Sector Log: Fine-Grained Storage Management for Solid State Drives *

Seongwook Jin,Jaehong Kim,Jaegeuk Kim,Jaehyuk Huh, and Seungryoul Maeng
Computer Science Division
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon 305701, Korea
{swjin, jaehong, jgkim}@calab.kaist.ac.kr, {jhuh, maeng}@cs.kaist.ac.kr

ABSTRACT

Although NAND flash-based solid-state drives (SSDs) excel magnetic disks in several aspects, the costs of write operations have been limiting their performance. The overheads of write operations are exacerbated by the fixed write unit (page) of flash memory, which is much larger than the sector size in magnetic disks. A write request from a file system, with a data size smaller than a page, becomes a full page write in SSDs. With the page size hidden internally in SSDs, file systems and applications may not be optimized to a fixed page size. Furthermore, to increase the density and bandwidth of flash memory, page sizes in SSDs have been increasing.

In this paper, we propose a sector-level data management mechanism for SSDs, called *sector log*. Sector log manages a small part of NAND flash memory in SSDs with sector-level mapping, and stores sub-page writes more efficiently than conventional SSDs. While current small DRAM buffers cannot absorb the working set of sub-page writes for certain applications, sector log uses ample persistent storage in flash memory. With the sector mapping mechanism, sector log provides a sector-accessible block device abstraction upon page-managed flash memory.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Mass storage; D.4.2 [**Storage Management**]: Storage hierarchies

General Terms

Design, Performance, Measurement

Keywords

Flash Memory, Storage System, Solid State Drive

*This work was supported by the IT R&D Program of MKE/KEIT. [2010-KI002090, Development of Technology Base for Trustworthy Computing]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'11 March 21-25, 2011, TaiChung, Taiwan.
Copyright 2011 ACM 978-1-4503-0113-8/11/03 ...$10.00.

1. INTRODUCTION

Solid-state drives (SSDs) using NAND flash memory have been replacing traditional magnetic disks in various segments of storage markets. However, flash memory imposes several critical restrictions on SSD designs, one of which is that it requires an erase operation before re-writing a location. Since erase operations take much longer latencies than write operations, updated data are commonly written to a new location pre-erased by garbage collection. As each update changes the physical location of a logical address, a mapping mechanism called Flash Translation Layer (FTL) supports dynamic logical-to-physical re-mapping.

The smallest mapping granularity in SSD is a page, and the page size is much larger than the write unit of hard disks (sector). The page size may vary by the internal organizations of flash chips and SSDs. As current file systems and applications are designed for magnetic disks, they frequently issue writes with a smaller data size than the page size of SSDs. Furthermore, such *sub-page writes* will occur more frequently as the page size of SSDs increases with shrinking flash cell sizes by technology advancement, and with super-paging to improve SSD bandwidth. Sub-page writes can cause significant performance degradation on SSDs since each sub-page write becomes a full-page write in flash memory. To reduce writes to flash memory, such sub-page writes should be merged to full-page writes as much as possible before they are issued to the page-managed flash memory. DRAM buffers in SSDs can reduce sub-page writes for certain applications, but their capacity is too small to accommodate the working set of general workloads. Furthermore, the volatility of DRAM may limit such buffering, to avoid the risk of losing data on power failures.

This paper proposes a new fine-grained mapping mechanism, called *sector log*, which reduces the costs of sub-page writes in SSDs. Sector log manages a small part of NAND flash area with a sector-level mapping, which supports 512B granularity. Sector log resides on top of a conventional FTL, and the FTL manages the rest of the flash area at page granularity. Sub-page writes are written to the sector log region, while full page writes are directly transferred to the FTL. By storing sectors in flash memory, sector log can absorb sub-page writes from applications with a large working set, which cannot fit in small DRAM buffers. Furthermore, as the sub-page write data are written to flash memory quickly, they can persist during power failures unlike volatile DRAM buffers. The available DRAM buffer size is limited, with the DRAM size an order of magnitude smaller than that of sector log in flash memory. As sector log provides efficient

sector-level data management in flash memory, file systems and applications do not need to be modified to a specific write unit of flash memory, which may vary by SSD designs. To the best of our knowledge, this is the first study in SSDs to manage some part of flash memory at sector granularity.

Using a trace-driven simulator, we evaluated the proposed sector log architecture with two on-line transaction processing traces (TPC-C and Financial) and a desktop trace. Sector log reduces 42% of the total writes in the TPC-C trace and 13% of the total writes in the Financial trace with 8KB page size. The decrease in write traffic to flash memory improves the overall SSD throughput by 126% for TPC-C and 53% for Financial with 8KB page size. By absorbing sub-page writes in a sector-mapped storage, sector log improves the performance and endurance of NAND flash based SSDs.

2. BACKGROUND

2.1 SSDs with NAND Flash Memory

NAND flash is a non-volatile semiconductor memory device, with several unique characteristics compared to DRAM and magnetic disks. A flash chip consists of multiple blocks. Each block is divided into pages, typically containing 64 or 128 pages. The size of a page is typically 2KB or 4KB for single-level cell (SLC) flash. The unit of reads and writes is a page. Before overwriting a page, the page must be erased at a block granularity.

Using flash memory chips, solid-state drives (SSDs) provide a block device abstraction to the host system. A typical SSD consists of a host interface logic, NAND flash packages, DRAM and an SSD controller. The SSD controller runs a software layer called flash translation layer (FTL). The FTL translates read/write requests from the host into NAND flash operations (read, write and erase). Using a dynamic logical-to-physical address mapping, the FTL hides the erase-before-write characteristic of flash memory. The SSD controller maintains the mapping information in the DRAM. Also a DRAM buffer can temporarily store read/write pages to allow fast accesses to recently accessed data.

Type	Read	Write	Erase
DRAM	20ns	20ns	
NAND flash	165.6us	905.8us	1.5ms

Table 1: NAND Flash latency

Among the unique characteristics of NAND flash memory, two factors make write operations costly. Firstly, as shown in Table 1, the write latencies to flash memory are 6-7 times longer than the read latencies and the unit of a write is fixed to a page. Secondly, due to the erase-before-write characteristic and limited write endurance of flash memory, each update is written to a new location, requiring a dynamic logical-to-physical mapping by FTLs. To always have some free space for new writes, FTLs execute costly garbage collection processes to erase obsolete blocks. Since the unit of erase operations is a block, the garbage collection process consumes a significant write bandwidth to copy valid pages in victim blocks.

To increase the bandwidth of read/write in SSDs, many SSD manufacturers use interleaving techniques to maximize the parallelism of flash accesses. SSDs can use multiple flash packages with separate channels to access the chips simultaneously. Instead of reading or writing a page, multiple pages can be accessed in parallel to reduce the latencies of accessing a large chunk of data. The technique called *super-paging* combines several NAND flash physical pages, and the SSD accesses the flash memory by the super-page unit for each read and write operation [3, 6, 12].

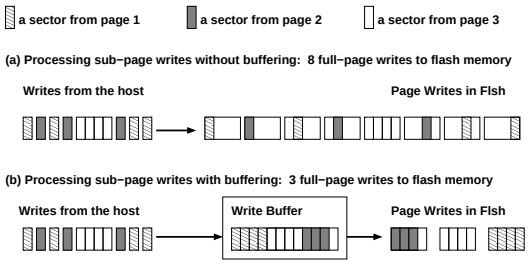


Figure 1: Merging sub-page writes with buffering

2.2 The Costs of Sub-page Writes

NAND flash memory allows write operations only at page granularity. A sub-page write occurs when the host sends a write request smaller than the SSD write unit (page or super-page). In SSDs, sub-page writes consume extra read bandwidth, due to out-of-place updates. Writing a part of a page requires reading the current unmodified page, merging the sub-page write portion with the unmodified part, and writing the merged page to a new free page. After the update, the logical page address must point to the newly written page.

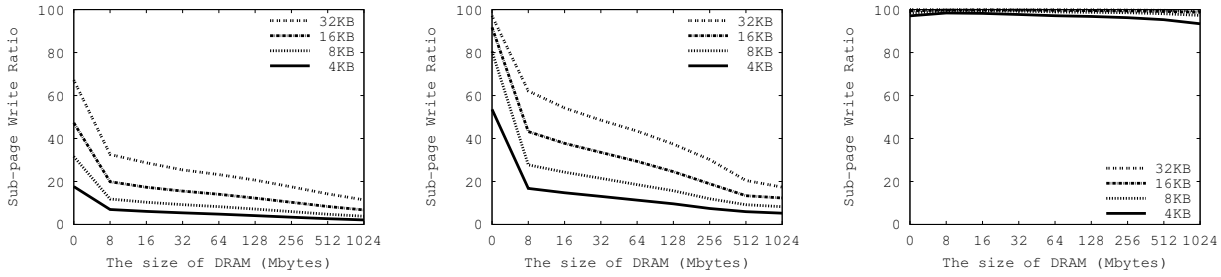
The most important cost of sub-page writes is that they increase page writes to flash memory unnecessarily. If multiple separate requests update parts of the same page, each update requires writing the entire page, increasing total writes to flash memory. Increasing writes to flash memory not only consumes the expensive write bandwidth of SSDs, but also increases the occurrences of garbage collection processes to clean blocks to make free pages. It has been shown that the garbage collection processes can significantly degrade the performance of SSDs [13, 10, 8]. Furthermore, increasing writes to flash memory adversely affects the endurance of SSDs too. As pages are updated more frequently, consuming more limited erase cycles, the longevity of SSDs decreases.

Buffering incoming sub-page writes can reduce page writes, as shown in Figure 1. A small box represents a sub-page write, which updates only a quarter of a page. Without buffering, 8 writes from the host are turned into 8 full-page writes to flash memory. With some buffering, 8 writes are merged into 3 full-page writes, reducing the total writes to flash memory. If a SSD uses a DRAM buffer, it can reduce writes by merging sub-page writes.

3. MOTIVATION

DRAM buffers can improve SSD performance by temporarily storing all recent read and write data. They can potentially reduce sub-page writes by merging them into full-page writes. In this section, we present the ratio of sub-page writes with varying amounts of DRAM buffers, to investigate the working set of write data for our benchmark applications which are described in Section 5.1.

Figure 2 presents the ratio of sub-page writes to the total writes with varying DRAM buffer sizes from 0 (no buffering)



(a) General (b) Financial (c) TPC-C

Figure 2: The sub-page write ratio with varying page size

to 1GB. However, current SSDs use only a small amount of DRAM buffer in a range of 8KB-64MB[12], due to the high cost of DRAM. In the figure, for each trace, we show four different page sizes (4, 8, 16, and 32KB page sizes). The DRAM buffers store recently written data, including both full-page and sub-page writes at page granularity, and evict the least recently used page when the capacity becomes full.

3.1 Sub-page Writes from the Host

Some applications frequently generate small-sized writes without much locality in the DRAM buffer. Due to the lack of locality, such small writes with random patterns, are difficult to be merged to full-page requests in the kernel or DRAM buffer. In online transaction processing (OLTP) workloads, small read/write requests dominate the total requests, as the applications process many small transactions. [14, 20]. Furthermore, most of the file systems and applications are unaware of the internal organization of SSDs, which are hidden inside the SSDs and vary by SSD designs[12]. Due to the hidden page size, the host frequently sends writes smaller than the page size without merging in the file systems.

In Figure 2, for 4KB page size, the General and Financial traces show a significant reduction of sub-page writes even with modest 8-64MB DRAM buffers. With a 64MB DRAM buffer, The Financial trace can reduce the sub-page writes to 11% from 53% before merging. The Financial trace shows a high temporal and spatial locality of writes in the 64MB DRAM buffer (70% write hit ratio in the DRAM buffer).

However, TPC-C shows that 98% of writes from the host are sub-page writes, but the DRAM buffers are too small to hold the sub-page writes of the TPC-C workload. A 64MB DRAM buffer can only reduce the sub-page writes by 1%, and 97% of total writes to NAND flash is still sub-page writes. Even with a 1GB DRAM buffer, sub-page write ratios do not decrease significantly. Since full-page writes without locality evict sub-page writes from the DRAM buffer, even a large 1GB buffer cannot reduce sub-page writes effectively.

3.2 Increasing Page Sizes in SSDs

The page size of flash memory has been increasing to lower the production cost by decreasing flash cell sizes or to improve the bandwidth of SSDs with super-paging.

Advanced lithography: As NAND flash memory technology advances, the density of NAND flash memory increases and each bit cell becomes smaller. However, wires for interconnection do not scale as well as the cell size. Since the area for wires is significant in flash memory, NAND flash manufacturers increase the page size to improve the density with less wiring overheads [5]. The latest 25nm NAND flash memory has 8KB page size which is larger than 4KB page size in the previous generation of technology [18].

Super-paging: As mentioned Section 2.1, super-paging can reduce the latencies of accessing a large contiguous chunk of data. However, not all applications can benefit from the latency reduction by super-paging, if applications access the storage in smaller units than the super-page size. With super-paging, the write granularity of SSDs becomes larger, increasing the sub-page writes.

Figure 2 shows the effect of increasing page sizes with varying DRAM buffer sizes. TPC-C shows high ratios (more than 95%) of sub-page writes for all page sizes. The Financial and General traces show a significant increase of sub-page write ratios from 4KB to 32KB page size. Although modest DRAM buffers (8-64MB) can reduce the sub-page write ratio effectively for 4KB page, the ratios increase significantly with larger page sizes, more than tripling the ratio from 4KB to 32KB.

Advanced lithography may cut production cost, and super-paging can improve the access latencies for a large chunk of data. However, they can potentially increase sub-page write ratios, losing the opportunity to improve the performance further. Sector log mitigates the negative impact of the increase of page sizes.

4. SECTOR LOG ARCHITECTURE

4.1 Overall Architecture

In the context of this paper, a sector is 512B unit, which is the smallest storage unit in the proposed sector architecture. *Sector Log* is a log-based buffer which efficiently stores non-contiguous sectors in flash memory at sector granularity. Sector log stores sub-pages writes in a small part of the flash memory. Logically non-contiguous sectors can be combined and stored in a flash page, since the write unit of flash is still a page. Sector log requires a fine-grained mapping mechanism to map sectors from different logical pages to a physical flash page.

Figure 3 shows the overview of sector log. Sector log is located between the DRAM buffer and FTL, and receives victim data evicted from the DRAM buffer. Sector log has three components, *Page Buffer*, *Sector Map*, and *Sector Data*. The page buffer and sector map reside in DRAM, and share the DRAM capacity with the DRAM buffer. The sector data uses a part of flash memory in SSDs, keeping the data in the persistent storage.

Page buffer can hold one page of data, and the sub-page

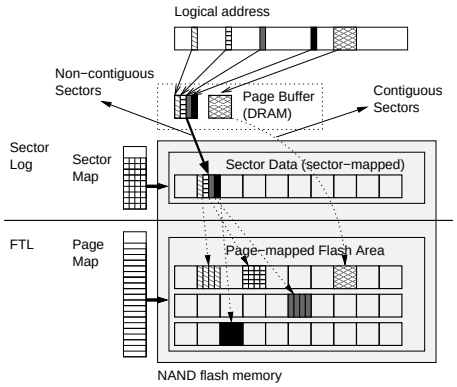


Figure 3: The architecture of sector log

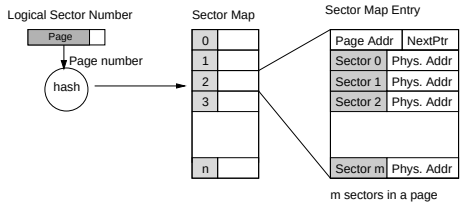


Figure 4: Sector map organization

writes from the DRAM buffer are accumulated in the page buffer. All incoming sub-page writes in multiples of sector unit are accumulated to the page buffer until the page buffer is full. The sectors stored in the page buffer can belong to different logical pages. As soon as one page is full with a set of sub-page writes, the content of the page buffer is stored in the sector data region. If the evicted data size from the DRAM buffer is a multiple of page size, it is transferred directly to the flash memory region managed by the FTL. Sector map records the physical locations of logical sectors. Section 4.2 describes the details of sector map organization.

The actual data of sector log is stored in the sector data consisting of blocks of flash memory. A physical page of the sector data can contain sectors from different logical pages. The sector data region is organized as a circular log, so new data from the page buffer are written to a free page at the head of the log. If the sector data region becomes full, a victim block must be evicted from the sector data region to the FTL. For victim selection, sector log picks the least recently written physical block of the sector data, i.e. the tail block. A victim page from the evicted block may contain sectors from multiple logical pages. For each sector in the victim page, sector log searches the sector map to find other sectors belonging to the same logical page. If other sectors in the same logical page exist in sector log, the sectors of the same page are merged and sent to the FTL together.

4.2 Sector Mapping Mechanism

Sector log provides sector-level management of data by mapping logical sector numbers to physical locations in the sector data region. Sector map is used to locate physical sectors only in the sector data, and the FTL will use its own map to manage its data region at page granularity. The sector map uses a hash-based index structure for fast look-ups, with logical page numbers as hash keys [21].

Figure 4 shows the sector map organization. The sector map has n entries indexed by hashed logical page numbers. Each entry in the sector map has physical locations of all sectors in the same logical page (m sectors in a page in the figure). Each entry also has the original logical page number and a pointer to the next entry for resolving hash collisions. The page-grouped map allows fast searching of all the sectors in a logical page. When a sector is evicted from the sector data region and moved to the FTL layer, all the other sectors belonging to the same logical page must be searched in the sector map. The page-grouped map makes the sector search efficient, as looking up an entry will provide the locations of all the other sectors if they exist in sector log.

4.3 Examples

The examples in this section assume that a page has four sectors. Figure 5 describes write, read, and flush operations of sector log. In the figure, each sector map entry has the physical locations of four sectors. A physical location is represented in three numbers, block number, page number, and offset (block:page:offset).

Write: Figure 5 (a) shows two sub-page writes issued to sector log. The first write is to logical sector number (LSN) 0 and its size is one sector. The second write is to LSN 4 and its size is three sectors. (1) The two sub-page writes are stored in the page buffer. (2) Since the page buffer is full, a page holding the two writes is sent to the free page 0 of block 0. (3) The sector map points to the new physical locations of the two sub-page writes. In the figure, the logical page numbers (LPNs) of the two sub-page writes are 0 and 1 respectively. The map entries corresponding to the two LPNs point to page 0 of block 0.

Read: Figure 5 (b) describes the processing of a read request to LSN 0. The request size is 4 sectors. (1) Sector log looks up the page buffer. (2) If there is no data corresponding the read request, sector log searches the sector map by LPN 0 (LSN 0 is in page 0). The read request accesses multiple sectors, and the data are in physical page 0 and page 1 of block 0. (3) Sector log processes two read operations on the two physical pages.

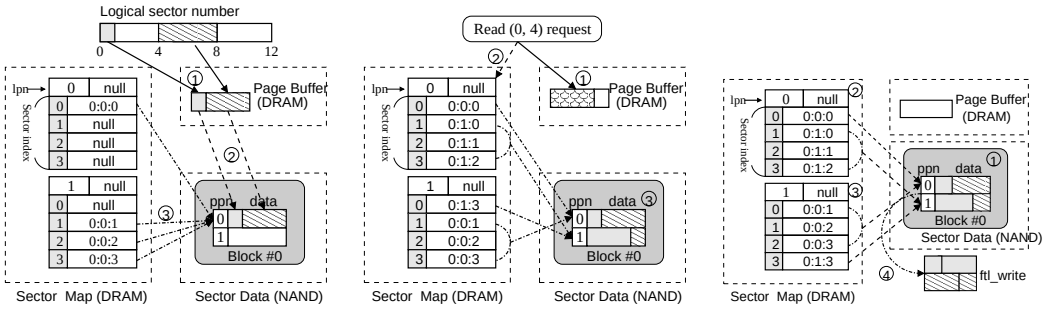
Flush & Merge: If there is no free page in the sector data region, a flush operation must start to make free pages, as shown in Figure 5 (c). (1) Sector log selects block 0 as a victim block, and searches valid pages in the victim block. The valid pages are physical page 0 and page 1. (2) Sector log reads the data in page 0 and 1 corresponding to LPN 0. (3) Sector log reads the data in page 0 and 1 corresponding LPN 1. (4) Sector log merge the data for two pages and transfer them to the underlying FTL. For wear-leveling, sector log exchanges the erased block with a free block from the FTL, to use the wear-leveling technique of the FTL.

4.4 Localities of Sector Log

Sector log must reduce the total writes to the flash memory used both for sector log and FTLs. The total writes from the host ($Writes_{host}$) consists of full-page writes ($Writes_{full}$) and sub-page writes ($Writes_{subpage}$).

$$Writes_{host} = Writes_{full} + Writes_{subpage}$$

Sector log stores only the data of sub-page writes. $Writes_{SL}$ is the number of writes to the sector data region in sector log. Full-page writes from the host are directly sent to the FTL ($Writes_{full}$). With sector log, the total number of writes



(a) Write (b) Read (c) Flush

Figure 5: Sector Log Example

to the flash memory in an SSD is the sum of direct writes to the FTL ($Writes_{full}$), writes to sector log ($Writes_{SL}$), and evicted pages from sector log ($Evicts_{SL}$). The evicted pages are written to the FTL. This model does not include the writes by garbage collection, which varies by FTL designs.

$$Writes_{SSD} = Writes_{full} + Writes_{SL} + Evicts_{SL}$$

For sector log to be effective, $Writes_{SSD}$ must be less than the number of writes from the host ($Writes_{host}$), and sector log must reduce $Writes_{SL}$ and $Evicts_{SL}$ as much as possible. $Writes_{SL}$ depends on how many sub-page writes are accumulated as one page in the page buffer and written to sector log together. Without sector log, every sub-page write must be a full-page write to the FTL. Sector log combines sub-page writes in the page buffer and writes once to the sector data region. Suppose that the average request size (in sectors) for all the sub-page writes is $SectorSize_{avg}$, and the number of sectors in a page is $N_{sectors}$. If the number of sub-page writes is $Writes_{subpage}$, $Writes_{SL}$ is represented as follows:

$$Writes_{SL} = Writes_{subpage} \times \frac{SectorSize_{avg}}{N_{sectors}}$$

For example, with 512B sector and 4KB page, if the average sector size for all the sub-page writes is 2 sectors, then $Writes_{SL}$ is $Writes_{subpage}/4$. In the example, the number of writes to sector log becomes a quarter of the number of the original sub-page writes. As the average size of sub-page writes is smaller, sector log becomes more effective to reduce the total writes.

The second important factor for the effectiveness of sector log is $Evicts_{SL}$. For the sectors stored in sector log, $Evicts_{SL}$ represents how many pages are evicted to the FTL due to the limited capacity of sector log. Sector log attempts to minimize $Evicts_{SL}$ by exploiting localities existing in sub-page writes. Sector log relies on two localities, temporal and spatial localities.

Temporal Locality: Sector log reduces $Evicts_{SL}$ if many incoming sub-pages writes hit the stored sectors in sector log. If a sub-page write hits in sector log, the new sectors will be written to the head of the sector data. The old location of the same sectors in sector log becomes obsolete. When the block containing the obsolete sectors is cleaned, the obsolete sectors are just discarded, without any eviction to the FTL. Therefore, as more sub-page writes hit sector log, $Evicts_{SL}$ is reduced.

Spatial Locality: Sector log attempts to merge sectors from the same page into a full page write. Such merging will be effective, if the host updates the rest of the page while the sectors from the initial sub-page writes are still in the data region of sector log. For example, suppose that four separate writes to different sectors of the same page are written to sector log. When a block containing one of the sector is cleaned, all the sectors are merged and evicted to the FTL as a write request. As more sector writes for the same page are merged until they are evicted to the FTL, $Evicts_{SL}$ decreases. An indirect metric to show the spatial locality is the average write size (in sectors) of the evicted data from sector log. As there is more spatial locality, the average write size of the evicted data will increase compared to the average write size from the host. In Section 5.2, to analyze the existing localities in the workloads, we will present hit ratios to sector log and the average write size for evicted pages from sector log.

5. EVALUATION

5.1 Evaluation Environment

To simulate SSDs, we use a trace-driven simulator, which models several types of FTLs, a DRAM buffer, and NAND flash packages [16]. The simulator processes each request serially, and measures the elapsed time for processing all the requests in traces. We extended the simulator to add the support for sector log. In more details, the simulator counts the number of operations from DRAM, sector log, and NAND flash memory such as page read/writes, block erases, DRAM accesses for processing a given trace. The simulator calculates the total elapsed time by multiplying latency of each operation shown as Table 1.

In this paper, we use two real workload traces and a synthetic workload trace. The characteristics of traces are shown in Table 4. Financial [1] and TPC-C [2] represent OLTP environments. The Financial trace is collected from a financial OLTP application, and TPC-C is from an OLTP benchmark from TPC. Both traces represent applications which may have many sub-page writes. We collected the trace for General from desktop activities such as surfing webs, writing documents, copying and moving multimedia files for 5 days [16]. The DRAM buffer and sector map share 64MB DRAM. The total flash memory capacity is as large as the block address space of each trace with 3% of extra capacity as an over-provisioning space for effective garbage collection. To simulate the effect of garbage collection with warmed-up SSDs, the simulator processes traces twice before measuring results. To show the effectiveness of sector

Page	Sector log Size	Sub-page Hit Ratio / Average Size					
		General		Financial		TPC-C	
4KB	0 MB	0%	3.79	0%	3.12	0%	2.82
	128 MB	65.2%	4.81	42.3%	4.53	12.3%	3.02
	256 MB	77.5%	4.81	49.9%	4.75	22.4%	3.18
	512 MB	99.8%	0.0	99.6%	0.0	34.1%	3.39
	1024 MB	99.8%	0.0	99.6%	0.0	46.9%	3.66
8KB	0 MB	0%	7.14	0%	4.94	0%	3.47
	128 MB	51.0%	9.29	34.0%	8.09	12.3%	3.95
	256 MB	63.7%	9.91	50.3%	8.40	21.8%	4.30
	512 MB	99.8%	0.0	70.9%	8.07	32.9%	4.77
	1024 MB	99.8%	0.0	99.8%	0.0	45.0%	5.40
16KB	0 MB	0%	13.1	0%	7.70	0%	3.94
	128 MB	42.8%	17.4	34.3%	14.1	12.5%	4.94
	256 MB	51.2%	18.6	50.2%	14.6	21.3%	5.60
	512 MB	65.6%	19.7	58.2%	15.5	31.5%	6.54
	1024 MB	99.8%	0.0	99.9%	0.0	42.9%	7.82
32KB	0 MB	0%	23.2	0%	11.3	0%	4.23
	128 MB	37.7%	30.2	34.0%	23.0	11.9%	6.12
	256 MB	43.8%	33.6	44.2%	25.6	20.2%	7.32
	512 MB	56.3%	36.4	59.3%	27.1	29.8%	9.10
	1024 MB	68.7%	37.9	79.9%	23.4	40.8%	11.6

Table 2: locality of sector log with various page size

Page	Sector log Size	Read / Write					
		General		Financial		TPC-C	
4KB	128 MB	1.089	0.989	1.147	0.979	1.498	1.170
	256 MB	1.092	0.984	1.153	0.973	1.651	1.047
	512 MB	1.073	0.976	1.182	0.937	1.843	0.910
	1024 MB	1.073	0.967	1.182	0.937	2.031	0.776
8KB	128 MB	1.226	0.986	1.199	0.951	1.482	0.989
	256 MB	1.241	0.977	1.221	0.932	1.664	0.854
	512 MB	1.242	0.956	1.253	0.911	1.903	0.713
	1024 MB	1.242	0.956	1.290	0.878	2.143	0.581
16KB	128 MB	1.473	0.979	1.232	0.888	1.456	0.823
	256 MB	1.538	0.967	1.282	0.861	1.648	0.680
	512 MB	1.605	0.951	1.313	0.847	1.896	0.539
	1024 MB	1.652	0.919	1.400	0.786	2.136	0.414
32KB	128 MB	1.860	0.965	1.244	0.790	1.415	0.675
	256 MB	2.001	0.944	1.289	0.757	1.598	0.528
	512 MB	2.131	0.919	1.362	0.725	1.851	0.393
	1024 MB	2.408	0.900	1.442	0.695	2.106	0.283

Table 3: Normalized request counts with each page size

log with an FTL, we use the DAC FTL [4]. DAC uses a page mapping scheme, and maintains multiple separate regions for efficient hot-cold separations. Using a page mapping scheme, DAC handles random writes more efficiently than BAST [11] and FAST [15] FTLs, while it requires more DRAM for the page map.

5.2 Access Hit Rates and Average Write Size

As discussed in Section 4.4, the effectiveness of sector log depends on two factors, temporal and spatial localities. In Table 2, we show two metrics for localities: hit ratio for temporal locality, and the average write size of evictions (the number of sectors per eviction) for spatial locality. Firstly, the hit ratios show how often sub-page writes hit the sectors already stored in sector log. As more hits on sector log occur, less valid sectors are evicted from the sector log, reducing the number of writes to the flash memory managed by the FTL. Secondly, the average write size of evictions shows how many sub-page writes to the same page are merged while they are in sector log. If there is no eviction from sector log, the average write size is zero.

General and Financial show high hit ratios of 77.5% and 49.9% with a modest 256MB sector log in 4KB page size. In the General workload, more than 77% of incoming sub-page writes from the host are absorbed in sector log. For 4KB page, the sub-page writes of General and Financial fit in a 512MB sector log, without any eviction from sector log. Since sector log stores only sub-page writes, with a relatively small log size of 512MB, sector log can keep all the sub-page write data for the two traces for 4KB page size. However, as the page size increases to 32KB, sector log can no longer hold the working set of sub-page writes, since sub-page writes increase significantly. TPC-C shows relatively low hit rates for all page sizes. With a 1024MB sector log, 46.9% of sub-page writes hit in sector log with 4KB page size. In general, the benchmarks traces have a high temporal locality with 256MB-1GB sector logs. The results show that sector log can absorb a significant portion of sub-page write data, minimizing the eviction to the FTL.

Trace	Addres Space	Request Size	Read	Write
Financial	11.7GB	25.6GB	36%	64%
TPC-C	29.5GB	307.7GB	86%	14%
General	18.3GB	22.4GB	39%	61%

Table 4: The characteristics of traces

The average write size of evicted sub-page writes increases as the sector log size increases, showing that with more capacity, the sector log can hold sector data longer to exploit more spatial locality. For General and Financial, a small 128MB sector log can extract most of the available spatial locality. However, TPC-C requires a large 1GB sector log, to effectively merge the sectors from the same page, due to the large working set of sub-page writes. As the page size increases, sector log can extract more available spatial locality.

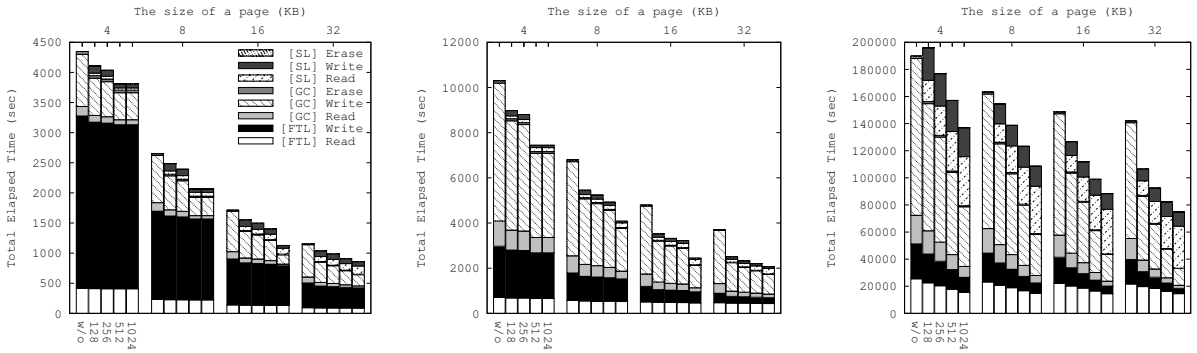
5.3 Read and Write Count Changes

In this section, we show how many write and read operations sector log adds or reduces. Table 3 shows the sum of the total read or write operations in the sector log and FTL regions, excluding read and write operations generated by garbage collection. The read and write overheads of garbage collection vary by FTL designs, and in this section, we evaluate only the FTL-independent aspect of sector log. In the Table 3, read and write counts are normalized to a base SSD without sector log.

In general, sector log increases the number of read operations in the sector log and FTL data regions, but reduces write operations. In SSDs, each write causes much more read and write operations later for garbage collection, which are not included in the table. The write reductions are significant for TPC-C, by 23% with 4KB page, and by 71% with 32KB page. In the next section, we will show how the reduction of writes actually improves the overall throughput.

For the General and Financial traces, sector log does not reduce writes significantly with 4KB and 8KB page sizes. It is because a small DRAM buffer already filters out many sub-page writes, as the first working sets of sub-page writes are small in the two traces. However, with 16KB and 32KB pages, the reductions increase, by 21% with 16KB page and by 30% with 32KB page in Financial. For those two traces, a modest size of sector log, with less than 512KB, can reduce writes effectively.

As the most of the write requests in TPC-C are sub-page writes, TPC-C shows significant reductions of writes even with 4KB page. However, in TPC-C, sector log requires a



(a) General (b) Financial (c) TPC-C

Figure 6: Total elapsed times with DAC FTL

large data capacity to absorb enough sub-page writes. TPC-C requires at least 1024MB of data capacity for sector log to reduce write operations in flash memory significantly. This requirement for a large capacity re-affirms that a small separate DRAM buffer for sub-page writes is not effective enough for TPC-C.

5.4 Performance of Sector Log

Figure 6 presents the elapsed times with sector log combined with the DAC FTL [4]. The elapsed times are further divided to reads and writes to the FTL; reads, writes, and erases for garbage collection; and reads, writes, and erases for sector log.

Firstly, the results show that the total elapsed times decrease as the page size increases. With a large page size, the FTL can write and read a large chunk of data at once, improving the overall throughput. The performance improvements by increased page sizes are significant for General and Financial even without sector log, since their writes can be merged easily to large page sizes. However, TPC-C shows modest reductions of execution times, since the workload issues many sub-page writes in random patterns.

As discussed in Section 5.3, without the effect of garbage collection, using sector log reduces write operations, but increases reads. However, after including read operations for garbage collection, sector log reduces the total reads to flash memory, since it reduces the occurrence of garbage collection processes. The garbage collection often requires reading multiple pages to copy valid pages from a victim block. The overheads of such valid page copy are significant in SSDs, and increase as the number of writes to SSDs increases. Figure 6 shows that sector log reduces the erase, read, and write operations for garbage collection significantly for all traces.

The TPC-C results in Figure 6(c) show the benefit of sector log clearly. The performance improves significantly, by 126% with 8KB page size. The benefit of sector log increases with larger page sizes. Not only the read and write operations in the FTL decrease, but also, the overheads for garbage collection reduce significantly. Sector log degrades performance, if the size of sector log is 128MB and the page size is 4KB. It is because the sub-page write working set of TPC-C is much larger than the other traces and the 128 MB sector log is not large enough to absorb it.

5.5 Sector Log Overheads

Sector log incurs two types of overheads. The first type is the overhead in memory usage to maintain the sector map in DRAM. As shown Table 5, the amount of memory used for the sector map is small, under 36MB. For 4KB page, Financial uses only 1.8MB, while General uses 4.2MB. As the page size increases, the memory usage increases since writes to more pages become sub-page writes. In the experiments for this paper, we share the limited DRAM both for sector map and a DRAM buffer, not to give an unfair advantage to sector log. TPC-C uses a relatively large amount of memory for the sector map. However, the DRAM buffer is quite ineffective for TPC-C due to low locality in the DRAM buffer, and using the DRAM space for the sector map results in better performance.

Trace	4KB	8KB	16KB	32KB
Financial	1.8MB	2.7MB	4.4MB	6.8MB
TPC-C	16.2MB	20.1MB	26.8MB	35.4MB
General	4.2MB	5.6MB	7.6MB	9.7MB

Table 5: Memory usage for sector map

Trace	4KB	8KB	16KB	32KB
Financial	1.2	1.3	1.4	1.6
TPC-C	1.9	2.1	2.4	2.8
General	1.1	1.1	1.2	1.4

Table 6: Average hash search counts per request

The other overhead is searching the sector map. The sector map uses a hash index structure for storing mapping information, using linked lists to resolve hash collisions. As shown Table 6, the hash searches take on average only 1-2 linked list traversals, except for TPC-C with larger than 4KB page sizes (3 traversals)

6. RELATED WORK

To the best of our knowledge, this work is the first study to use part of flash memory at sector granularity when the page size is much larger than 512B. The page sizes of early flash implementations were as small as the sector size, and with such sector-unit flash, some FTLs could support sector-level mapping [15]. However, with the current capacity-optimized flash memory, the page size is much larger than the sector size.

Several different ways to implement dynamic mappings for FTLs have been proposed. A page-mapping FTL consumes a large amount of memory but improves SSD performance by reducing the costs of garbage collection. Using such page-mapping, Chiang et al. segmented NAND flash memory into regions to improve hot-cold separation [4]. Gupta et al.

proposed to cache the page mapping information on demand to reduce memory usage [7]. Although block-mapping FTLs use much less memory than page-mapping FTLs, they suffer some performance degradation compared to page-mapping FTLs [17]. Hybrid-mapping FTLs combine page-mapping and block-mapping in an FTL [9, 16]. As a hybrid-mapping FTL, BAST FTL provides block-level associativity, using a log block associated to a data block [11]. FAST FTL uses a fully-associative log buffer, relaxing the log mapping restriction of BAST [15].

There have been several studies on DRAM write buffers to reduce the cost of handling writes to flash memory. The write buffer can merge incoming writes and evicts pages in the same block to the FTL [8, 10]. Such buffering mechanism can be optimized to evict clean pages rather than dirty pages [19].

7. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a new sector-level logging architecture to alleviate the overheads of sub-page writes in SSDs. Sector log efficiently stores sub-page writes at sector granularity in flash memory. Exploiting the spatial and temporal localities of sub-page writes in applications, sector log reduces write traffics to the underlying FTL-managed flash memory. The decrease of write traffics reduced not only the write bandwidth consumption, but also the occurrences of expensive garbage collection processes.

Based on this architecture, we are extending our study to investigate crash-recovery and wear-leveling. Sector log can use spare area in flash memory to store logical addresses to recover from a crash. In case that the spare area is too small to fit the entire sector address, a small part of data area can be used for recovery addresses. For wear-leveling, sector log uses the mechanism provided by the underlying FTL. Sector log receives free blocks from the FTL and return victim blocks to the FTL. Therefore, as the FTL manages free blocks for wear-leveling, sector log does not need to directly address wear-leveling issues. However, investigating further optimizations of wear-leveling techniques for sector log will be our future work.

8. REFERENCES

[1] OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

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

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of USENIX Annual Technical Conference, ATC 2008*, pages 57–70.

[4] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using Data Clustering to Improve Cleaning Performance for Flash Memory. *SoftwareâĂŤPractice and Experience*, 29(3):267–290, 1999.

[5] E. Deal. Trends in NAND Flash Memory Error Correction, 2009.

[6] S. Electronics. NAND Flash-based Solid State Disk Module Type Product Data Sheet, 2007.

[7] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *Proceedings of ACM International conference on Architectural support for programming languages and operating systems, ASPLOS 2009*, pages 229–240.

[8] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee. FAB: Flash-aware Buffer Management Policy for Portable Media Players. *IEEE Transactions on Consumer Electronics*, 52(2):485–493, 2006.

[9] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In *Proceedings of ACM International conference on Embedded software, EMSOFT 2006*, pages 161–170, 2006.

[10] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proceedings of USENIX Conference on File and Storage Technologies, FAST 2008*, pages 239–252.

[11] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho. A Space-efficient Flash Translation Layer for CompactFlash Systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, 2002.

[12] J.-H. Kim, D. Jung, J.-s. Kim, and J. Huh. A Methodology for Extracting Performance Parameters in Solid State Disks (SSDs). In *Proceedings of IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2009*, pages 1–10.

[13] J. Lee, S. Kim, H. Kwon, C. Hyun, S. Ahn, J. Choi, D. Lee, and S. H. Noh. Block Recycling Schemes and Their Cost-based Optimization in NAND Flash Memory Based Storage System. In *Proceedings of the ACM International conference on Embedded software, EMSOFT 2007*, pages 174–182.

[14] S.-W. Lee, B. Moon, and C. Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of ACM International conference on Management of data, SIGMOD 2009*, pages 863–870.

[15] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation. *ACM Transactions on Embedded Computing Systems*, 6(3):18, 2007.

[16] Y.-G. Lee, D. Jung, D. Kang, and J.-S. Kim. μ-FTL:: A Memory-efficient Flash Translation Layer Supporting Multiple Mapping Granularities. In *Proceedings of ACM International conference on Embedded software, EMSOFT 2008*, pages 21–30.

[17] M-Systems. Flash-memory Translation Layer for NAND flash (NFTL), 1998.

[18] J. Nq. IMFT's 25nm NAND Flash Will Cut Production Costs in Half, Spur New SSDs, 2010.

[19] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee. CFLRU: A Replacement Algorithm for Flash Memory. In *Proceedings of the ACM International conference on Compilers, architecture and synthesis for embedded systems, CASES 2006*, pages 234–241.

[20] G. J. Powell. Oracle High Performance Tuning for 9i and 10g, 2003.

[21] T. Wang. Integer Hash Function. http://www.concentric.net/ Ttwang/tech/inthash.htm, 2007.