International Journal of Control and Automation
Vol. 7, No. 11 (2014), pp. 301-310
http://dx.doi.org/10.14257/ijca.2014.7.11.30

Storage Management Method for Hybrid Storage System to Reduce Extent Fragmentation

In-Soo Bae[1], Yun-Su Lee[1], Seung-Kook Cheong[2], Yunsik Kwak[1] and Seokil Song[1]

[1]Department of Computer Engineering, Korea National University of Transportation,
Chungju, Chungbuk, Korea
[2]Electronics and Telecommunications Research Institute, Daejeon, 305-700, Korea
[1]{gkdrmf23, yunsou.lee}@gmail.com, {yskwak, sisong}@ut.ac.kr
[2]skjeong@etri.re.kr

Abstract

A hybrid storage system consists of a small and fast solid state drive (SSD) and one or more slow and large hard disk drives (HDD). The purpose of a hybrid storage sys-tem is to exploit the IO performance of SSDs to enhance the overall IO performance with comparatively low cost. In a hybrid storage system, since the size of a SSD is small, the storage utilization for the SSD is very important to guarantee the performance of a hybrid storage system. Some existing hybrid storages use extents to manage a large storage efficiently. However, they cause an extent fragmentation problem. The extent fragmentation reduces the SSD utilization and the overall performance of a hybrid storage system. In this paper, to solve the problem we propose a new mapping method that uses different size of IO unit for SSD and HDDs. Also, we perform simulations to show that our proposed method guarantees better SSD utilization than existing methods.

Keywords: *hybrid storage, DRAM SSD, mapping method, fragmentation*

1. Introduction

Some hybrid storage systems [1-4] that combine a small SSD with HDDs have been proposed to exploit the IO performance of SSDs to enhance the overall IO performance with comparatively low cost. Usually, a hybrid storage system consists of a small and fast solid state drive (SSD) and one or more slow and large hard disk drives (HDD). For a past decade, SSDs have been becoming cost effective and their performance, also, have been dramatically enhanced. However, they are not still ready to re-place HDDs in enterprise environment [xx]. Therefore, it is certainly reasonable to com-bine a small SSD with HDDs to exploit the IO performance of SSDs to boost the overall IO performance with comparatively low cost.

Existing hybrid storage systems may be grouped into two approaches according to their organization way of SSDs and HDDs [8]. An approach is to organize hierarchically SSDs and HDDs to exploit the locality of data accesses. In this approach, data is cached and accessed from SSDs while HDDs provide data to the faster storage media. Data blocks are moved between the different layers of the storage hierarchy based on the data access characteristics. Traditionally, storage systems that employ faster storage media as caches generally improve performance while hiding the complexity of handling the diversity of multiple storage media with different characteristics to the upper layers. This kind of hybrid storage system is proposed in [2].

The other approach is to organize SSDs and HDDs in a different way which accompanies block migration are proposed in [3-5]. In such systems, the frequently accessed data blocks are stored on the faster storage media to improve performance while realizing the combined capacity of the devices. Also, these architectures can be organized as a hierarchical architecture with SSDs at higher layers being accessed first.

HCMSS (Hot Cold Management Storage System) [8] is one of the recently proposed hybrid storage system. It organizes a DDR-SSDs [6] and HDDs on the same layer. Frequently accessed data are placed on SSDs and data blocks are migrated between SSDs and HDDs according to their access frequency and recency. The mapping manager of HCMSS manages storage space in extents. When HCMSS receives IO requests from VFS, and it maps the IO requests to a physical extents in a SSD or a HDD. Sometimes, it allocates new extents from a SSD or a HDD, or migrates extents in a device to the other device by considering their temperature. An extent size is a multiple of the page size and it is given when creating a hybrid block device. Extents may decrease the management cost (the size of main memory for mapping and free space management) of HCMSS, and increase the IO performance of HDDs.

However, the extent mapping method may cause an extent fragmentation problem. In Figure 1, the extent fragmentation problem is described. In this figure, we assume that the block size is 4kbytes and the extent size is 64kbyte (16 blocks). As shown in this figure, only 4 blocks in extent 0 are used, *i.e.*, only 4 blocks have valid data, and we call this extent as a fragmented extent. There are 4 fragmented extents such as extent 0, 1, 3 and 4 in this figure, and 40% of the SSD is not used even though the SSD is full. Consequently, a new write request from VFS cannot be processed in the SSD.

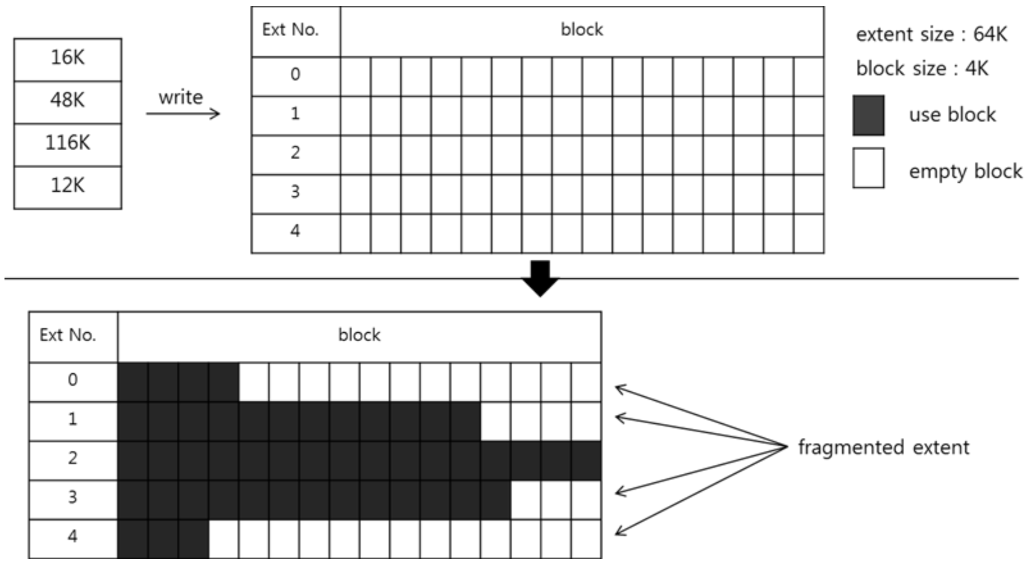


Figure 1. Extent Fragmentation Problem

The extent fragmentation problem arises by random writes of small files, and it is becoming more serious when small files are created and deleted repeatedly. Since we assume that the size of HDD is very large, the fragmentation problem of HDDs may not impact the overall performance of HCMSS. However, the problem of SSDs which is relatively small is critical for the performance of HCMSS. The extent fragmentation reduces the SSD utilization and the overall performance of the HCMSS.

The extent fragmentation problem of HCMSS causes inefficiency of extent migration operations. In Figure 2, extent 2, 3, 4 are migrated from a HDD to a SSD. The blocks of extent 2 are fully used, so the migration of extent 2 is not a problem. However, some blocks of the extent 3 and 4 are not used. Even, in extent 4, only 3 blocks are used. If we can migrate used blocks of extent 3 and 4, the bandwidth of the SSD is not wasted. However, HCMSS only uses extent based mapping method, it is impossible to migrate used blocks of extent 3 and 4.

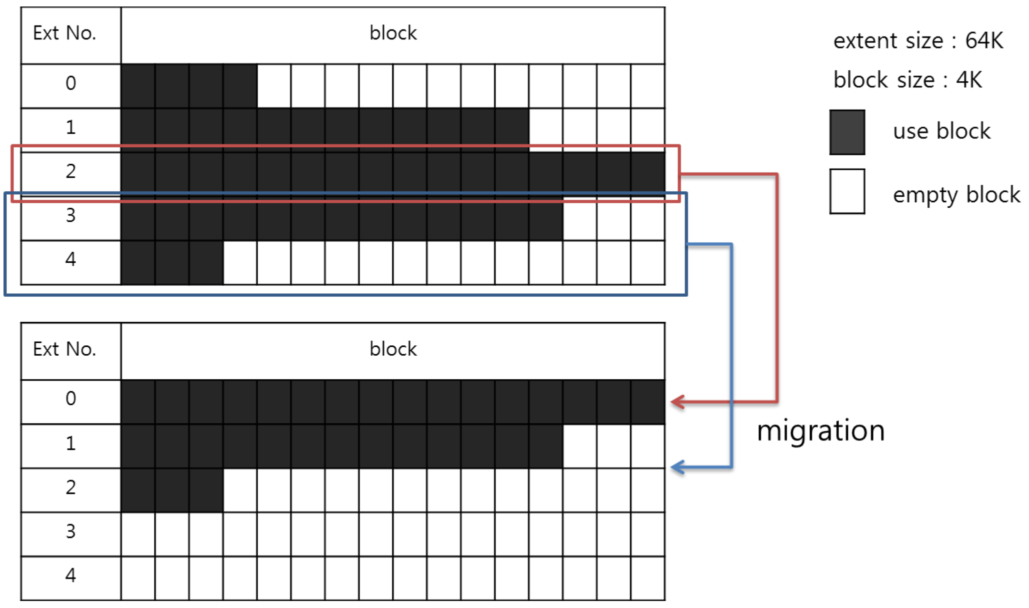


Figure 2. Ineffectiveness of Migration Operation in HCMSS

In this paper, we propose a new mapping table for the HCMSS to solve the extent fragmentation problem of SSD. We propose a hybrid mapping method that uses different size of IO unit for SSD and HDDs. In our proposed mapping method, block based mapping is used for SSDs and extent based mapping is used for HDDs. Because the mapping units of both devices are different, we need to maintain the status of each block in an extent to migrate only used blocks from a HDD to a SSD. We perform simulations to verify our proposed method.

This paper is organized as follows. In Section 2, we describe related work, and in Section 3, the proposed mapping method is presented. Then, Section 4 gives the experimental results, and finally we conclude our paper in Section 5.

2. Related Work

SSDAlloc [2] is a hybrid main memory management system. It virtualizes SSD as an extension of the RAM in a system so as to make application programmers easy to use. Instead of using the SSD as a cache for the hard drive, it moves the SSD upward in the memory hierarchy. Consequently, the SSD is treated as a larger and slower form of RAM.

The hybrid storage system of [3] utilizes both initial block allocation as well as migration to reach “Wardrop equilibrium”, in which the response times of different devices equalize. The hybrid storage system of [4] consists of the flash-SSD and HDD at the same level of the memory hierarchy, and places a data block to only one of these disks according to the workload of the block. Blocks with a read-intensive workload are placed on the flash disk, while blocks with a write-intensive workload are placed on the HDDs.

[5] proposes a hybrid copy-on-write storage system that combines flash-SSD and HDD for consolidated environments. In order to take advantage of both devices, the

proposed scheme places a read-only template disk image on a SSD, while write operations are isolated to the HDD. In this hybrid architecture, the disk I/O performance benefits from the fast read access of the solid-state disk, especially for random reads, precluding write operations from the degrading flash memory performance. [7] Proposes a buffer management method for hybrid main memory and flash memory storages.

[8] proposes HCMSS (Hot Cold Management Storage System) that combines a DDR-SSD and HDDs on the same level of the memory hierarchy. Since our proposed method is to enhance the mapping method of [8], we describe HCMSS in more detail. Figure 3 shows the architecture of HCMSS of [8]. The HCMSS is working between virtual file system (VFS) and the device drivers as shown in the figure. The hybrid storage system creates a hybrid block device with a HDD and a DDR-SSD. It manages storage space in extents and the size of an extent is a multiple of the page size.

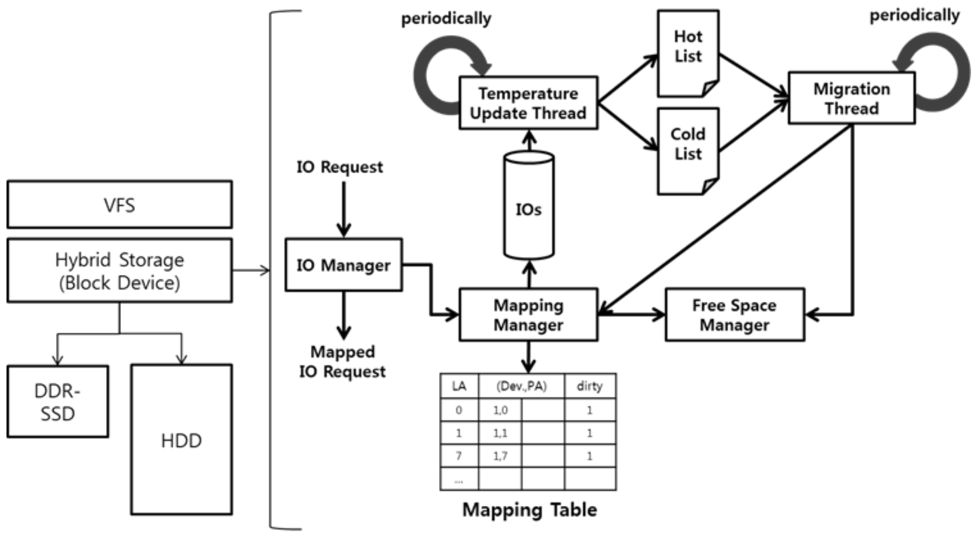


Figure 3. Architecture of HCMSS [x]

The hybrid block device receives an IO request from VFS, and maps the IO request to a physical address in a SSD or a HDD. The physical address is the first block address of an extent including the IO request. Sometimes, it allocates new extents from a SSD or a HDD, or migrates extents in a device to the other device by considering performance issues and capacity according to its migration policy.

HCMSS has a mapping table to allow indirect addressing between the logical address from VFS and the actual physical address on the device where extent places. This map-ping table allows integrating a SSD and HDDs as hybrid storage. The mapping manager manages this mapping table. It maps a logical block to a physical extent by using map-ping table, and updates the mapping table when a new extent is allocated or an extent migrates between devices. Temperature thread periodically calculates the temperature of each extent according to our temperature calculation method and produces a hot extent list and a cold extent list.

Migration thread determines whether migrating an extent or not by analysis the hot and cold lists. The free space manager manages free blocks of a SSD and HDDs, and allocates new blocks from a SSD or a HDD. IO manager performs physical IOs. In the following, we describe the core parts of our hybrid storage system.

As described earlier in Section 1, HCMSS has the extent fragmentation problem. In order to achieve the management efficiency such as unified IO model for SSDs and HDDs, IO performance of HDDs and small size of meta-data for mapping table and free space management, HCMSS maps logical address from VFS to a physical extent. Besides mapping management, free space management, temperature management and migration management, also, use extent approach.

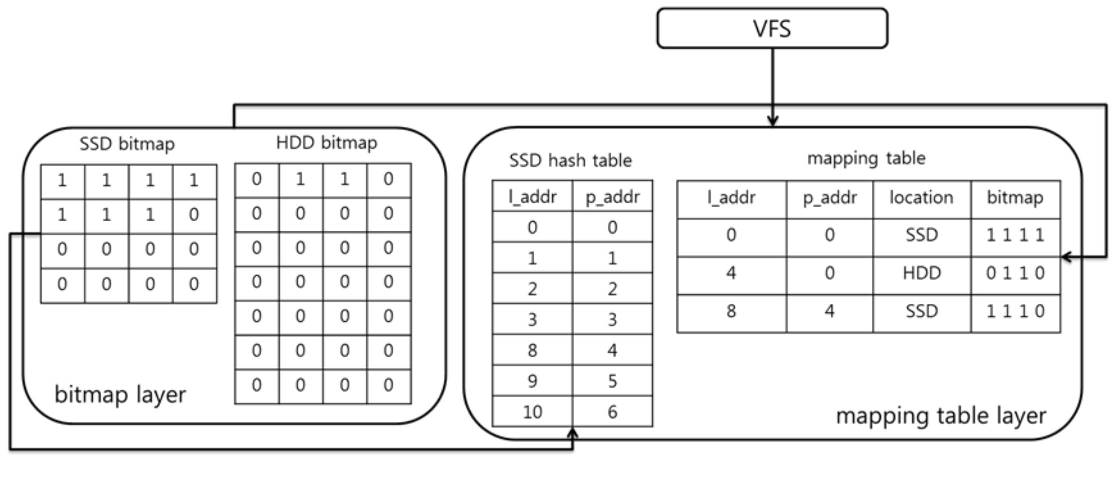
In this paper, we propose a hybrid mapping method for HCMSS, which uses different units such as block and extent for SSD and HDD respectively to reduce the extent frag-mentation problem in SSD. We assume that the size of HDDs can be very large in HCMSS so the extent fragmentation in HDDs is not severe. Actually, in HDDs, extent based IO is much better for performance.

3. Proposed Mapping Method

3.1. Overview

Figure 4 shows that the architecture of the proposed mapping method. Basically, the proposed mapping method is an enhanced version of HCMSS's mapping method. Un-like the HCMSS's mapping method, we add a block level mapping table and a free space bitmap for SSD, and also we change the structure of the original mapping table by adding a bitmap field. The bitmap field of the mapping table is for indicating that each block of an extent is used or not. The HDD bitmap indicates that each extent is used or not. On the other hand, the SSD bitmap indicates that each block of SSD is used or not. While the mapping table maps block IO requests that contains logical block number to physical extents in HDDs, the SSD hash table maps block IO requests to physical blocks in SSD.

In Figure 4, the block size is 4kbytes and the extent size is 16kbytes, so the bitmap field of the mapping table has 4 bits for the corresponding 4 blocks of an extent. The SSD contains 16 blocks, and the HDD have 28 extents (112 blocks). The logical extents 0 and 1 are placed on the SSD, and the logical extent 5 is placed on the HDD. For the extents 0 and 1 on the SSD, we manage block level mapping information in the SSD hash table.



SSD bitmap:

1	1	1	1
1	1	1	0
0	0	0	0
0	0	0	0

HDD bitmap:

0	1	1	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

SSD hash table:

l_addr	p_addr
0	0
1	1
2	2
3	3
8	4
9	5
10	6

mapping table:

l_addr	p_addr	location	bitmap
0	0	SSD	1 1 1 1
4	0	HDD	0 1 1 0
8	4	SSD	1 1 1 0

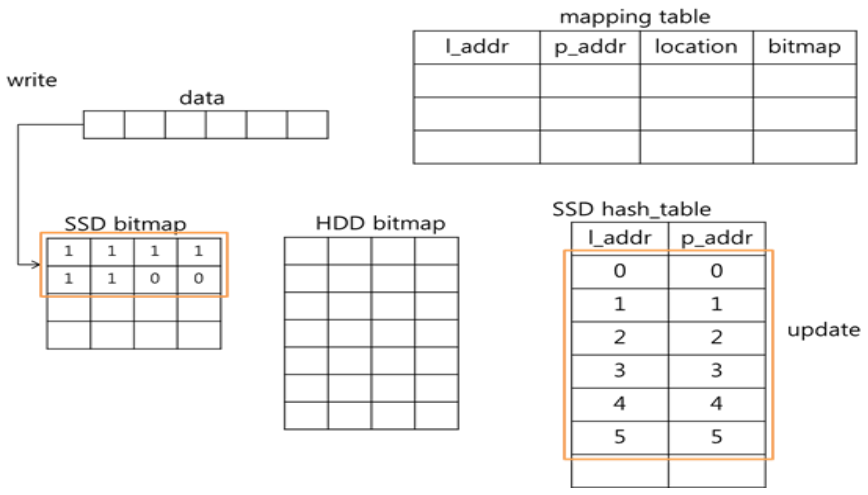
Figure 4. Architecture of the Proposed Mapping Method

When a block request is given by VFS, our mapping manager checks that the block is on the SSD or the HDD by searching the mapping table. If the block is on the SSD, the mapping manager maps the block to a physical block on the SSD by searching the SSD hash table. Otherwise, the mapping manager maps the block to an extent on the HDD. In some cases, the mapping operation for the block IO request may fail. It means that the block IO request is a new write operation. In that case, the mapping manager tries to allocate a block from the SSD and if it fails, it allocate an extent from the HDD. When some extents of the HDD need to be migrated, the mapping manager moves only the blocks which corresponding bit is 1. On the contrary, when some blocks of the SSD need to be migrated, extents for the blocks are allocated from the HDD and the blocks are moved to those extents.

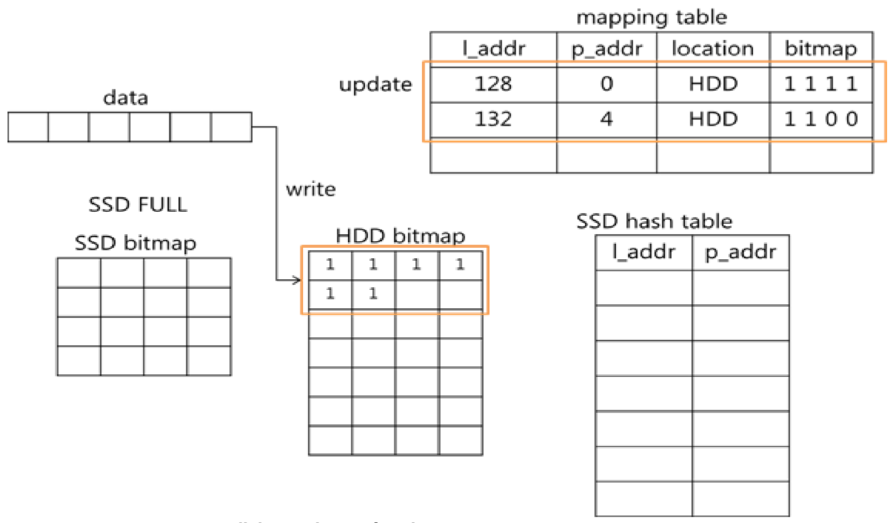
3.2. Read and Write Operations

Figure 5 shows an example of write operations. In Figure 5 (a), the size of write request is 6 blocks. If the SSD has enough room for the write request and the extent of the write request is not found in HDD, the blocks of the write request is placed in the SSD. When the blocks are written to the SSD, SSD bitmap and SSD hash table should be updated as shown in the figure. Figure 6 (a) assumes that the SSD is full so the blocks of the write request cannot be placed on the SSD. In that case, a new extent is allocated from the HDD and the blocks are written to the extent on the HDD.

In Figure 6, an example of read operations is presented. In this figure, the size of read request is 4 blocks. To process the read request, we search mapping table first to check whether the extent for the read request is on the SSD or not. If the extent is on the SSD, we search the SSD hash table to locate physical address of the blocks of the read request.



(a) Write Blocks to SSD



(b) Write Blocks to HDD

Figure 5. Example of Write Operations

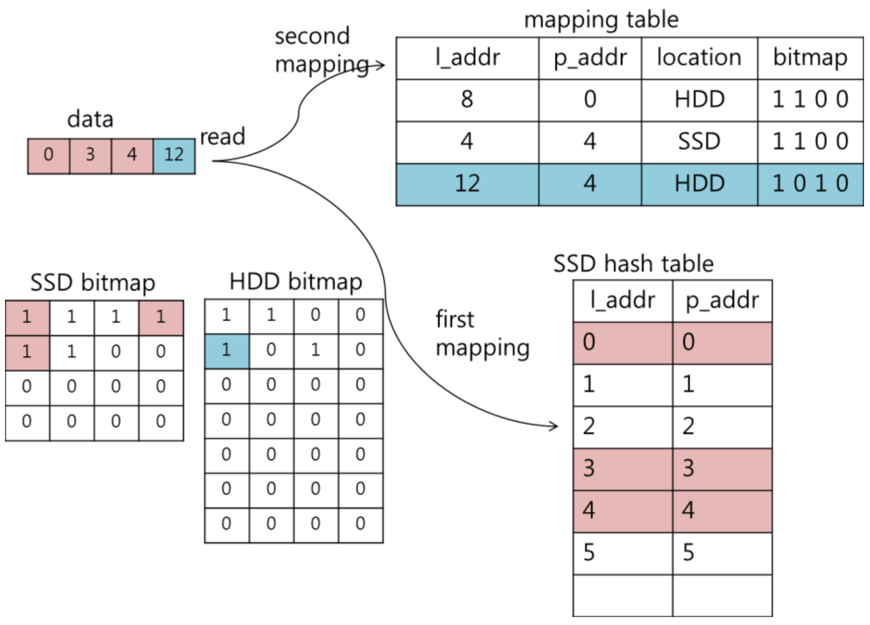


Figure 6. Example of Read Operation

Algorithms of read and write operations are presented in Figure 7. In this figure we use some variables. *bio* is input value given by VFS. It contains requested blocks, the number of blocks and IO direction flag (read or write). *bio_nr_blk* is the number of blocks of bio. *blk_ptr* is used to refer each block of bio and its initial value is always 0. *ext_id* is extent number that is the first block number of an extent. *ssd_nr_free_blk* means the number of free blocks of SSD. *ssd_blk_addr* is used to represent block address in SSD.

Write operation is performed as follows. When a bio is given by VFS, write operation calculates ext_id. Then, it checks whether SSD has enough room for the bio and ext_id extent is on SSD. If SSD cannot accommodate bio or ext_id extent is on HDD, write operation writes the blocks of the bio to ext_id extent. In that case, if needed, new extent is allocated from HDD. Otherwise, the blocks are written to SSD. When writing the blocks to SSD, each block is mapped to a physical block in SSD.

Read operation is performed in the similar way to write operation. Read operation calculates ext_id with bio. Then it maps ext_id to find out the location of the ext_id extent. If the extent is on HDD, read operation reads the whole extent from HDDs. Otherwise, it maps each block of bio to a physical block to read.

(a) variable definition

- bio : Input value from VFS. requested blocks, the number of blocks and IO direction flag (read/write)
- bio_nr_blk : number of blocks of bio
- blk_ptr : refers each block of bio. initial value is 0
- ext_id : extent number
- ssd_nr_free_blk : number of free blocks of SSD
- ssd_blk_addr : block address in SSD

(a) write algorithm

```
hcm_make_request (bio) //bio->bi_rw is WRITE
{
   bio_nr_blk = bio->nr_blk;
   ext_id = calculate ext_id with bio;


   if (search_mapping_table(ext_id) == HDD || bio_nr_blk > ssd_nr_free_blk)
   {
      write_HDD(bio);
      return;
   }


   while(i<bio_nr_blk;) // for all blocks of IO request, blk_ptr refers each block
    {
      if ((ssd_blk_addr = search_SSD_hash_table(bio)) == SUCCESS)
          write_SSD(bio, blk_ptr++);
       else
          if(find_first_zero_bit (ssd_bitmap))
            write_SSD(bio, blk_ptr++);
   }
   return;
}
```

(b) read algorithm

```
hcm_make_request(bio)  // bio->bi_rw is READ
{
   bio_nr_blk = bio->nr_blk;
   ext_id = calculate ext_id with bio;


   if (search_mapping_table(ext_id) == HDD)
   {
      read_HDD(bio);
      return;
   }


   while(i<bio_nr_blk;) // for all blocks of IO request, blk_ptr refers each block
   {
      if ((ssd_blk_addr = search_SSD_hash_table(bio)) == SUCCESS)
         read_SSD(bio);
      else
      {
         search_mapping_table(bio);
         read_HDD(bio);
      }
   }
   return;
}
```

4. Performance Evaluation

In this paper, we evaluate the performance of our proposed mapping method through simulation. We compare our proposed method with the mapping method of the HCMSS in terms of the storage utilization and the migration cost. The simulation parameters used in our experiments are shown in Table 1. The size of IO requests and their logical block addresses are randomly generated. The total number of IOs is 10K and extent size is 64Kbytes. According to [8], the performance of HCMSS is the best when the extent size is 64Kbytes. Also, we assume that the HCMSS consists of a SSD and a HDD. Finally, we fix the migration area as 30% of the total storage space, and the area is randomly selected.

Table 1. Simulation Parameters

parameters	Value
Block Size	4Kbyte
Extent Size	64Kbyte
I/O Size	4~256Kbyte
I/O Count	10000
Migration Area	30%(random)

Figure 8 shows the storage utilization of a SSD space utilization with varying the size of IO requests. As shown in the figure, the storage utilization of our proposed method is much better as when the IO size is becoming smaller. That is, the proposed mapping method works well even though IO requests are random writes for small files. When the IO size is large (more than 96 Kbytes), the storage utilizations of both method are almost 100%. It means that the storage utilization becomes better as the IO size is large.

Figure 9 shows the number of migration blocks with varying the size of IO requests. As shown in the figure, the number of migration blocks of the proposed method is much better as when the IO size is becoming smaller like the previous experiment. Also, when the IO size is large, the migration cost is almost same. The reason is that the storage utilization effect to the migration cost.

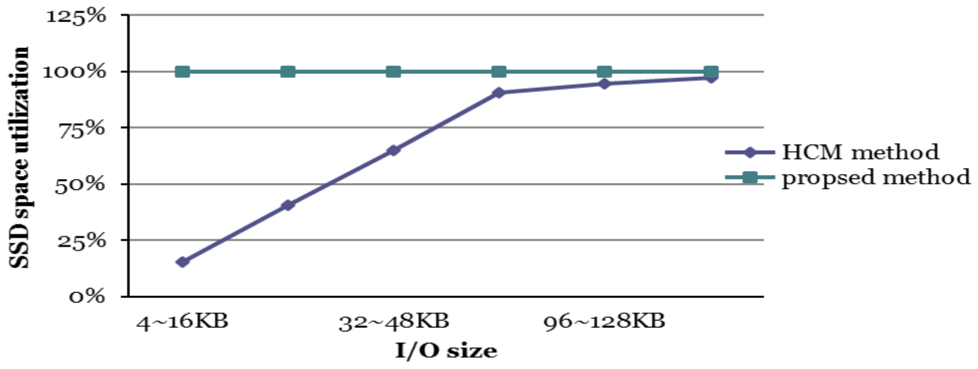


Figure 8. Storage Utilization of SSD with Varying the Size of IO Requests

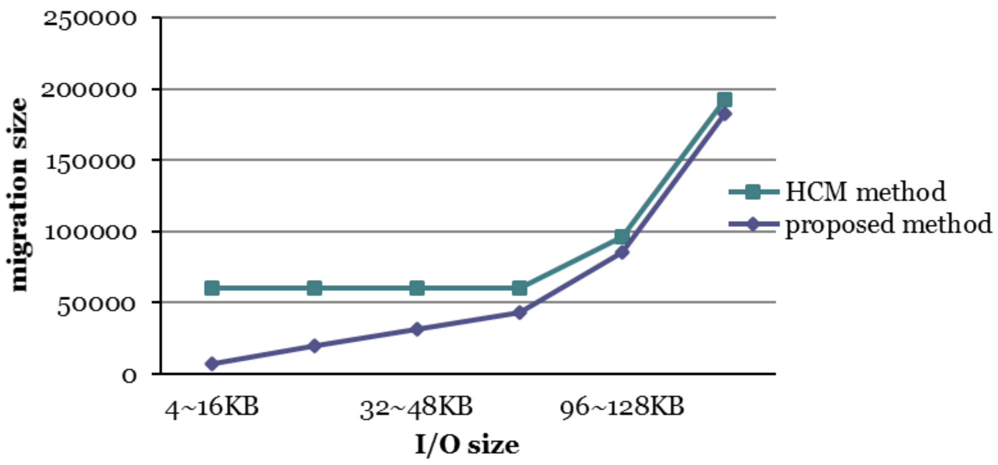


Figure 9. Migration Size with Varying the Size of I/O Requests

5. Conclusion

In this paper, we proposed a new mapping method for the HCMSS to solve the extent fragmentation problem. The proposed method used hybrid approach that manages SSD and HDD with different IO unit size. The proposed mapping method increased the storage utilization of the SSD up to 4 times when the IO size is 4 or 16kbytes comparing to the original mapping method of the HCMSS.

Acknowledgement

This paper is a revised and expanded version of a paper entitled Mapping Method for Hybrid Storage to Enhance Storage Utilization of SSD presented at CES-CUBE 2014 that was held last July 17-20, 2014 at Pacific Islands Club, Saipan, USA.

References

[1] J. Hwang and S. Chung, "Technology trend of DDR based SSD storage system", NI-PA Weekly Trend Report, 1421, **(2009)**, pp. 28-41.
[2] A. Badam and S. SDAlloc, "Hybrid SSD/RAM memory management made easy", "Proceedings of the 8th USENIX conference, **(2011)**.
[3] X. Wu and A. Reddy, "Exploiting concurrency to improve latency and throughput in a hybrid storage system", Annual IEEE/ACM International Symposium on Modeling, **(2010)**, pp. 14-20.
[4] I. Koltsidas and S. D. Viglas, "Flashing up the storage layer", Proceedings of the VLDB Endowment, **(2008)**, pp. 514-525.
[5] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee and S. Maeng, "SSD-HDD-Hybrid virtual disk in consolidated environments", Proceedings of International Conference on Parallel Processing, **(2009)**, pp. 375-384.
[6] http://www.storageperformance.org/.
[7] Y. Ryu, "A buffer management scheme for mobile computers with hybrid main memory and flash memory storages", IJMUE, vol. 7, **(2012)**, pp. 235-240.
[8] K. Khil, S. Song, K. Bok and S.-K. Cheong, "Hot and Cold Data Replacement Method for Hybrid Storage System", Information Journal, vol. 16, no. 12, **(2013)**, pp. 8331-8338.