Hot Data Detector based High-Performance NAND and PRAM Hybrid Storage

Yu Zhang, Yizhi Wu, Demin Li, Haixiao Shan
School of Information Science & Technology Donghua University, Shanghai, China
yz_wu@dhu.eud.cn

Abstract - The high-performance and low cost NAND Flash and PRAM (Phase-changed RAM) hybrid storage designs have been proposed for embedded systems. However, the system is still suffering from serious performance degradation for continuous write/read operations in the same logical address frequently due to physical constraints of NAND Flash: erase-before-program and different unit size of erase and program operation. To deal with these constraints, a hot data detector is added in the FTL (Flash Translation Layer) of the NAND flash and PRAM hybrid architecture. In this paper, a circular queue based hot data detector algorithm (CQHDD) is proposed, which mainly includes hot-area counting queue design, queue update algorithm and hot-area counting algorithm. Comparing to existing methods, the simulation results show that the read and the write performance of the suggested method improve by 14.2% and 23.7%, respectively, and the average recognition missing rate decreases by 6.3%.

Index Terms - NAND flash; Phase-changed （PRAM）; Hot data detection; Hybrid Storage; Flash Translation Layer

I. Introduction

NAND flash-based storage is widely used due to its various advantages, including low power consumption, a small footprint, high density, non-volatility and so on. However, flash memory devices have some physical constraints. For example, the erase operation is required along with the write operation, and each flash memory block suffers from limited erase operations (10,000-100,000). In addition, flash memory devices are accessed by page, for example with 2KB, but the erase operation is run by block, usually 128KB. To overcome these constraints, FTL (Flash Translation Layer) [1] is used to abstract the NAND Flash memory into an over-writable block device like HDDs (Hard Disk Drives) using special writing mechanism. It works as this: the write operations of a particular physical page are not performed directly to their original page, but to another buffer page until the buffer page is full, then the full buffer page is combined with the original page and released for later buffer page request. This writing mechanism is largely cost consuming so that how to decrease the system overhead has already become a hot research area [2][3][4].

Recently, PRAM (Phase-change RAM)[5][6] which supports byte-addressable interface and in-place update without erase-before-program constraint, is introduced as a next generation non-volatile memory. However, in current technology, its density and write latency are still obstacles from completely replacing the NAND Flash for the bulk data storage. So many studies have proposed hybrid storage solution by using PRAM devices as the write buffer storage for NAND Flash as shown in Fig. 1. The proposed hybrid architecture combines the advantage of NAND Flash and PRAM and improves the storage system performance effectively.

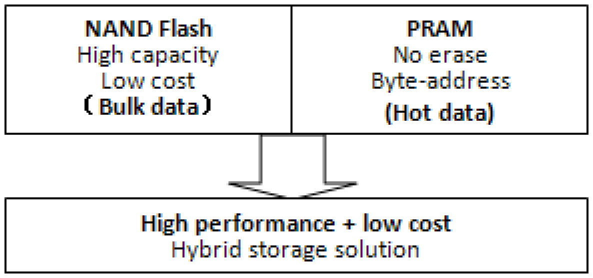


Figure 1. The proposed hybrid architecture

Although the combination operations of buffer pages has decreased and the efficiency has increased in hybrid storage because of the randomly access characteristics of PRAM, the limited capacity of the off-the-shelf PRAM devices can only satisfy part of the writing buffer requests and make it a key problem to choose which pages to be buffered in PRAM. Hot data is the storage pages which access much more intensively than other pages according to the time and space locality of program execution. In this paper, a hot data detection layer is added on FTL based on the PRAM+ NAND flash hybrid architecture. The hot data detection is used to detect hot data and then place this hot data on PRAM. Since the PRAM provides not only better performance than NAND Flash, but also byte-level access, storing frequently accessed sectors (hot data) into the PRAM may increase the performance of the storage significantly.

Our contributions in this paper are summarized as follows.

1. We propose a circular queue based hot data detector (CQHDD) to deal with hot data tracking and counting effectively, which in turn increases the performance of PRAM+NAND hybrid storage system.
2. We give the detailed design of the CQHDD including circular queue structure, queue update algorithm and hot-area counting algorithm.
3. We implement CQHDD and evaluation shows that the proposed method improve the efficiency of the hot data detector by 35.8 %.

The rest of the paper is organized as follows. Section 2 introduces the most common hot recognition algorithms and conventional architecture of NAND Flash-based storage.

*This work was partially supported by the of National Natural Science Foundation of China (No. 71171045/G0104) to Mr. Li.

Section 3 describes hybrid storage architecture and introduces the components of CQHDD. Section 4 presents the simulation and detailed experimentation results. Finally, additional remarks and conclusion about CQHDD are given in Sections 5.

II. Related Work

A. Hot data detection

Many researchers have proposed methods of the identification of hot data. Ref. [4] proposes a hot data identification method, direct address method (DAM), in which a hot data counter is provided for each LBA (Logical Block Address) that is accessed by host. When a read/ write request comes, the relevant hot data counter plus one. But this hot data recognition algorithm is less efficient because it occupies too much memory. However, it reflects the frequency of an address is accessed most completely and we can use it as a baseline to evaluate the other hot data recognition methods. The two-level LRU [7] maintains two fixed length queue, one is used to store LBA that may become hot data, another is used to store hot data. The records in the two queues are eliminated by LRU (least recently used) algorithm. The multi-hash-function (MHF) framework [8] uses *k* independent hash functions to map the given LBA to a M-dimensional hash table, and each entity occupies a *c*-bit counter to record the number of visits of the LBA. However, both of the methods require considerable computing overheads and most of them do not reflect the relationship of hot data variation with the time commendably.

B. Conventional architecture of NAND Flash-based storage

Fig. 2 shows the conventional architecture of NAND Flash-based storage which is mainly consisted of file system and FTL. The main function of file system is to organize and manage files. To hide the constraints of NAND Flash and to mimic HDD-like storage, FTL should do remapping between logical address space and physical address space and hide erase operations of a NAND Flash-based storage. Because that FTL encapsulate the NAND Flash with the interface of the existing HDD-like storage device, it is convenient for the existing file system only needs further measure to improve the performance, which is the main focus of this paper. LLD layer is mainly used to directly drive controller to finish the concrete operation of the bottom, such as the basic read, write and erase operations.

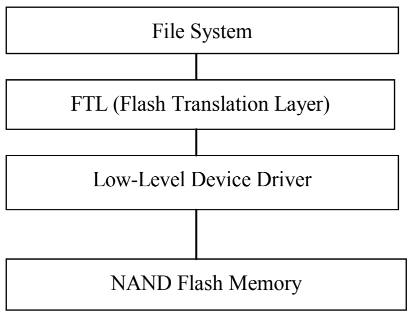


Figure 2. The architecture of general NAND storage

III. The Hot Data Detector

A. Hybrid stroage system with the hot data detector

In order to effectively detect hot data, in this paper, a hot data detector is added as an additional layer between the file system and FTL, as shown in Fig. 3. In the hot data detector based hybrid memory storage system, PRAM is used to store the hot data that detected by the recognition algorithm and the NAND Flash is used to store cold data. Two address mapping tables are maintained in the storage system. A mapping table is used to record the mapping information of the data in PRAM, and another mapping table, mapping information of the data in NAND Flash. Two mapping tables are stored in the PRAM. The hot data recognition algorithm is realized in the hot data detector, in which a hot data ranking list calculated every fixed period of time according to monitored I/O operations issued by the operating system. Then FTL will allocate the hot data detected into PRAM and cold data into NAND flash respectively, at the same time, the FTL will also update the PRAM and NAND Flash mapping tables.

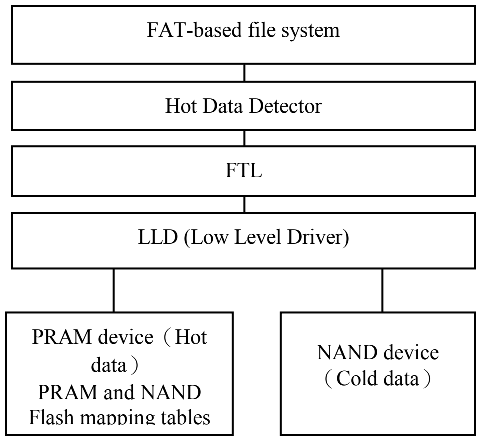


Figure 3. Hybrid storage architecture with the hot data detector

B. The Circular Queue based Hot Data Detector

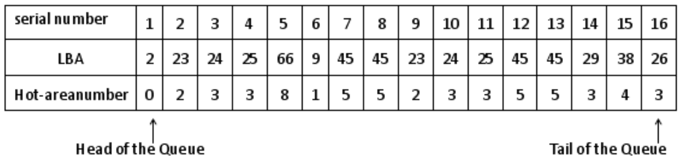
In this paper, we put forward for the first time a hot data detection algorithm basing on a circular queue, i.e. Circular Queue based Hot Data Detector(CQHDD), to identify the hot data and reflect the relationship of hot data variation within each period of the time effectively. The CQHDD algorithm establishes a queue with fixed length, *n*, a queue head pointer(hP) and tail pointer (tP). Initially both the hP and the tP point to the same position. When a read/write request arrives, the related LBA is added to the position the tP is pointing and the tP moves forward. When the tP meets the hP again, the hP needs to move forward one position, ant that means a LBA record is deleted. To determine whether a LBA is hot data, we only need to scan each element of the queue, and count the number of the LBA appeared in the queue every detecting period. The circular queue based design increases the implementation efficiency because that shifting operations required by general queue is not needed.

Furthermore, this calculation method can calculate the accessing frequency of each LBA in a period of time, but it will consume large amounts of memory. For a NAND flash that has 2 GB address space and 2KB page size, not only at least 22-bit is needed for a single LBA to store the access address, but also a large number of records is need to store the access information. For the NAND Flash with greater storage capacity, much more storage space would be consumed by CQHDD. In the practical application execution situation, if a LBA is accessed recently, the same LBA or its nearby address will be accessed again in a short period of time [5]. Traces collected from daily usage of computers show that more than two-thirds of write operations are of a size of no more than 8 sectors, and these writes only touch about 1% of the address space of the disk [6]. Meanwhile, due to the inherent characteristics of the flash memory that the unit of write is page, therefore, if we assume a particular LBA is hot data, then data in the page (usually 8 sectors) that LBA belongs are hot data too. So, we set the unit of hot-area as 8 sectors and each hot-area has a hot-area number. For example, we assume the LBA of a I/O request is *x*. If a I/O request issued to the FTL, firstly, the hot-area unit number (hot-area number =[*x* / 8]) is calculated, then the hot-area number is stored in head of the queue. Because every 8 LBA takes up a queue element position, memory consumption needed by the hot data detector is largely decreased.

C. The Hot Data Detector Algorithms

To determine whether a LBA is hot data, the hot-area unit number mentioned above should be calculated firstly. Then each element of the queue is scaned and the times of the hot-area number appeared in the queue is counted. If the times is more than a certain threshold, which means the hot-area has been visited very frequently in recent period of time, the LBAs in hot-area can be defined as hot data. Queue's length (*n*) determines accuracy of hot-data identification, and we know that the larger of value *n*, the higher of the accuracy, but it consumes more memory space.

The illustration example is shown as in Fig. 4, in which the first line is I/O operation sequence number, the second line is the LBA of the IO operations, and the third line is corresponding hot-area number of each LBA. We assume that queue length, *n,* is 16, and define the threshold of hot-area occurrence is 4. We can see that hot-area number 3 and 6 are hot-areas because that hot-area number 3 appears 6 times and hot-area number 5 appears 4 times. LBA (24 to 31, 40 to 47) corresponding hot-area number 3 and 6 are identified as hot data.



serial number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LBA	2	23	24	25	66	9	45	45	23	24	25	45	45	29	38	26
Hot-areanumber	0	2	3	3	8	1	5	5	2	3	3	5	5	3	4	3

Head of the Queue
Tail of the Queue

Figure 4. Illustration of circular queue of the hot data detector

The figure 5 shows queue update algorithm which is called by the hot data detector whenever a read or read operation is coming.

```
Algorithm 1 Queue Update
Input: LBA //The LBA of write or read operation issued by host
Output: que // The queue
#define MAXSIZE N
typedef int Elemtype;
typedef struct
{ Elemtype *base;
  int front ; // Head pointer of the queue
  int rear; // Tail pointer of the queue
}SqQueue;
SqQueue *que;
que=NULL;
InitQueue(que); //Initialize the queue
hot-area number =[X / 8];// Calculate hot-area unit number
EnQueue(que, hot-area number); // join hot-area number to the queue
If (Q.rear+1)%MAXQSIZE=Q.front // If the queue is full
{DeQueue(que, Elemtype *s); Delete the first element of the queue }
Return * que;
```

Figure 5. Queue update algorithm

The figure 6 shows hot-area counting algorithm. It is called every fixed hot-area counting period.

```
Algorithm 2 Hot-data Counting
Input: * que // The queue
Output: hot-area number
Threshold: Criterion represents judgment threshold of hot data
Length: Length represents the length of the queue
Hot-data-detector ( * que )
for 1 : Length // Scan the queue
    hot-area-num-count ← The number of occurrences of each hot-area in queue
    if hot-area-num-count >= Threshold
   hot-data ← LBA corresponding to hot-area-num-count
    end
end
return hot-data
```

Figure 6. Hot-area counting algorithm

IV. Simulation Result

To evaluate the performance of the proposed software architecture, the hybrid storage system simulator is designed. In hot-data counting algorithm (shown in Figure 6), we set *Lengh* = 1000, and threshold that identify hot data is set to 10.

The experiments are based on a set of realistic I/O traces collected by Bates[9] and Diskmon running on a laptop (Fujitsu Lifebook V with Intel Pentium Dual Core 2GHz processor and 2GB DRAM) [8]. The characteristics of trace files are as shown in Table 1. Due to Finanicial1 and Finanicial2 file collected by Bates are too large, so only capture the part of the data, and the four files detected by Diskmon are running on the applications of word and MATLAB. Because that the direct address method (DAM), introduced in Section II, sets up a counter for each of the LBA, and there is no hot data identification missing. Therefore, we use the results of the DAM as the baseline, and define hot area recognition missing rate as the ratio of the mismatched hot areas recognized by an algorithm with the results of the DAM.

Figure 7 shows the comparison of recognition missing rate of the MHF algorithm and CQHDD algorithm in the 7 test file respectively. The recognition missing rate of hot area resulted by CQHDD is decreased largest by 11.2% than that by the

MHF algorithm in traces of Finanicial2 and averagely by 6.3% among the 7 trace files.

TABLE I. TRACEFILE DESCRIPTION

File	Total request	Write	Read
Finanicial1	115077	63796	51281
Finanicial2	103430	17092	86338
Diskmon1	4971	2085	2886
Diskmon2	8592	7257	1355
Diskmon3	7728	6207	1521
Diskmon4	7964	6609	1355
Diskmon5	6185	5923	262

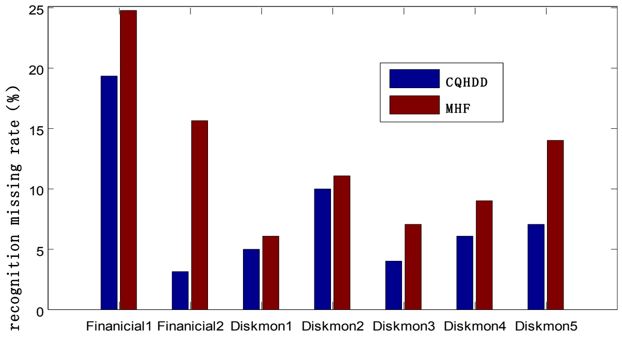


Figure 7. comparation of hot data detector algotithm

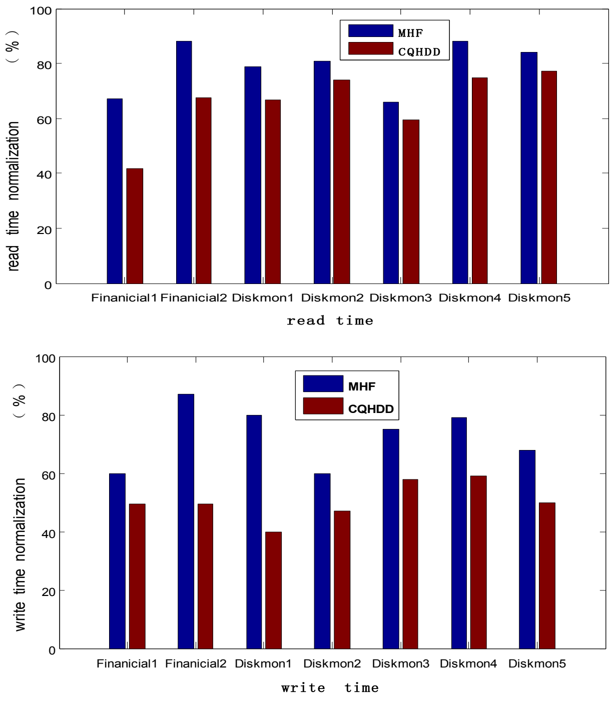


Figure 8. Read and Write performance

Figure 8 shows the read and write performance evaluations of the proposed CQHDD. Comparing with the MHF algorithm, the proposed CQHDD based hybrid storage reduces the read and write time averagely by 14.2% and 23.7% respectively. Due to no need of shifting operation required by general queue, the circular queue based design also increases the implementation efficiency. The evaluation shows that the proposed method improved the efficiency of the hot data detector by 35.8%.

The experiments show that the proposed CQHDD based hybrid storage has higher performance in read, write, and recognition missing rate than the MHF algorithm.

V. Conclusions

In this paper, PRAM was used to complement the performance loss which comes from physical constraints of NAND-flash-memory-based storages. To do that, a hot data detection algorithm called CQHDD algorithm was suggested in order to identify hot data based on a hot data queue. And then we described that how to maintain mapping table and conduct the read/write operations. Finally, the experimental results show that our proposed CQHDD algorithm enhances the read and write performance of the storage system by 14.2 % and 23.7 % respectively comparison with MHF in hybrid structure that with hot detector.

References

[1] Intel Corp., “Understanding the flash translation layer (FTL) specification,” 1998.

[2] L.-P. Chang, T.-W. Kuo, and S.-W. Lo. Real-Time Garbage Collection for Flash-Memory Storage Systems of Real-Time Embedded Systems. ACM Transactions on Embedded Computing Systems,3(4):837{863, Nov. 2004.

[3] Y. W. Park, S. H. Lim, C. Lee and K. H. Park "PFFS: a scalable flash memory file system for the hybrid architecture of phasechange RAM and NAND flash," Proceedings of the 2008 ACM symposium on Applied computing, pp. 1498-1503, March 2008.

[4] lung Sik Park, Hi-Seok Kim, “PRAM and NAND Flash Hybrid Architecture based on Hot Data Detection,” International Conference on Mechanical and Electronics Engineering (ICMEE 2010)

[5] Y. Xie, “Modeling, architecture, and applications for emerging memory technologies,” IEEE Design Test of Computers, vol. 28, no. 1, pp. 44–51, Jan.–Feb. 2011.

[6] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, “Emerging Non-volatile memories: opportunities and challenges,” in CODES-ISSS, 2011.

[7] CHANG Li-pin, KUO Tei-wei. An adaptive striping architecture for flash memory storage systems of embedded systems / / Proc of the 8th Real-Time and Embedded Technology and Applications Symposium. Washington DC: IEEE Computer Society, 2002: 187-196.

[8] HSIEH J W, KUO Tei-wei, CHANG Li-pin. Efficient Identification of hot data for flash memory storage systems. ACM Trans on Storage, 2006, 2(1) : 22-40.

[9] http://traces.cs.umass.edu/index.php/Storage/Storage.