ReSSD: A Software Layer for Resuscitating SSDs from Poor Small Random Write Performance

Youngjae Lee
Computer Science Department
KAIST
Deajeon, South Korea
yjlee@camars.kaist.ac.kr

Jin-Soo Kim
School of Information and Communication Engineering
Sungkyunkwan University
Suwon, South Korea
jinsookim@skku.edu

Seungryoul Maeng
Computer Science Department
KAIST
Deajeon, South Korea
maeng@camars.kaist.ac.kr

ABSTRACT

NAND flash-based solid state drives have emerged as revolutionary storage media during recent years. However, the wide-spread of SSD technology is currently obstructed by the fact that the random write bandwidth is lower than the sequential write bandwidth by several orders of magnitude.

This paper proposes a novel software layer called ReSSD whose purpose is to resuscitate SSDs from poor small random write performance with low memory usage. ReSSD works as a virtual block device on SSD which does not require any modifications of the operating system kernel and applications. By inspecting all incoming requests, ReSSD identifies small random writes which have potential to degrade SSD's performance significantly and transforms them into sequential and ordered-sequential writes which are more favorable to SSDs. Our evaluation results with Postmark show that ReSSD improves the overall performance by up to 72% using a few megabytes of kernel memory.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.7 [**Operating Systems**]: Organization and Design

General Terms

Design, Management, Performance

Keywords

Solid state drive, NAND flash memory, Small random write

1. INTRODUCTION

Recently, NAND flash-based solid state drives (SSDs) have drawn considerable attention in both industry and academia. Many studies have been carried out to adopt SSDs to database systems or storage systems.

However, the random write performance, particularly in small size, is somewhat problematic in SSDs.For example, the 4KB random write bandwidths of SSDs from Samsung and SuperTalent are 0.421MB/s and 0.018MB/s, respectively, while their sequential write bandwidths are more than 110MB/s. Since this is a critical obstacle impeding the widespread adoption of SSD technology, it is necessary to resuscitate SSDs from such poor small random write performance.

A software company called EasyCo LLC has recently proposed an SSD performance enhancement solution called Managed Flash Technology (MFT) [2]. Although the detailed architecture of MFT is still not known exactly, it appears to handle all write requests in a sequential fashion, so that the underlying SSD never encounters random writes. However, MFT has a serious drawback that the amount of mapping information needed to keep track of the physical locations becomes inevitably very huge. It is known that MFT on a 128GB SSD typically requires about 150MB of kernel memory. Considering the current trend of increasing SSD capacity, the MFT approach is not practical especially for systems with a modest amount of memory such as netbooks.

In this paper, we propose a novel software approach called ReSSD which aims at resuscitating SSDs from its poor small random write performance with low memory usage. Unlike MFT, ReSSD records only identified small random writes sequentially in the reserved area and moves them to their original location eventually in an ordered-sequential write fashion [1] . The reserved area size is smaller than the workload size by an order of magnitude, so the amount of in-memory mapping information is also very small.

2. ReSSD

2.1 Basic components

ReSSD consists of three major components: *the router*, *the mover*, and *the RB-tree (red-black tree)*, as shown in Figure 1. The underlying SSD is divided into two regions, *the normal area* and *the reserved area*. The normal area is visible to the upper layer as a single block device. On the other hand, the upper layer is not aware of the reserved area.

All I/O requests which the upper layer issues are first transferred to the router. In case of write requests, the request pattern identifier (RPI) inside the router checks whether the request is a small random write or not. If the request is identified as a small random write, its data are sequentially written into the reserved area. Otherwise, the request

[1] Note that the ordered-sequential write denotes the write pattern where the accessed logical blocks are arranged in increasing order of LBAs (Locgical Block Addresses) [1].

SAC'10 March 22-26, 2010, Sierre, Switzerland.

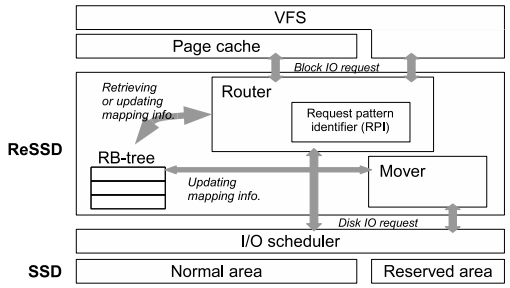

Figure 1: The Overall Architecture of ReSSD

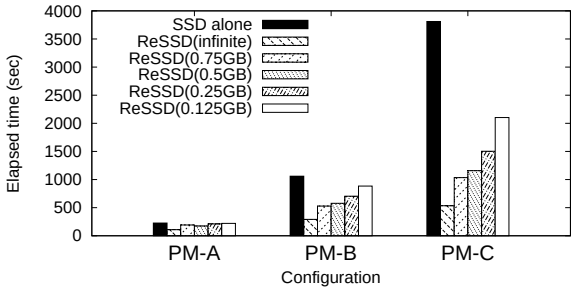

Figure 2: Postmark elapsed time

is forwarded to the normal area. When the router receives read requests, it serves them either from the reserved area or from the normal area according to the mapping information stored in the RB-tree.

The main role of the mover is to reclaim free space in the reserved area by moving some of data to the normal area. The mover usually remains inactive until the router wakes it up as the amount of free space falls below a certain threshold whose default value is half the reserved area size.

The RB-tree contains the mapping information needed to track the original locations of data, which are sequentially written to the reserved area. All nodes of the RB-tree is always kept in memory and their memory usage is about 1MB per 100MB of the reserved area.

2.2 Improving small random writes

The RPI considers a write request as a small random write if its logical sector numbers (LSNs) do not immediately follow the previous write request and its size is not larger than 8KB, as this type of write request has potential to degrade SSD's performance significantly. The router redirects such identified small random writes to the reserved area and their data are written sequentially. In case the reserved area already has some data for the requested sector numbers, the previous versions of data are invalidated by the router.

As soon as the mover becomes active by the router, the mover reads valid data from the reserved area in the ascending order of their original LSNs and writes them into the normal area in an ordered-sequential write fashion.

By filtering out small random writes, the underlying SSD does not suffer from random writes whose size is less than or equal to 8KB. Instead, the underlying SSD only encounters sequential writes given by the router and ordered-sequential writes issued by the mover, which show higher performance than small random writes in most of SSDs. Therefore, ReSSD can improve the overall performance significantly.

3. EVALUATION

We have implemented ReSSD using *the device-mapper* on Linux 2.6.29.4, one of the Linux kernel components for logical volume management. We have evaluated ReSSD on a SuperTalent FTM60GK25H SSD. The SSD has two partitions, one 51.2GB and the other 8GB. In this evaluation, ReSSD employs the first partition as the normal area and part of the second one as the reserved area. When the performance is measured on SSD alone (without using ReSSD), the second partition has not been used. The ext4 file system and the CFQ scheduler have been used. The memory size available to the kernel is limited to 384MB in order to minimize the effect of page cache.

Table 1: Postmark configurations

	Initial file set size	Initial files	Directories	Transactions
PM-A	0.8 GB	80000	10000	30000
PM-B	1.6 GB	160000	20000	60000
PM-C	2.4 GB	240000	30000	90000

Figure 2 presents the elapsed time to execute each *Postmark* workload configuration described in Table 1. The number in parentheses for ReSSD indicates the reserved area size. The *infinite* size represents that the reserved area is large enough to make the mover remain inactive during the entire Postmark run. ReSSD outperforms the case with SSD alone in all workload configurations. With the larger reserved area size, ReSSD accomplishes more performance improvement. In case of PM-C, ReSSD improves the elapsed time by 45% (with 0.125GB) – 72% (with 0.75GB) depending on the reserved area size. This improvement is close to the upper bound of 85% when the reserved area size is infinite.

4. CONCLUSION

In this paper, we propose ReSSD to resuscitate SSDs from poor small random write performance with low memory usage. ReSSD identifies small random writes and converts them into sequential and ordered-sequential writes, which are more favorable to SSDs. The proposed approach accomplishes noticeable performance improvement with low memory usage under the workload including many small random writes.

5. ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (R01-2007-000-11832-0). Also, this work was supported by the IT R&D Program of MKE/KEIT [2009-F-039-01, Development of Technology Base for Trustworthy Computing].

6. REFERENCES

[1] L. Bouganim, B. por Jonsson, and P. Bonnet. uFLIP: Understanding flash io patterns. In *CIDR*, 2009.

[2] D. Dumitru. Optimizing flash storage with linearization software. In *FlashMemory Summit 2009*, 2009.