Acta Technica *62* (2017), No. 5B, 687–696 © 2017 Institute of Thermomechanics CAS, v.v.i.

Improve SSD performance by prefetch strategy

Chao Yin[2], Tong fang Li[3], Yan Liu[2], Sihao Yuan[2], Qin Zhan[2]

Abstract. SSD (Solid state disk) based flash is a new storage device in recent years. Compared with the traditional hard disk, SSD has obvious advantages in performance, energy consumption, reliability, size and so on. Now, SSD has gradually become an important storage device for portable computer systems, desktop computer systems, large server systems, high-performance computing systems. SSD has many features, such as erasion before writing, limited numbers of erasion. People should take good use of these features so that they can use SSD with the maximum effectiveness.

In order to improve the performance of SSD, it is necessary to design the software structure of SSD under certain conditions. One of the most important software structure is DFTL (demanded-based translation layer flash), which was designed from FTL (translation layer flash) and has higher performance than FTL because of using the workload locality.

DFTL only considered the time locality and ignored the spatial locality. In order to further improve the hit rate of cache, we need to fully exploit the workload, especially the spatial locality of the workload. Prefetch technology is an effective way to solve this problem. In the paper, we have proposed a new prefetch technology, called TLLLP (Two Level Linked List Prefetch), which can improve the hit rate of cache greatly. In order to make full use of the spatial locality of the workload, TLLLP determine the prefetch length of cache according to the historical mapping information, the current request length and available space information in the mapping table. At the same time, it can prefetch the mapping entries which will likely be accessed in the near future in advance into the cache to improve the hit rate of cache.

We have used four traces to evaluate our TLLLP prototype by simulation test and compared with No Cache, DFTL. The result shows that TLLLP has higher hit rate than DFTL because of prefetch technology. The performance is also significantly improved by reducing the overhead of the additional operation. The average response time of TLLLP is 12% lower than that of DFTL and The response time standard deviation of SSD in TLLLP have been reduced by an average of 15% than that of DFTL.

Key words. SSD, TLLLP, DFTL, prefetch.

[1]Acknowledgment - This work was supported by Science and technology project of Jiangxi Provincial Department of Education (GJJ151081 and GJJ161072).

[2]Workshop 1 - School of Information Science and Technology, Jiujiang University, Jiujiang 332005, China

[3]Corresponding author: Tong fang Li ;email:maggie0611@163.com

1. Introduction

With the rapid development of Internet technology, not only large companies but also small enterprises have already accumulated PB level data. It becomes more and more urgent to improve the management ability for big data because the exponentially network access increases. In cloud computing technology nowadays, a large number of applications focused on data are becoming more and more popular. A large amount of data and users put forward higher requirements to the availability and scalability of storage resources.

In recent years, flash memory technology has been greatly improved. Since the increasing of its storage density and the decreasing of the unit capacity cost, SSD based flash has received widespread attention. Compared to the disk, SSD can provide better bandwidth and lower access delay, so that it has been widely deployed in the high performance storage system. SSD has the characteristics of fast reading and writing, light weight, low energy consumption and small size, which are not available in traditional mechanical hard disk.

SSD is regarded as a block device as it provides interface like the traditional mechanical hard disk. However, internal structure of SSD is different from that of the mechanical hard disk. Unlike mechanical hard disk positioning data by the movement of the magnetic head, SSD mainly depends on a software structure FTL[1] to convert the logical address into specific physical address. FTL has three functions, that are address mapping, garbage collection and wear leveling. When there are no free blocks, it will depend on the garbage collection mechanism to release the resource. It can avoid some data to be erased repeatedly through the wear leveling.

There are many mapping strategies in SSD. Page level mapping is one of the frequently used mapping strategies which is always used in high performance storage because of its small size and flexible character. The mapping table will be a little larger if using page level mapping so that it will cost a lot. Many algorithms have been proposed, such as DFTL[2], S-FTL[3]. DFTL is a new page level mapping FTL based on the requirements. DFTL has used the locality of the workloads, 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.

DFTL only considered the time locality and ignored the spatial locality. In order to further improve the hit rate of cache, we need to fully exploit the workload, especially the spatial locality of the workload. We have developed a prefetch technology named TLLLP, which aims to improve the spatial locality greatly. In order to make full use of the spatial locality of the workload, TLLLP has used two level linked list to organize the data. After analyzing the relationship and the spatial locality of the request, TLLLP can transfer the relative request into cache. When these requests are called in the near future, they can read from the cache directly. This operation will improve the system performance greatly.

The map entries are cached in the smallest size by the segmented caching strategy of DFTL, and are divided into two sections: the hot process and the cold one. In order to improve the efficiency, we had better update the latest entries which belong to one mapping page together. This operation can avoid the extra overhead because of searching such an entry from the entire cache. BPLRU[4], PUDLRU[5] have designed two linked list to deal with the above problem effectively. In the two linked list, the first linked list is composed of mapping page nodes, and each page node maintains a second linked list composed by mapping entries in the same mapping page. Comparing with the strategy in BPLRU, the size of the minimum organization is much smaller, because the minimum organization is mapping entries in the two level linked list, while the minimum organization is mapping pages in BPLRU. In the following, we will design this two level linked list into our system.

The contributions of this paper are described as follows:

TLLLP is based on the principle of spatial locality of workload. By prefetch, the mapping entries that are likely to be accessed in the near future are also transferred to the cache. The next request from the upper layer can be hit directly in the cache, which will reduce the overhead of additional reading map tables.

TLLLP has used the two level linked lists, in which the minimum organization is mapping pages. The two level linked lists can make the layout more reasonable and improve the prefetch efficiency.

Experimental results show that TLLLP algorithm has improved a lot in the performance of the system.

The rest of this paper is organized as follows. Section 2 introduces the design and implement of TLLLP algorithm. The experimental result and evaluation are described in Section 3. Section 4 is related work. Section 5 is the conclusion.

2. The prefetch algorithm

2.1. The design of the algorithm

The mapping table caching algorithm in DFTL only considered the time locality and ignored the spatial locality. In order to solve this problem, it is necessary to use prefetch technology. In this section, we will describe the TLLLP algorithm in detail.

As mentioned earlier, prefetch strategy is used to make better use of the spatial locality of the workload, which is not considered in the original cache strategy. In order to use prefetch policy, there must be an additional cache space to do prefetch, and this extra space occupation must ensure that the performance of the cache policy does not fall. At some time, if the prefetch condition is satisfied, that is, when the prefetch operation is triggered, the cache must provide the space to store the prefetch entries. We need to take into account the prefetch condition, that is the order of the information that has been visited, the page size of the current access request, and the size of the current cache. For the current page level request, if the corresponding mapping information is not found in the cache, it is necessary to do prefetch operation. We describe the TLLLP algorithm as follows:

After the current request is divided into one or more page requests. If it is the first page request or the first page of a new page mapping request, the prefetch window is set two times the pages from the current page level request to the end page of the current request.

If the entries in current page level request is in the same mapping page as those in the last page level request, the prefetch window should be set four times the pages from the current page level request to the end page of the current request. This principle is going on.

The number of the prefetch is the minimum number between the mapping entries of the current prefetch window and those sequential requested in history.

2.2. The implementation of the algorithm

To clearly understand the problem, we have developed a model based on DFTL to perform an in-depth analysis on the overhead of address translation. Table 1 gives a list of symbols used in our analysis.

Combining with the number of page level requests for historical sequential access and the relationship among the three symbol can be derived as Equation 1.

$$prefetch\_num = min\left(fetch\_num + 1, min\_fetch\_num\right) \tag{1}$$

Table 1. Configure Information

Symbol	Description
fetch_num	the prefetch value of the precursor page level request
min_fetch_num	the prefetch value based on the current request
prefetch_num	the last prefetch value
visited_num	the visited number of the mapping entries

In addition, the numbers of prefetch mapping entries should combine with the current capacity of cache. If the cache is full, we should drive out the mapping entries in one page mapping room as far as possible. This operation can avoid extra reading and writing overhead because of driving out the entries in different mapping page.

The cache space is very limited so that only the subset of the global mapping tables are stored in the DFTL mapping cache. After the system has worked for a time without dealing with the cache, there is no extra space in the mapping table cache to store the prefetch entries. It is necessary to vacate the space for the mapping entries that may be accessed at some point in the future by driving out a portion of the mapping entries used infrequently when the cache is full.

In the TLLLP algorithm we should consider two important questions. The first one is what kind of mapping entries should be driven out. The second one is how many mapping entries should be driven out each time.

For the first question, the mapping entry to be driven out first must be the least visited mapping entry in the cache. we define the visited number of the mapping entries as visited_num. However, the mapping entry to be driven out is not necessarily the mapping entry of the smallest visited_num in the cache organization of the two level linked list. It is the integrated result of two level lists. At the first level, the mapping entry to be driven must be the least visited map page. We evaluate the least visited map page by taking the average value of visited_num in the second level list. The mapping page with the minimum value is selected in the first level list, the mapping entry with smallest value of visited_num in the second mapping list will be driven out.

It is a critical issue to determine how many mapping entries to be driven out each time. If the numbers of the mapping entries driven out each are too small, the prefetch effect is not obvious. If the numbers are too much, it is easier to driven out some entries that will be requested in the near future.

According to the discuss above, TLLLP is designed to drive in accordance with the size of the mapping page, that is, at least in the cache will be the most recent visit to the map of all the mapping entries are out. All the mapping entries in the most recently visited mapping page are driven out each time.

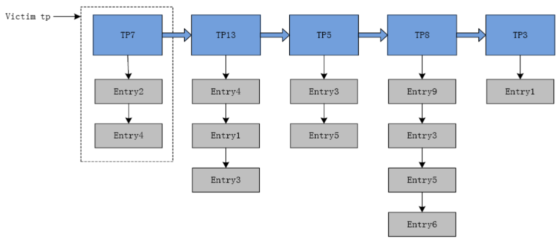

Fig. 1. The Structure of TLLLP

The first level list is arranged in an orderly fashion according to the frequency at which the mapping entries in the cache map are located, that is, the front row of the map page is rarely accessed. The structure of TLLLP is shown as Figure 1. TP7 is a mapping page, under which there are two mapping entries Entry2 and Entry4. These two entries have recently been rarely accessed. When the cache is full, we select the mapping page TP7 to be driven out according to the order, select the mapping entry all TP7 as out of the items. The system will have an idle mapping page and two mapping entries. From Figure 1 we can see that the vacated space is the sacrificed mapping page that the actual prefetch needs.

This scheme is convenient for prefetch and implementation easily. It can obtain a larger cache space at one prefetch operation. As a result of the removal of a number of mapping entries, some hot items may be driven out of the cache. The next time you visit it, it will bring extra reading and writing overhead.

3. Experimental results

3.1. *Geometry of the plate*

We have tested our system by three classes of traces. The first class of traces is collected from Finance. Financial 1 is based on small write request, while Financial 2 is mostly small read request. The second class of traces is Websearch. The last class of traces is MSR (Microsoft server).

3.2. *Experimental tests*

3.2.1. Geometrical dimensions We have used the prefetch strategy to improve the hit rate. Through effective prefetch algorithm, the mapping entries likely to be accessed will be transferred into cache to improve the cache hit rate in the near future.

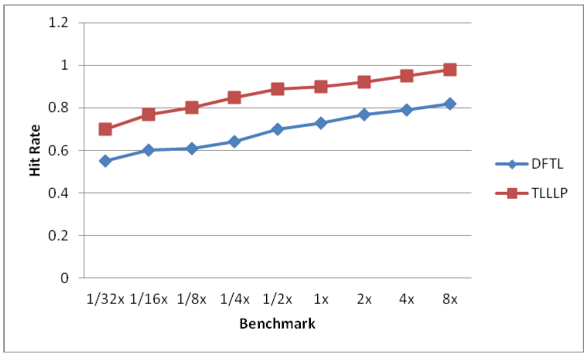

Fig. 2. The Structure of Financial 1, Financial 2, Websearch and MSR

From Figure 2, we can see that the hit rate of TLLLP is higher than that of DFTL in the case of the size of the mapping table cache changing. The Websearch has the most significant increase in hit rate. Based on the analysis of websearch, we find that its temporal locality is very poor. For the cache algorithm only considering temporal locality of the workload as DFTL, the hit rate is very low. We have full consideration of the temporal and spatial locality of the workload in TLLLP, so that the hit rate of TLLLP is greater than that of DFTL. The hit rate of DFTL is very high when using MSR, which is because the locality of MSR is better and the average length of request is larger. We can get more mapping entries into cache by prefetch algorithm.

3.2.2. The performance comparison In view of the impact of TLLLP on SSD performance, we have considered not only the hit rate of mapping table cache, but also two other indicators: the average response time of SSD and the standard deviation of response time. The average response time of SSD is the mean time from the time that the request reaches the SSD to the time that the request have been deal with. The standard deviation of the response time measures the change of the response time of SSD, which represents the discrepancy of the response time of SSD with the average response time of SSD. The smaller the value, the higher the consistency of the SSD quality of service.

We have tested the effect of several different mapping table caching algorithms on the average response time of SSD in the case of four cache workloads shown as Figure 3. From the analysis of the figure, it can be seen that the average response time of TLLLP is 12% lower than that of DFTL. It shows that the algorithm improving the average response time of SSD is no good as improving hit rate. This is because the time in garbage is a large proportion of the response time of the SSD, and TLLLP and DFTL algorithms have used the same garbage collection algorithm.

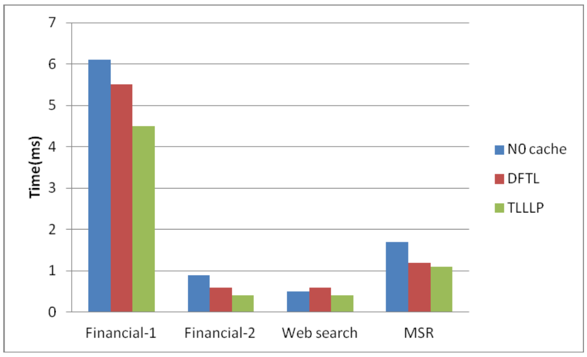

Fig. 3. Average Response Time

In these four workloads, the average response time of the Financial 1 is the longest. This workload is a small write type, which will bring more page-level writes. The response time of TLLLP is the shortest, which can reduced by 16.64% than that of DFTL. This is because the mapping table cache will create a large number of dirty entries in the small write workload, and TLLLP has greatly reduced the read and write mapping.

We used the four standard workloads in several cache algorithms to test the response time standard deviation of SSD, which is shown in Figure 4. The response time standard deviation of SSD in TLLLP have been reduced by an average of 15% than that of DFTL except in the workload of Financial 1. For read-based workload, TLLLP can improve the hit rate and reduce the operation to the flash mapping table. To write-based trace, especially for workloads with a large proportion of writes, the difference in cost is relatively large between write and read. In the case of large number of write hit rate, such as Financial 1, which is write-based. The time cost is very large among the operations, such as read operations, big write operation and small write operation, so that its standard response time is greater than that of DFTL by 10%. However, for the average response time reduction of 16.64%, the appropriate increase in the standard deviation is acceptable, especially for this type of workload.

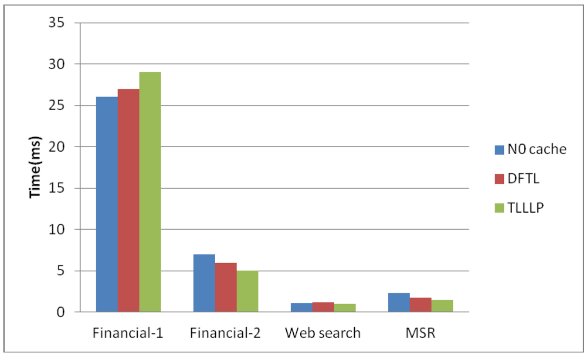

Fig. 4. SSD Response Time Standard Deviation

4. Related works

The data which has not yet been accessed but will be accessed in the near future, will transferred from the low-speed storage device to cache devices to improve the speed of data access and the performance of the whole storage system. These operations are named prefetch [6][7][8]. From the above definition, we can see that the two key problems need to be solved by using prefetch technology. The first one is the expected precision which is an important index to the cache hit rate. If the data prefetched into high-speed device data is rarely accessed in the near future, it means the prefetch accuracy very low. It is not only a waste of bandwidth, what's more, it is a waste of the valuable space. The second one is the mining of continuity of the prefetch data. There are two aspects in the sequential mining, the first aspect is to search the order of the historical data to predict the length of the future. A large number of studies have shown that the longer a sequential access stream has been accessed, the more likely it is to access the next sequential data. Another aspect is to fully exploit the sequential access in the current access to improve the cache hit rate.

A very important factor in sequential prefetch technology [9] is the length information which is a quantitative index of prefetch. The difference between the random request and the sequential request is enormous to the traditional mechanical hard disks. The traditional prefetch strategy [10] [11]always puts the entry which will be accessed in the near future into the cache to optimize the operational speed. In the actual design process, we need to take good use of the length of sequential prefetch, the length of the current request and the buffer size left. We must find out a balance point among these information to avoid excessive prefetch. We also should prevent some hot items being driven out because of getting rid of the entries in the cache frequently.

5. Conclusion

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.

References

[1] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, L. Wang: *Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation.* In Proceedings of Mass Storage Systems and Technologies (2010).

[2] A. Gupta, Y. Kim, B. Urgaonkar: *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings.* In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating System (2009), 229–240.

[3] S. Jiang, L. Zhang, X. Yuan, H. Hu, Y. Chen: *S-FTL: an efficient address translation for flash memory by exploiting spatial locality.* In Proceedings of Mass Storage Systems and Technologies (MSST) (2011).

[4] H. Kim, S. Ahn: *A Buffer Management Scheme for Improving Random Writes in Flash Storage. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies.* San Jose (2008).

[5] J. Hu, H. Jiang, L. Tian: *PUD-LRU: An Erase-Efficient Write Buffer Management Algorithm for Flash Memory SSD.* Proceedings of the 18th Annual meeting of IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (2010).

[6] S. Park, D. Jung, J. Kang: *A Replacement Algorithm for Flash Memory. In: Proceedings of the 4th International Conference on Compilers.* Architectures, and Synthesis for Embedded Systems (2009).

[7] G. Graefe: *The Five-minute Rule Twenty Years Later and How Flash Memory Changes the Rules.* In: Proceedings of 3rd international workshop on Data Mangement on New Hardware (2007).

[8] F. Chen, T. Luo, X. Zhang: *a content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives.* In Proceedings of the 9th USENIX Conference on File and Stroage Technologies (2011), 77–90.

Received November 16, 2017