Implementation of Cache Schedule Strategy in Solid-state Disk

Baoping Wang
School of software, Nanyang Normal University, Nanyang 473061, Henan, China

Chao Yin*
School of Information Science and Technology, Jiujiang University, Jiujiang 332005, Jiangxi, China
*corresponding author(E-mail: ChaoYin2010@163.com)

Xingang Zhang
School of Computer and Information Technology, Nanyang Normal University, Nanyang 473061, Henan, China

Abstract
The solid-state disk (SSD in short) based on the flash memory is the best outstanding due to its advantages. In order to improve the performance of SSD, it should be designed into special structure to achieve the goal under the same conditions of hardware structure. There is a complex software layer in SSD, translation layer flash (FTL in short), which is the core technology in the SSD software. Because of the FTL, the file system can be used directly as a common SSD block device. Nowadays, the storage systems use the page level mapping FTL based on requirements, which is called DFTL. There are additional read and write operations too much, and there is no good use of the local load space problem. We have developed an improved FTL technology, named IFTL, which uses a new technology of write back operation to solve the problem of DFTL. It is designed to update the entries in the same pages and the same nature when the dirty entries are dropped off. This can avoid the extra operation problem of an additional read or write mapping table when dropping off the dirty entries in DFTL. We evaluate our IFTL prototype by simulation test and compare with DFTL. The result shows that IFTL can significantly improve the performance by reducing the overhead of the additional operation brought by the read and write mapping table. The average cost of reading map table with IFTL is 68.96% lower than that with DFTL, and the cost of writing map table is reduced by 97.8% averagely.

Keywords: Solid-State Disk (SSD), Cache Schedule Strategy, Translation Layer Flash (FTL), Write Back Operation

1. INTRODUCTION

With the development of multi-core technology and a large number of threads are widely put into use in recent years, the processor performance have been improved millions of times. But the performance of the disk storage devices hasn’t increased correspondingly while I/O has become an important factor restricting the performance of the data center. People have more and more requirements to delay, bandwidth and power consumption of the storage system. In this demand, a large number of new storage media appear, such as FLASH Memory, phase change storage (PCM in short), Magnetic Random Access Memory (MRAM in short) (Li, Shi and Gao et al., 2015). PCM exist high prices and high power consumption problems, which haven't been solved temporarily. Its capacity is so small that it can only be used in the memory level, rather than in the storage hierarchy. Though MRAM breakthrough capacity constraints, it can’t be used in large-scale deployment. From above all, we can see that only flash memory have the potential to be able to study.

Currently SSD has been widely used in various industries as storage medium, such as aviation, military communications and other fields. It has been formed the market of billions of dollars one year.

SSD based on flash memory has its unique operating characteristics. Neither has it so hard with mechanical parts, nor does it need rotating the disk head when requesting for data. The storage mediums are many flash chips. When reading or writing the flash memory chips, we deal with the operations through a layer of software layer. The main task of this software layer is to convert read and write request from the above system to the flash operation command.

Although SSD has many advantages, such as higher read and write speed, reliability and low energy consumption, it has its own mechanisms that the system must erase before write and erase flash chip with limited number of features. These mechanisms let SSD have multi constraints in practical application. It becomes a hot issue at the present stage about how to improve the reading and writing performance, and reducing the numbers of erase times to improve the service life of SSD.

How to improve the performance of SSD has been a hot issue in academic and industrial research. There are two ways to improve the performance of SSD at present. The first one is the new flash translation layer
algorithms, which can provide more efficient address mapping strategies, garbage collection algorithms. The second one is the use of local principle of the load. The system adds a cache in the flash translation layer, which give effective caching strategy.

There are three address mapping strategies in SSD: page level mapping, block level mapping and hybrid mapping three. The mapping strategies determine the garbage collection and wear leveling. Because the page level mapping granularity is small and flexible, so people usually use page level mapping strategy in high performance solid state storage. The table is usually large, while the RAM space is relatively small and the cost is high. It has become a hotspot problem that how to reduce the size of RAM in the mapping table and still retain even improve the performance of SSD with the page level mapping table. Gupta et al.(Wu, He and Xie et al., 2013) has proposed DFTL, which is a new page level mapping FTL based on the requirements. It is purely page level mapping. The map entry stored in the cache is a subset of the flash memory. When a request hits or misses in the cache, the data can be found accurately.

We propose a new flash translation layer algorithm IFTL based on the DFTL algorithm, which is mainly aimed at the disadvantages of DFTL. We use write back operation in the process of driving a dirty entries out of the cache. Because the dirty entries is the latest, we use write back operation. At the same time, we also write back the latest mapping entries in the same mapping entry page. It turns out two read and write operation into one operation, which can improve the efficiency a lot.

The contributions of this paper are described as follows:

- We put forward write back operation, which can write back the dirty entries in the same map page. The operation can reduce the cost of read and write map table when a dirty item is written in the cache every time.
- Experimental results show that IFTL algorithm has improved a lot in the performance of the system.

The rest of this paper is organized as follows. Section 2 describes related works. The design and implement of IFTL algorithm is introduced in Section 3. Section 4 is the experimental result and evaluation. Section 5 is the conclusion.

2. RELATED WORKS

2.1. SSD

Compared with the traditional hard disk, SSD not only has obvious performance advantages, but also has higher reliability and lower power consumption because of without mechanical components. It can adapt to the bad environment such as high temperature, severe vibration etc. As a new type of storage devices invented for solving many problems of traditional disk, SSD can alleviate the bottleneck problem of I/O performance. It has become a hot issue in the field of industry.

There is a huge difference in the internal read and write principle between SSD (Dong, Xie and Zhang et al., 2013; Lee, Sim and Hwang et al., 2015) and the traditional mechanical hard disk. SSD can provide an interface of block equipment, which can be used the same as mechanical hard disk.

SSD takes the page as the basic unit of read and write, while the basic unit of erasure is the block. SSD has some characteristics as follows. First, erase before write. Different from the traditional mechanical hard disk, SSD must find a free page to write instead of writing coverage in the original data stored. The second characteristic is the limited number of erase. As mentioned earlier, most of the current SSD is used as storage medium can endurance rewrite cycle of about 10000 times. The flash chip was erased after ten thousand times on the same position, it will not be able to continue to use. It will lead more and more bad blocks with the solid state disk used constantly. The users will feel that the storage space of SSD becomes smaller and smaller. In order to extend the life span, a series of software structures are used to manage the flash memory chip.

2.2. FTL

The software structure of SSD system mainly includes: data cache management module, FTL module and other modules. FTL is the most important software structure in SSD. It converts the request from the upper file system to a request for a specific physical location on SSD storage chip, and then takes read, write and erase operations on the specific storage unit.

FTL is divided into three parts: address mapping, garbage collection and wear leveling. There are three kinds of address mapping algorithms: page level mapping, block level mapping and hybrid mapping.

Page level mapping (Park, Kim and Choi et al., 2015; Lee, Shin and Kim et al., 2008) takes page as the basic unit of the mapping. In this mapping algorithm, the logical page data can be stored in any physical flash memory, which is a fully associative relationship. Page level mapping strategy has strong flexibility, while it has relatively large mapping table due to the small size page. It will take up more valuable RAM space and increase the cost.

Block level mapping (Shi, Li and Xue et al., 2011) regards blocks as the basic mapping unit. The relationship between the mapping page and the physical page is set-associative, that is, the logical page can only be stored in a specified physical block.
Hybrid mapping (Zhao, Venkataraman and Zhang et al., 2014) is a tradeoff between page level mapping and block level mapping. This mapping method takes into account the advantages and disadvantages of the page level mapping and the block level mapping. It divided the storage area of the flash memory space into two parts: the data area and the log area.

3. THE IFTL ALGORITHM

3.1. The Design of the DFTL Algorithm

The design of DFTL algorithm is inspired by TLB. It uses the local property of the load, which stores the most frequently used mapping entries in limited SRAM space, and puts the entire table and stored data into the flash memory chip.

The map entry stored in the cache is a subset of the flash memory. When the request in the cache hit, it will find the corresponding physical page number, and then find the physical page through the physical page number in the flash. If the request doesn’t hit in the cache, the result will be divided into two kinds of situations to talk about:

The first situation is the cache full. The mapping entries stored in the cache reach the maximum value, and cannot be used to save the new map entries. The system will find an existing map entry to eliminate. If the map entry is a dirty entry, which means the entry mapping is in the cache but not in the mapping table in the flash. It will go through a global data dictionary (GTD in short) to find the physical mapping page where the mapping entries load. Read the map entry page, update it and then write it back to the flash. If the map entry is not the newest, the system will remove the mapping entries from the cache directly. After that, the free space can hold the mapping entries of the current request from the flash mapping table to the cache.

If the request is a read request, it need read requests from the flash global mapping table entry into request mapping table in the cache. If it is a write request, it need find a free physical page and establish the corresponding relationship with the current request logical page.

Through the analysis of the map cache strategy in DFTL, we can know that when the load is very good, the life rate in cache is very high, and vice versa. But there is a big problem in this cache strategy. When the request is almost write request, the cache will be written a lot of latest mapping entries. These entries only exist in cache before they are dropped out of the cache. When this mapping entry was dropped out of cache, each one will bring out an additional read mapping table overhead and write mapping table overhead. If each new request will bring this additional operation, it will affect the performance of SSD. On the other side, the caching strategy doesn’t consider the local time with empty load. How to reduce the overhead without affecting the hit rate of cache, and make full use of the local load to improve the cache hit rate? It is the problem to be solved in this paper.

3.2. IFTL Algorithm

When a dirty entry is dropped out of the cache, it does not exist in the global mapping table because it is a new map entry. At the situation, we need update it to the global mapping table. We look up the map page in GTD, update the map page and write back to SSD.

We give an example of a dirty entry shown as Figure 1. Mapping entry LPN=1 and PPN=260 as sacrificial entry need to write back because the entry is the latest. In the process of writing back the dirty entry, we also write back the latest mapping entries in the same mapping page, so that we merge two read and write page mapping operations into only one. For example, mapping entry LPN=3, PPN=150 can be written back with mapping entry LPN=1, PPN=260. First, we look up GTD and find the physical page number mppn=21 where two map entries load. Second, we find a free page in the translation block, the physical page number mppn=23 as figure 1 shows, and copy the old map page information to the free page at the same time. At last, we should update the information mapping entries written back and the correspondence between the virtual page number MVPN with MPPN in GTD. At some time in the future, when the mapping entry LPN=3, PPN=150 is to be driven out, it can be driven out directly without additional operation.

IFTL algorithm can effectively reduce the extra cost of read and write operation, which can effectively reduce the cost of read and write request, reduce the access number of SSD and improve the performance.

4. EXPERIMENTAL RESULTS

4.1. Algorithm Implement

Our system is realized by simulation software: disksim. Disksim is a disk system simulator, which has the advantages of high efficiency and accuracy. It can be configured with appropriate modifications that can be used conveniently in time of storage system. This is because it is written by C and open source. A large number of facts have proved that disksim can really simulate the working conditions of the storage system, so that it has
been widely used. The architecture of disksim is shown as Figure 2.

![Figure 1. IFTL Algorithm Description](image1)

![Figure 2. The architecture of disksim](image2)

We built a simulation environment based disksim to compare the different mapping table caching algorithms. Our mapping table caching algorithm is to solve the problems in the DFTL cache management algorithm, so we compare our algorithm with DFTL algorithm. In order to show the effect of the cache, the situation without cache is also compared. The environment of the hardware and software are listed as Table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R)Core(TM) i73770K <a href="mailto:CPU@3.50GHz">CPU@3.50GHz</a></td>
</tr>
<tr>
<td>Memory</td>
<td>DDR3 4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 10.04</td>
</tr>
<tr>
<td>Kernel</td>
<td>2.6.32</td>
</tr>
<tr>
<td>Disksim</td>
<td>Disksim3.0</td>
</tr>
</tbody>
</table>

**4.2. The Overhead of Read and Write Map Table**

The result is shown in Figure 3 and Figure 4. From the figure we can see that IFTL can effectively reduce the overhead of the read and write mapping table. For Financial-1, the overhead of read map table using IFTL is reduced by 84.67% than that using DFTL. The overhead of write map table is reduced by 98.3%. For Web search, IFTL relative to DFTL in the overhead of read map table is reduced 58.8%, while in the overhead of write map table is reduced 99.94%.
Figure 3. The overhead of reading map table

Figure 4. The overhead of writing map table

5. CONCLUSIONS

We have developed IFTL algorithm which can effectively reduce the overhead of read and write map table in the FTL, and improve the reading and writing performance of SSD. However, there are some problems in this algorithm. For example, when the local load is not very good, the extra space overhead caused by the introduction of the cache map page will be relatively large. These are the aspects we will improve in the future.

Acknowledgements

This work was supported by the Science and Technology Planning Project of Henan Province (No.142300410044), Colleges and University Major Scientific Project of Henan Province (No.15BS20022), and the National Natural Science Foundation of China (No.61662038), Science and technology project of Jiangxi Provincial Department of Education (No. 151081), Scientific research project of Jiujiang University (No. 2015LGYB28).

REFERENCES


