HeteroDrive: Reshaping the Storage Access Pattern of OLTP Workload Using SSD[*]

Sang-Hoon Kim[*] Dawoon Jung[†] Jin-Soo Kim[‡] Seungryoul Maeng[§]

Department of Computer Science
KAIST
Daejeon 305-701, South Korea
{sanghoon[*], dwjung[†], maeng[§] }@calab.kaist.ac.kr

School of Info. & Comm. Eng.
Sungkyunkwan University
Suwon 440-746, South Korea
jinsookim@skku.edu[‡]

ABSTRACT

The on-line transaction processing (OLTP) workload is known to produce intense random accesses with high ratio of write. Coping with the random access pattern has been a challenging issue to the underlying storage system. Although SSDs are considered as a breakthrough in storage systems, their random write performance still falls far behind the sequential performance.

In this paper, we propose a hybrid storage architecture, HeteroDrive. HeteroDrive actively reshapes random writes to sequential writes. Random writes are routed to SSD and written sequentially. The large capacity of SSD is used as a cache while the contents are flushed to HDD sequentially. According to our evaluation, HeteroDrive improves the transactions per second (TPS) by up to 201 %.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*Storage hierarchies, secondary storage*

General Terms

Design, Implementation, Evaluation

Keywords

Solid-state drive (SSD), OLTP workload, storage architecture, TPC-C, Oracle, NAND flash memory

1. INTRODUCTION

The on-line transaction processing (OLTP) system is a class of systems that facilitate and manage transaction-oriented applications. Many database operations fall in this class. It is widely known that OLTP systems produce intense random storage accesses with high ratio of write [15]. The conventional hard disk drive (HDD) suffers from the random access pattern and results in high latency in OLTP applications.

Recently emerging NAND flash memory-based solid-state drives (SSD) might be considered as an alternative to HDD in OLTP applications. As SSD stores and retrieves data electronically, it eliminates the seek time which is one of the key contributors of the latency on HDD-based storage systems. It also exhibits many appealing aspects: low energy consumption, high shock resistance, and small and lightweight form factor.

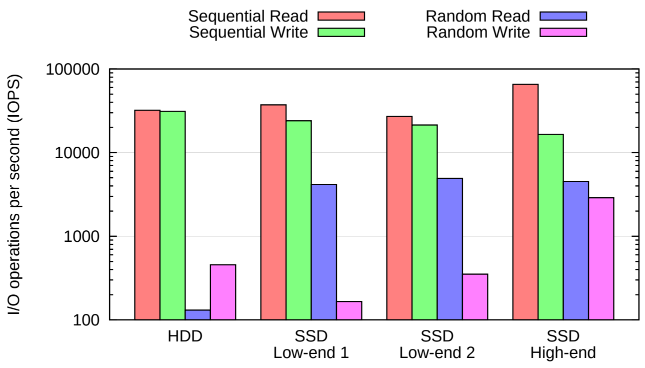


Figure 1: Performance of HDD and various SSDs. The result is obtained with `fio` [14], which issues 4 KB synchronous requests to each device.

In spite of those attractive features, the random write performance of SSD falls far behind expectations. Figure 1 compares the I/O performance of one HDD and three SSDs available in the market. The low-end SSDs perform worse than HDD in random write. Considering the superior random read performance of SSD, which is an order of magnitude higher than that of HDD, the random write performance of the high-end SSD is still disappointing. Also, the price per gigabyte of the high-end SSD is about ×50 higher than that of HDD, and the capacity of SSDs is insufficient for data-intensive OLTP applications. Consequently, simply replacing HDDs with SSDs in the OLTP system promises neither high performance nor cost-effectiveness.

Based on the properties of OLTP applications and SSDs, we designed and implemented a prototype of storage architecture, called *HeteroDrive*. HeteroDrive is tailored to the characteristics of the OLTP workload by combining two heterogeneous storage devices, SSD and HDD. Intense random writes are reshaped to sequential writes to SSD whose sequential write performance is better than the random write performance. With the high random read performance, SSD acts as a cache of HDD. The cache contents are flushed to HDD by sequential write. Figure 2 illustrates the concept of re-

[*]This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (R01-2007-000-11832-0)

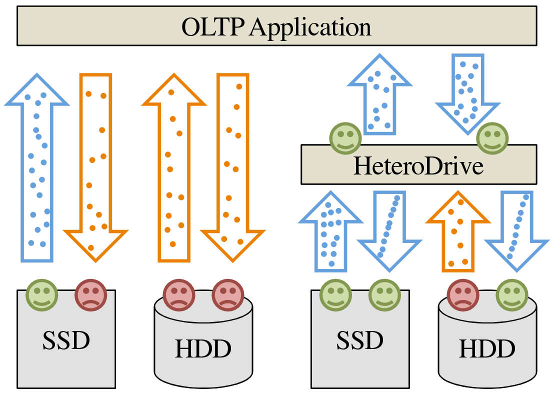


Figure 2: Concept of request reshaping in HeteroDrive

quest reshaping and its benefit. HDD provides an inexpensive large capacity of storage, and offers high bulk write performance with its high sequential write performance. HeteroDrive is viewed as a normal block device to upper layers, hence no application modification is needed. Consequently, HeteroDrive provides a practical opportunity to improve the I/O performance of the storage systems in OLTP applications.

We evaluated HeteroDrive using the Oracle DBMS with the TPC-C benchmark. The evaluation shows that HeteroDrive reshapes requests to be friendly to underlying devices, and improves the I/O performance by up to 201 %.

The rest of This paper is organized as follows. Section 2 goes over the related work. Section 3 presents our motivation and system details. We will evaluate the system in Section 4 and draws a conclusion in Section 5.

2. RELATED WORK

NAND flash memory (or flash) in SSDs has a limitation that a page must be erased before new data is overwritten to the same location. Although many flash translation layers (FTLs) are introduced to hide the erase-before-write limitation, the random write performance has not been improved much.

Adopting a non-volatile memory (NVRAM) in HDDs has been widely discussed [2][5][10][16][17]. They take advantage of the complementary properties of HDD and NVRAM, and try to improve performance or to reduce energy consumption. Hybrid drives [3][4][6][12][13] and Intel Turbo Memory [9] are two representative examples of this approach. However, they employ only a small amount of NVRAM. Our work differs from the previous approaches in that we consider the use of SSD which provides the larger capacity than the NVRAM in Hybrid drives or Intel Turbo Memory.

Narayanan et al. [11] analyze the tradeoffs of SSD in server workload. They build a “solver” which inputs workload and storage specifications, then gives an optimal configuration satisfying the given constraints. They show that replacing HDD with SSD is not beneficial yet due to the high price per gigabyte of SSD. They also suggest a “tiered model” as an intermediate configuration of deploying SSDs. Although it is very similar to our approach, the implementation detail has not been discussed.

Koltsidas et al. [7][8] suggest an architecture using SSD as a page cache and study page replacement policy on the architecture. But the disparity of sequential and random access performance has not been taken into account. Also, it requires the modification of existing applications, which is infeasible for commercial or enterprise environment.

3. DESIGN AND IMPLEMENTATION

3.1 Motivation

To understand the storage characteristics of OLTP workloads, we collected a block-level trace while running TPC-C benchmark for 24 hours. Besides the well-known intense random access pattern, the access count distribution shows that about 20% of touched sectors were read or written more than 10 (up to 400) times. Also, their read count has some correlation with the write count. This implies that the locality might be exploited as randomly written contents are likely to be randomly read in the near future.

As we examined in Section 1, SSD is good at random read while its random write performance is disappointing. The performance of sequential write is an order of magnitude better than that of random write. These properties led us to the idea that reshaping random write into sequential write could utilize SSD better.

Our approach is to place an SSD in front of an HDD (or an array of HDDs) in storage hierarchy, and to co-operate with each other considering the complementary properties of them. Random writes are reshaped into sequential writes and written to SSD sequentially in a log-structured manner. Additionally, the written contents are served as a cache for read requests at high random read performance of SSD. The large capacity of SSD also gives an opportunity to reshape the flush requests, as well. Figure 2 demonstrates the effect of request reshaping done by HeteroDrive.

We design HeteroDrive to be transparent to upper layers so that application modification is not required. As a result, we could run a commercial Oracle DBMS on top of HeteroDrive.

3.2 Architecture Overview

HeteroDrive is implemented as a Linux mapped-device module which accepts block I/O requests and forwards them to the underlying block devices transparently.

Using write requests, HeteroDrive determines whether the current write pattern is sequential or random. If the access pattern is sequential, write requests are routed to HDD. If the pattern is random, write requests are routed to SSD and written sequentially. Read requests are served by mapping the sector of the requests to the corresponding locations. Figure 3 illustrates an example where 8 sectors starting from sector 7970 are mapped to SSD sector 214 (4 sectors), HDD sector 7974 (2 sectors), and SSD sector 408 (2 sectors). Since the mapping information is large, a portion of the mapping information can be swapped out into SSD or loaded into memory on demand.

A background kernel thread, called *flush worker*, performs SSD flushing. The flush worker groups a certain amount of cached contents, and issues write requests to HDD in increasing order of their sector numbers. Such policy reduces the arm movement of HDD and helps to improve the flushing throughput. The flush worker is scheduled when the device is idle, which minimizes the performance impact incurred by flushing operations.

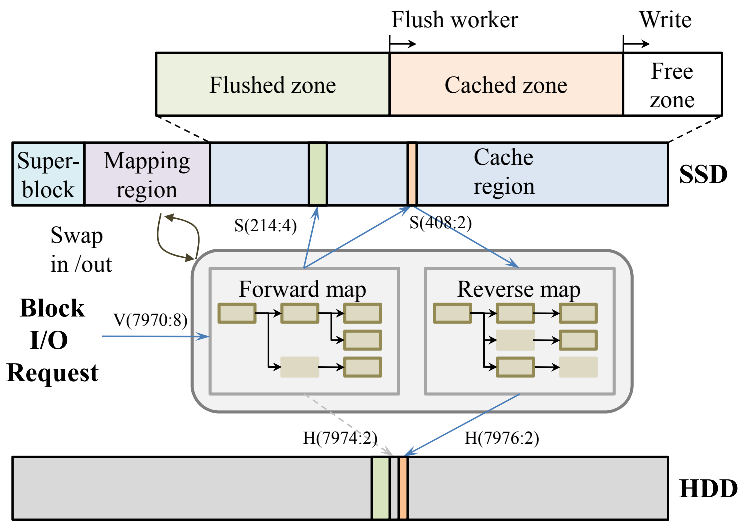


Figure 3: Architecture and example of mapping

3.3 SSD Layout

Figure 3 illustrates the overall layout of SSD under HeteroDrive. SSD is statically partitioned into *superblock region*, *mapping region*, and *cache region*.

The superblock region stores the metadata of HeteroDrive which will be used in mounting and recovery. The mapping region stores the mapping information which is used in translating sector numbers as described in Section 3.5.

The cache region occupies the most of the SSD capacity and is used as a read/write cache for HDD. It is managed as a circular buffer. The cache region is divided into three zones. The *free zone* is an area which is not used yet. The *cached zone* and the *flushed zone* are used as read caches. Random write requests are reshaped into sequential write, and appended at the end of the cached zone. The appending expands the area of the cached zone by consuming the free zone. The cached zone contains dirty contents which are not yet flushed to HDD. As the contents in the cached zone are flushed by the flush worker, the cached zone is converted to the flushed zone.

3.4 Request Routing Policy

The sequential write performance of HDD is better than or comparable to that of SSD. Therefore, HDD is the better place for bulk write.

HeteroDrive routes a write request based on its size and estimated access pattern. If the size of a request is equal to or larger than a threshold, the request is routed to HDD. If the size is smaller than the threshold, the request is routed by the following estimation.

HeteroDrive counts the number sectors which are written consecutively. If the counter exceeds a threshold, the current pattern is estimated as sequential, and requests are routed to HDD. If a write request is not consecutive to the last request, the counter is halved. As the counter gets below the threshold, the current pattern is estimated as random, and requests are routed to SSD. Since the counter decreases fast by the latter case, HeteroDrive can adapt to random write pattern quickly.

3.5 Mapping Layer

In HeteroDrive, every I/O request must be translated to its physical location. Consequently, it mandates the mapping layer to be lightweight and to allow concurrent access for high performance. Considering the huge space to map, the mapping layer must exhibit small memory footprint as well.

Our design is motivated by the multi-level page table used in virtual memory. A 36-bit key is converted to a 64-bit value through four-level translations. Each translation uses 9 disjoint bits of the key. The 36-bit key allows 64 G sectors and 32 TB space to be addressable.

To limit memory footprint, we divided up the whole page table into a number of *tablets*. A tablet fits in a page (4 KB) and is used as a unit of swapping — tablets can be swapped out to the mapping region described in Section 3.3. The tablets can be brought into memory as needed.

While allocating a tablet, its swap location is allocated from the mapping region, and the starting sector of the swap location is used as the tablet identifier. Tablets which reside in memory are indexed by red-black tree for fast lookup. If a tablet is required, it is looked up through the tree. If the required tablet is not in the tree, the tablet is loaded from SSD located by the tablet identifier. Recall that the tablet identifier is the starting sector of its swap location. If allocating or loading a tablet results in exceeding a memory footprint limit, a tablet is evicted by the LRU policy. If the tablet is dirty, the contents are written to SSD before being discarded from memory.

Using the mapping layer, HeteroDrive defines two mapping tables: *forward map* and *reverse map*. The forward map is used to map read requests to its corresponding sectors. The forward map stores the mapping from a *virtual sector number (VSN)* of HeteroDrive to a *physical sector number (PSN)* of SSD. It only keeps the information for sectors written in SSD, that is, the sectors in HDD are not explicitly mapped. For example, in Figure 3, VSN 7970 is explicitly mapped to PSN 214, while VSN 7974 is not. The VSN 7974 is translated to PSN 7974 at HDD. Such a policy helps to reduce the size of mapping information.

The reverse map is used to flush the cache region. The reverse map stores the mapping from a PSN of SSD to a PSN of HDD. Once the contents in the cached zone is flushed by the flush worker, its reverse map entry is removed.

4. EVALUATION

To evaluate HeteroDrive, we used two servers: database server and load server. Each server is equipped with Intel Core2Quad Q9650 CPU and 4GB RAM. They are connected with Netgear GS748TS Gigabit Ethernet switch.

The database server runs Oracle 10gR2 on Linux. Oracle is configured to use at most 312 MB of memory as data buffer to stress storage. Database file and index are located at the device as a block device to minimize the effect of buffer cache. The transaction log, rollback segments, and temporary table space are located on a separate disk. We used Samsung MCCOE64G5MPD-OVA 64 GB SSD and Seagate ST3500418AS 500 GB SATA-2 HDD. The raw performance of the SSD and HDD are illustrated in Figure 1 as the "Low-end 2 SSD" and "HDD", respectively. The devices are controlled by Intel SRCSATAWB RAID controller which is connected to mainboard at PCI-e 4x.

The load server runs Benchmark Factory v5.5.0. We used the TPC-C benchmark scenario. The benchmark is configured to use 64 warehouses, 32 users, 30 minutes run (10 minutes pre-sampling plus 20 minutes sampling), and 45/43/4/4/4 percentage of transaction mix (new order, payment, order-status, delivery, and stock-level transactions, respectively).

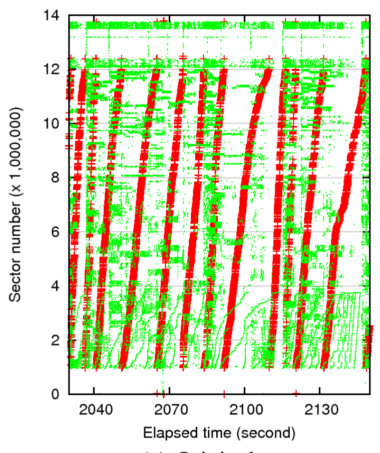
To illustrate the request reshaping, we collected block-level traces using `blktrace` [1]. Figure 4 plots a part of the traces for 120 seconds. Figure 4(a) plots the original access pattern of the OLTP workload. At a glance, the random writes look sequential. However, the average sector distance between two consecutive writes is about 35,000 sectors and most requests are 8 KB in size. If the pattern is applied to HDD, the arm must be moved for each request. Also, many read requests will disturb the sweep motion of the arm. If the pattern is applied to SSD, the small writes will be scattered over the storage. It will result in many garbage collections inside SSD which degrade performance.

Figure 4(b) depicts the access pattern of SSD in HeteroDrive configuration. The random writes are reshaped into sequential writes at sector 13–14 M. Such a reshaped pattern will fully utilize the high sequential performance of SSD. The writes around sector zero and 2 M are caused by forward map and reverse map, respectively.

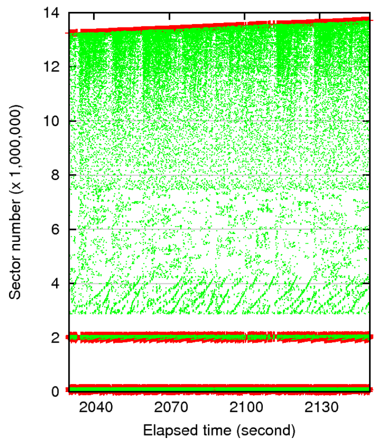
Figure 4(c) depicts the access pattern of HDD in HeteroDrive configuration. Compared to the original access pattern, many write requests are eliminated and only large write requeste are survived. The eliminated writes are redirected to SSD, and reshaped to sequential writes. The flush writes are located around sector 10 M. Although the pattern is not fully sequential, we believe that the large capacity of SSD gives a chance to reshape flush requests. We will validate the claim later by tuning the flush worker's policy.

We measured the transactions per second (TPS) of various storage configurations. HeteroDrives are configured to use 64 GB of HDD and 32 GB of SSD and to keep tablets in memory up to 64 MB. Table 1 summarizes the result.

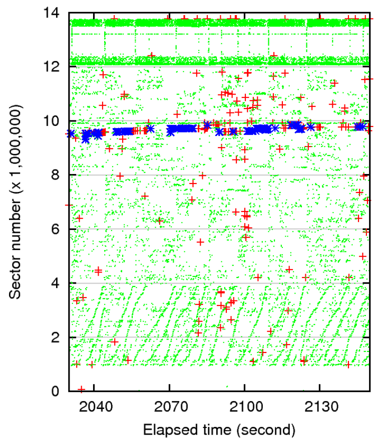
One SSD handles about 41.6 TPS which is almost twice the per-



(a) Original



(b) HeteroDrive SSD



(c) HeteroDrive HDD

Figure 4: Reshaping the storage access pattern in the TPC-C workload. The red, green, and blue marks denote write, read, and flush operation, respectively. The x-axis represents the elapsed time.

Table 1: I/O performance of SSD, HDD, and HeteroDrive configuration.

Configuration	I/O performance (Transaction per second)
HDD	18.14
SSD	41.55
HeteroDrive	54.44

formance of a single HDD (18.1 TPS). As the most distinguishable characteristics of the SSD from the HDD is the smaller random read latency, we concluded that the key contributor of the improved performance is the fast random read performance offered by the SSD.

HeteroDrive configuration achieves 54.4 TPS, which outperforms a single SSD and HDD by about 31 % and 201 %, respectively. There are a number of reasons for the performance improvement. The reshaped access pattern improves the device performance. The high read ratio from SSD (48 %) results in low random read latency. Also, the multiple devices in configuration increases the degree of concurrent request processing. We left the quantitative analysis of the improvement as future work.

5. CONCLUSION

In this paper, we presented the design and implementation of HeteroDrive. It is tailored to the characteristics of the OLTP workload by combining two heterogeneous storage devices, SSD and HDD. HeteroDrive actively reshapes random writes to sequential writes and uses SSD as a large non-volatile cache. As HeteroDrive is transparent to applications, no application modification is needed. We evaluated the performance of HeteroDrive, and the request reshaping is demonstrated by capturing block-level traces. The TPC-C benchmark results show that HeteroDrive improves the transaction rate by up to 201%.

Although contents in SSD are used as a read cache, HeteroDrive does not actively pull up data from HDD to SSD. It is clear that such an approach will not result in significant performance improvement under read-intensive workload. We will address this issue in the next version of HeteroDrive. In addition, we will implement a recovery scheme to cope with sudden power failure and improve the policy of estimating the current access pattern. Evaluation using high-end SSDs and in-depth performance analysis are also planned.

6. REFERENCES

[1] blktrace(8) linux man page. `http://linux.die.net/man/8/blktrace`.

[2] B. Marsh and F. Douglis and P. Krishnan. Flash Memory File Caching for Mobile Computers. In *Proc. 27th Hawaii Int'l Conference*, pages 451–460, 1994.

[3] T. Bisson and S. A. Brandt. Reducing hybrid disk write latency with flash-backed i/o requests. In *Proc. 15th IEEE Int'l Symposium on Modeling, Analysis, and Simulation (MASCOTS)*, 2007.

[4] T. Bisson, S. A. Brandt, and D. D. Long. NVCache: Increasing the Effectiveness of Disk Spin-down Algorithms with Caching. In *Proc. 14th IEEE Int'l Symposium on Modeling, Analysis, and Simulation (MASCOTS)*, 2006.

[5] T. Kgil and T. Mudge. FlashCache: A NAND Flash Memory File Cache for Low Power Web Servers. In *Proc. Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pages 103–112, 2006.

[6] Y.-J. Kim, S.-J. Lee, K. Zhang, and J. Kim. I/O Performance Optimization Techniques for Hybrid Hard Disk-Based Mobile Consumer Devices. *IEEE Transactions on Consumer Electronics*, 53(4):1459–1476, 2007.

[7] I. Koltsidas and S. D. Viglas. Flashing up the storage layer. In *Proc. 34th Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 514–525, 2008.

[8] I. Koltsidas and S. D. Viglas. The case for flash-aware multi-level caching. Technical report, University of Edinburgh, 2009.

[9] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud. Intel®Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. *ACM Trans. Storage (TOS)*, 4(2):1–24, 2008.

[10] E. L. Miller, S. A. Brandt, and D. D. Long. HeRMES: High-Performance Reliable MRAM-enabled Storage. In *Proc. IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, May 2001.

[11] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proc. 4th ACM European Conf. on Computer Systems (EuroSys)*, pages 145–158, 2009.

[12] R. Panabaker. Hybrid hard disk and ReadyDrive technology: Improving Performance and Power for Windows Vista Mobile PCs. `http://www.microsoft.com/whdc/winhec/pres06.mspx`, 2006.

[13] Samsung Electronics Co., Ltd. Samsung Hybrid HDD. `http://www.samsung.com/global/business/hdd/learningresource/whitepapers/LearningResource_WhatIsHybridHDD.html`, 2008.

[14] SourceForge, Inc. fio. `http://freshmeat.net/projects/fio`, 2009.

[15] The Transaction Processing Performance Council. `http://www.tpc.org`.

[16] A.-I. A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning. Conquest: Better Performance Through a Disk/Persistent-RAM Hybrid File System. In *Proc. 2002 USENIX Annual Technical Conference*, 2002.

[17] M. Wu and W. Zwaenepoel. eNVy: A Non-volatile, Main Memory Storage System. In *Proc. 6th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 86–97, 1994.