architecture, as shown in Figure 1, supports up to 4-way interleaving to hide flash programming latency and to increase bandwidth through parallel read/write operations. An automatic interleaving hardware logic is adopted to maximize the interleaving effect with the minimal firmware intervention [18].

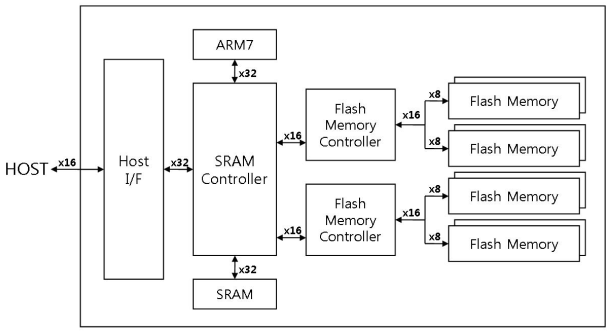

Figure 1: Dual-Channel Architecture of SSD

A firmware layer known as flash translation layer (FTL) [5, 12] is responsible for several essential functions of flash memory SSD such as address mapping and wear leveling. The address mapping scheme is based on super-blocks in order to limit the amount of information required for logical-to-physical address mapping, which grows larger as the capacity of flash memory SSD increases. This super-block scheme also facilitates interleaved accesses of flash memory by striping a super-block of one MBytes across four flash chips. A super-block consists of eight erase units (or large blocks) of 128 KBytes each. Under this super-block scheme, two erase units of a super-block are allocated in the same flash chip.

Though flash memory SSD is a purely electronic device without any moving part, it is not entirely latency free for accessing data. When a read or write request is given from a host system, the I/O command should be interpreted and processed by the SSD controller, referenced logical addresses should be mapped to physical addresses, and if mapping information is altered by a write or merge operation, then the mapping table should be updated in flash memory. With all these overheads added up, the read and write latency observed from the recent SSD products is approximately 0.2 msec and 0.4 msec, respectively.

In order to reduce energy consumption, the one-chip controller uses a small amount of SRAM for program code, data and buffer memory.[1] The flash memory SSD drives can be interfaced with a host system through the IDE standard ATA-5.

2.3 Flash SSD for Database Workload

Typical transactional database workloads like TPC-C exhibit little locality and sequentiality in data accesses, a high percentage of which are synchronous writes (*e.g.*, forced-writes of log records at commit time). Such latency hiding techniques as prefetching and write buffering become less effective for this type of workload, and the performance of transactional database applications tends to be more closely limited by disk latency than disk bandwidth and capacity [24]. Nonetheless, for more than a decade in the past, the latency of disk has improved at a much slower pace than the bandwidth of disk, and the latency-bandwidth imbalance is expected to be even more evident in the future [19].

In this regard, extremely low latency of flash memory SSD lends itself to being a new storage medium that replaces magnetic disk and improves the throughput of transaction processing significantly. Table 1 shows the performance characteristics of some contemporary hard disk and flash memory SSD products. Though the bandwidth of disk is still two to three times higher than that of flash memory SSD, more importantly, the read and write latency of flash memory SSD is smaller than that of disk by more than an order of magnitude.

As is briefly mentioned above, the low latency of flash memory SSD can reduce the average transaction commit time and improve the throughput of transaction processing significantly. If multiversion read consistency is supported, rollback data are typically written to rollback segments sequentially in append-only fashion and read from rollback segments randomly during transaction processing. This pe-

[1]The flash memory SSD drive tested in this paper contains 128 KByte SRAM.

Storage	hard disk[†]	flash SSD[‡]
Average Latency	8.33 ms	0.2 ms (read) 0.4 ms (write)
Sustained Transfer Rate	110 MB/sec	56 MB/sec (read) 32 MB/sec (write)

[†]Disk: Seagate Barracuda 7200.10 ST3250310AS, average latency for seek and rotational delay;
[‡]SSD: Samsung MCAQE32G8APP-0XA drive with K9WAG08U1A 16 Gbits SLC NAND chips

Table 1: Magnetic disk vs. NAND Flash SSD

culiar I/O pattern is a good match for the characteristics of flash memory itself and the super-block scheme of the Samsung flash memory SSD. External sorting is another operation that can benefit from the low latency of flash memory SSD, because the read pattern of external sorting is quite random during the merge phase in particular.

3. EXPERIMENTAL SETTINGS

Before presenting the results from our workload analysis and performance study in the following sections, we describe the experimental settings briefly in this section.

In most cases, we ran a commercial database server (one of the most recent editions of its product line) on two Linux systems (kernel version 2.6.22), each with a 1.86 GHz Intel Pentium dual-core processor and 2 GB RAM. These two computer systems were identical except that one was equipped with a magnetic disk drive and the other with a flash memory SSD drive instead of the disk drive. The disk drive model was Seagate Barracuda 7200.10 ST3250310AS with 250 GB capacity, 7200 rpm and SATA interface. The flash memory SSD model was Samsung Standard Type MCAQE32G8APP-0XA with 32 GB capacity and 1.8 inch PATA interface, which internally deploys Samsung K9WAG08U1A 16 Gbits SLC NAND flash chips (shown in Figure 2). These storage devices were connected to the computer systems via a SATA or PATA interface.

Figure 2: Samsung NAND Flash SSD

When either magnetic disk or flash memory SSD was used as stable storage for transaction log, rollback segments, or temporary table spaces, it was bound as a raw device in order to minimize interference from data caching by the operating system. This is a common way of binding storage devices adopted by most commercial database servers with their own caching scheme. In all the experiments, database tables were cached in memory so that most of IO activities were confined to transaction log, rollback segments and temporary table spaces.

4. TRANSACTION LOG

When a transaction commits, it appends a commit type log record to the log and force-writes the log tail to stable storage up to and including the commit record. Even if a no-force buffer management policy is being used, it is required to force-write all the log records kept in the log tail to ensure the durability of transactions [22].

As the speed of processors becomes faster and the memory capacity increases, the commit time delay due to force-writes increasingly becomes a serious bottleneck to achieving high performance of transaction processing [21]. The response time $T_{response}$ of a transaction can be modeled as a sum of CPU time T_{cpu}, read time T_{read}, write time T_{write} and commit time T_{commit}. T_{cpu} is typically much smaller than IO time. Even T_{read} and T_{write} become almost negligible with a large capacity buffer cache and can be hidden by asynchronous write operations. On the other hand, commit time T_{commit} still remains to be a significant overhead, because every committing transaction has to wait until all of its log records are force-written to log, which in turn cannot be done until forced-write operations requested by other transactions earlier are completed. Therefore, the amount of commit-time delay tends to increase as the number of concurrent transactions increases, and is typically no less than a few milliseconds.

Group commit may be used to alleviate the log bottleneck [4]. Instead of committing each transaction as it finishes, transactions are committed in batches when enough logs are accumulated in the log tail. Though this group commit approach can significantly improve the throughput of transaction processing, it does not improve the response time of individual transactions and does not remove the commit time log bottleneck altogether.

Log records are always appended to the end of log. If a separate storage device is dedicated to transaction log, which is commonly done in practice for performance and recoverability purposes, this sequential pattern of write operations favors not only hard disk but also flash memory SSD. With no seek delay due to sequential accesses, the write latency of disk is reduced to only half a revolution of disk spindle on average, which is equivalent to approximately 4.17 msec for disk drives with 7200 rpm rotational speed.

In the case of flash memory SSD, however, the write latency is much lower at about 0.4 msec, because flash memory SSD has no mechanical latency but only a little overhead from the controller as described in Section 2.3. Even the *no in-place update* limitation of flash memory has no negative impact on the write bandwidth in this case, because log records being written to flash memory sequentially do not cause expensive merge or erase operations as long as clean flash blocks (or erase units) are available. Coupled with the low write latency of flash memory, the use of flash memory SSD as a dedicated storage device for transaction log can reduce the commit time delay considerably.

In the rest of this section, we analyze the performance gain that can be obtained by adopting flash memory SSD as stable storage for transaction log. The empirical results from flash memory SSD drives are compared with those from magnetic disk drives.

4.1 Simple SQL Transactions

To analyze the commit time performance of hard disk and flash memory SSD drives, we first ran a simple embedded SQL program on a commercial database server, which ran on two identical Linux systems except that one was equipped with a magnetic disk drive and the other with a flash memory SSD drive instead of the disk drive. This embedded SQL program is multi-threaded and simulates concurrent transactions. Each thread updates a single record and commits, and repeats this cycle of update and commit continuously. In order to minimize the wait time for database table updates and increase the frequency of commit time forced-writes, the entire table data were cached in memory. Consequently, the runtime of a transaction excluding the commit time (*i.e.*, $T_{cpu} + T_{read} + T_{write}$) was no more than a few dozens of microseconds in the experiment. Table 2 shows the throughput of the embedded SQL program in terms of transactions-per-seconds (TPS).

no. of concurrent transactions	hard disk		flash SSD	
	TPS	%CPU	TPS	%CPU
4	178	2.5	2222	28
8	358	4.5	4050	47
16	711	8.5	6274	77
32	1403	20	5953	84
64	2737	38	5701	84

Table 2: Commit-time performance of an embedded SQL program measured in transactions-in-seconds (TPS) and CPU utilization

Regarding the commit time activities, a transaction can be in one of the three distinct states. Namely, a transaction (1) is still active and has not requested to commit, (2) has already requested to commit but is waiting for other transactions to complete forced-writes of their log records, or (3) has requested to commit and is currently force-writing its own log records to stable storage.

When a hard disk drive was used as stable storage, the average wait time of a transaction was elongated due to the longer latency of disk writes, which resulted in an increased number of transactions that were kept in a state of the second or third category. This is why the transaction throughput and CPU utilization were both low, as shown in the second and third columns of Table 2.

On the other hand, when a flash memory SSD drive was used instead of a hard disk drive, much higher transaction throughput and CPU utilization were observed, as shown in the fourth and fifth columns of Table 2. With a much shorter write latency of flash memory SSD, the average wait time of a transaction was shortened, and a relatively large number of transactions were actively utilizing CPU, which in turn resulted in higher transaction throughput. Note that the CPU utilization was saturated when the number of concurrent transactions was high in the case of flash memory SSD, and no further improvement in transaction throughput was observed when the number of concurrent transactions was increased from 32 to 64, indicating that CPU was a limiting factor rather than I/O.

4.2 TPC-B Benchmark Performance

In order to evaluate the performance of flash memory SSD as a storage medium for transaction log in a more harsh environment, we ran a commercial database server with TPC-B workloads created by a workload generation tool. Although it is obsolete, the TPC-B benchmark was chosen because it is designed to be a stress test on different subsystems of a database server and its transaction commit rate is higher than that of TPC-C benchmark [3]. We used this benchmark to stress-test the log storage part of the commercial database server by executing a large number of small transactions causing significant forced-write activities.

In this benchmark test, the number of concurrent simulated users was set to 20, and the size of database and the size of database buffer cache of the server were set to 450 MBytes and 500 MBytes, respectively. Note that this setting allows the database server to cache the entire database in memory, such that the cost of reading and writing data pages is eliminated and the cost of forced writing log records remains dominant on the critical path in the overall performance. When either a hard disk or flash memory SSD drive was used as stable storage for transaction log, it was bound as a raw device. Log records were force-written to the stable storage in a single or multiple sectors (of 512 bytes) at a time.

Table 3 summarizes the results from the benchmark test measured in terms of transactions-per-seconds (TPS) and CPU utilization as well as the average size of a single log write and the average time taken to process a single log write. Since multiple transactions could commit together as a group (by a group commit mechanism), the frequency of log writes was much lower than the number of transactions processed per second. Again, due to the group commit mechanism, the average size of a single log write was slightly different between the two storage media.

	hard disk	flash SSD
Transactions/sec	864	3045
CPU utilization (%)	20	65
Log write size (sectors)	32	30
Log write time (msec)	8.1	1.3

Table 3: Commit-time performance from TPC-B benchmark (with 20 simulated users)

The overall transaction throughput was improved by a factor of 3.5 by using a flash memory SSD drive instead of a hard disk drive as stable storage for transaction log. Evidently the main factor responsible for this improvement was the considerably lower log write time (1.3 msec on average) of flash memory SSD, compared with about 6 times longer log write time of disk. With a much reduced commit time delay by flash memory SSD, the average response time of a transaction was also reduced considerably. This allowed

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.