Reducing Computation Overhead of Flash Translation Layer with Hash

ILHOON SHIN
Department of Electronics and Information Engineering
Seoul National University of Technology
Nowongu Gongleungdong SNUT 3th building 207, Seoul
South Korea
ilhoon.shin@snut.ac.kr

Abstract: - NAND flash memory does not support the overwrite operation and thus it deploys the flash translation layer which performs the out-place update. When designing a flash translation layer, memory consumption, computation complexity, and garbage collection overhead should be low. However, the representative sector mapping scheme of flash translation layer, FAST causes a huge computation complexity even though it minimizes the memory consumption and the garbage collection overhead. This paper presents a hashed page table for the FAST scheme to reduce the computation overhead. The simulation result shows that the presented method contributes to reduce the computation overhead considerably.

Key-Words: - NAND flash memory; flash translation layer; sector mapping; log block; hash;

1 Introduction
NAND-based block devices such as SD cards, USB drives have been popular in a mobile storage market. Recently, NAND-based block devices are even trying to enter a PC and a server storage markets as a form of solid state drives.

NAND-based block devices consist of NAND flash memory, internal RAM, and internal controller. NAND flash memory stores data, and a special firmware called Flash Translation Layer is executed by the internal controller with the internal RAM. As NAND flash memory has different I/O interface and characteristics from hard disk drives, the flash translation layer emulates the block device interface and exports the NAND flash memory like a block device to the above file system. For low manufacture cost, the internal controller is slow and the internal RAM is small, and thereby a memory requirement and a computation complexity of the flash translation layer should be low.

NAND flash memory is a kind of EEPROM (Electrically Erasable Programmable Read Only Memory) and does not support an overwrite operation. To update a cell, the cell should be erased first. As the erase is order of magnitude slower than a read/write operation and the unit of the erase operation is bigger than the unit of the read/write operation, the overwrite operation cannot be performed as an in-place update, which writes the new data to the original cell. Thus, usually flash translation layer performs an out-place update, which writes the new data to other clean cell. When doing the out-place update, the location of the valid data becomes different on every overwrite operation. Thus, the flash translation memory should maintain the location of the valid data. For a fast performance, the location of the data is maintained in the internal RAM as well as in the NAND cell. In order to reduce the memory requirement, this location information should be small.

Meanwhile, as performing the out-place update, the clean cell will be exhausted and we need to reclaim the obsolete cell that contains the obsolete data. To reclaim the cell, it should be erased, which is very slow (usually, erasing 128KB takes 2ms.). Thus, the frequency of the reclamation process should be low for a good performance of NAND-based block devices.

The previous studies have tried to design the flash translation layer which reduces the frequency of the reclamation process while at the same time requiring the small memory usage and the low computation overhead. However, the most studies have failed to achieve the goal. For example, the schemes with low memory overhead and computation complexity failed to reduce the frequency of the reclamation process, and the schemes that succeeded in reducing the frequency of the reclamation process caused a huge computation overhead. We carefully predict that the ideal flash translation layer is almost impossible to implement and thus choose the alternative approach. Our goal is to reduce the frequency of the reclamation process and the computation overhead for a good performance with utilizing additional memory. The increased size of the memory should be reasonable. In this paper, we achieve the goal by modifying the existing policy slightly to use a hash.

The remainder of the paper is organized as follows. Section 2 describes the previous flash translation layer schemes. Section 3 presents the approach to reduce the
computation overhead. The performance of the new approach is verified with a trace driven simulation in section 4. Section 5 draws a conclusion.

2 Flash Translation Layer

NAND flash memory consists of pages and blocks. Pages are the unit of the read/write operation. A page is generally 2KB in size. Blocks are the unit of the erase operation. A block is generally 128KB in size. As described in the section 1, NAND flash memory does not support the overwrite operation, and thus the flash translation layer performs the out-place update and the location of the valid data becomes different on every overwrite operation. The flash translation layer remembers the location of the valid data with the mapping table between the sector number and the actual physical location.

The original flash translation layer adopted the page mapping scheme [1], which maintains the location by a page. In the page mapping, the data are written in a page unit. For example, when overwriting a sector 32, the whole page that the sector 32 belongs to is written in a clean location and the mapping table is updated applying the change of the location. Thus, the number of entries of the mapping table is the same with the number of physical pages in NAND flash memory and the memory requirement of the page mapping scheme increases proportionally to the number of the physical pages. If the page size is 2KB, the capacity of the NAND-based block device is 16GB and the entry size of the mapping table is 4 bytes, the total size of the mapping table is 32MB, which is too large for NAND-based block devices.

In order to reduce the memory requirement, the block mapping scheme [2] was presented. It moves the data as the unit of a block. For example, when overwriting a sector 32, the whole block that the sector 32 belongs to is written in a clean location and the mapping table is updated. Thus, the number of entries of the mapping table is the same with the number of physical blocks in NAND flash memory. If the block size is 128KB, the capacity of the NAND-based block device is 16GB and the entry size of the mapping table is 4 bytes, the total size of the mapping table is 512KB. The drawback of the BAST scheme is that it is vulnerable against the widely spread random write pattern. As the log block can be maximally one data block in the BAST scheme, the log blocks are frequently exhausted in the random write pattern, which is called a log block thrashing problem. In the widely spread random write pattern, the log blocks are evicted under-utilized.

In order to solve the log block thrashing problem, the FAST (Fully Associative Sector Translation) scheme [4] allows the log blocks shared by multiple data blocks. The log blocks are managed by a FIFO (First-In First-Out) manner. The write requests are accumulated to the current working log block, regardless of the sector number. When there is no free space in the working log block, the next log block becomes the working log block. If there is no clean log block, the first-in log block is merged with the associated data blocks, which may take a very long time because it accompanies several erase operations and hundreds of read and write operations. The FAST scheme utilizes the log blocks maximally and thus it mitigates the log block thrashing problem even in the widely spread random write pattern. However, the computation overhead becomes huge. On the read request, the flash translation layer should find the location of the valid sector. The valid sector resides either in the log blocks or in the data block. In the FAST scheme, the computation overhead of finding the valid sector is reasonable, because the associated log block is maximally one. It is sufficient to scan the page mapping table of the associated log block. However, in the FAST scheme, the sectors are distributed to the entire log blocks, and thus we need to scan all the log blocks. The same problem happens also on the write request, because the flash translation layer has to find the current valid sector to invalidate it. As the flash translation layer is performed by low-speed controller inside the device, a huge computation overhead degrades the I/O throughput considerably.
The recent other sector mapping schemes such as SAST (Set Associative Sector Translation) [5] and LAST (Locality Aware Sector Translation) [6] are all based on the concept of sharing log blocks and thus have the same problem with the FAST scheme. Conclusively, the ideal scheme which causes a small memory usage and a low computation overhead and reduces the frequency of the merge process has not yet presented.

3 Reducing Computation Overhead with Hash

The FAST scheme copes well with the random write pattern. The problem was the huge computation overhead of finding the current valid sector. The goal of this work is to reduce the computation overhead of the FAST scheme.

The huge computation overhead results from the linear search of the sector. Fig. 1 shows the page mapping table of a log block in the FAST scheme. As seen in the figure, the table is indexed by a physical page index (PPI) of the log block, and the value is the sector number. From the table, we can easily know the sector number that each physical page stores. However, we need to scan the entire table in order to find the location of a specific sector. For example, if we need to know the location of the sector 32, we should search the entire table. Especially in the FAST scheme, as the sectors can be stored in any log block, we should scan the page tables of all the log blocks in the worst case. Thus, the computation overhead increases proportionally to the number of log blocks.

![Fig.1 The page mapping table of the log block in the FAST scheme.](image1)

To restrict the computation overhead of linear search, hash or various forms of trees have been used. In this work, we choose a hash because it is easy to implement and accompanies low computation overhead. Fig. 2 shows the hashed page table. Each hashed page table entry contains the sector number and the location information (PBN (physical block number) and PPI (physical page index)), and is hashed by the sector number. For example, when we find the location of the sector 1000, we first find the header0 by the hashing function such as a modular function using the sector number. Then, we search the linked hash list comparing the sector number. If the entry that has the target sector is found, we can easily get the physical location information of the sector. Thus, the computation overhead is determined by the number of the entries of each hash list. As each header is 4 bytes in size, increasing the number of header is not a big burden. Note that the hashed page table contains only the entries of the valid sectors. When the sector is invalidated, the corresponding entry is removed from the hash list. Thus, the number of the hashed entries is the same with the number of the valid sectors of the log blocks.

![Fig.2 The hashed page mapping table.](image2)

The tradeoff of the reduced computation overhead is an increase of memory consumption. In the worst case, the number of the hashed entries is the same with the number of pages of all the log blocks. Thus, the memory consumption increases proportionally to the number of log blocks. As each entry is 16KB in size, if the block size is 128KB, the page size is 2KB, the capacity of the NAND-based block device is 16GB, and 512MB (3.125%) is used as log blocks, the hashed page table occupies 4MB maximally.

4 Performance Evaluation

We evaluate the effect of the hashed page table with a trace-driven simulation. The traces are random write patterns that have different locality. In the Synthetic1 trace, all the sectors are written with the same probability. In the Synthetic2 trace, 10% of the sectors occupies 90% of the write requests. The storage size is 8GB and the total size of the written data is also 8GB. The simulator counts NAND read, write, and erase operations during execution, and the total I/O time is calculated by the formula (total elapsed time = read count*read latency + write count*write latency + erase count*erase latency). We also consider the searching time of the valid sector. The simulator counts the page mapping table lookup, and the searching time is...
approximately calculated based on the assumption that the page mapping table lookup takes 3 CPU cycles (load, compare, and store) and the internal controller speed is 100 MHz. The total elapsed time is calculated by adding the searching time to the total I/O time. Figure 3 and 4 show the result. X axis denotes the log block ratio as to the entire storage space. It is varied from 0.5% - 5%. Y axis denotes the total elapsed time including the lookup time of the page table. As seen in the figure, the FAST scheme delivers a much better performance than the BAST scheme by sharing the log blocks. However, the performance gap becomes smaller as the number of log blocks increase, because the searching time increases considerably. The performance is even degraded in the pure FAST scheme. In contrast, the FAST scheme with hashed page table delivers a better performance by reducing the count of the page table lookup, as the number of log blocks increases. The performance gap with the pure FAST scheme reaches up to 4 times in the Synthetic2 trace.

5 Conclusion
In this paper, we presented the method to reduce the computation overhead of the flash translation layer with mitigating the log block thrashing problem in the widely distributed random write pattern. We supplemented the FAST scheme because it coped well with the random write pattern and however it caused a considerable lookup time of the page mapping table. The hashed page table was added to the FAST scheme in order to reduce the lookup time. The simulation result showed that the hashed table contributed seriously to reduce the lookup time. The improvement reached 4 times.

Acknowledgement
This work was supported by National Research Foundation of Korea Grand funded by the Korean Government (2009-0074359).

References: