To TRIM or Not to TRIM: Judicious TRIMing for Solid State Drives

Choulseung Hyun*[1], Jongmoo Choi[2], Donghee Lee[1], and Sam H. Noh[3]

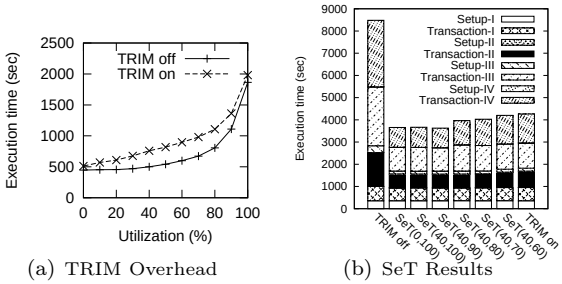
[1] *University of Seoul, {cshyun, dhl_express}@uos.ac.kr*
[2] *Dankook University, choijm@dankook.ac.kr*
[3] *Hongik University, Samhnoh@hongik.ac.kr*

The TRIM command is a relatively new addition to the ATA standard [2]. As solid state drives (SSDs) operate differently from hard disks, there exists a discrepancy between the operating system and device's view of storage. The TRIM command works to eliminate this discrepancy. In this study, we propose a policy that allows judicious use of TRIM to reap the benefits that were intended of TRIM.

Understanding TRIM

SSDs employ a sophisticated software module called the Flash Translation Layer (FTL) to provide the conventional disk interface. The FTL has its own view of storage that may differ from that of the file system. For example, consider a situation where file A that is composed of 5 blocks are deleted from the file system. The file system will mark these blocks as being free and make appropriated changes to the metadata that it manages. This also results in reduced utilization from the file system point of view. The blocks themselves residing in storage will not be explicitly deleted, that is, erased. However, with SSDs, if no explicit information is sent, then the 5 blocks for file A will still be considered to be in use even though file A does not exist. This hinders reclaiming of free space, and utilization for the device remains unchanged even though deletion has occured. Furthermore, when garbage collection occurs, the FTL will consider the 5 blocks to be valid and copy them to a new location incurring unnecessary copies that deter performance.

The TRIM command was introduced to effectively rid of these problems. TRIM allows the file system to send which blocks are no longer valid to the underlying storage device. The FTL can use this information when managing its storage such as performing garbage collection. Hence, TRIM reduces, on average, garbage collection cost and also increases the life time of SSDs. However, TRIM incurs non-negligible overhead, especially when utilization is high and under heavy workloads, as execution of TRIM has to follow strict guidelines due to synchronization issues [1]. Currently, operating systems employ TRIM in a naive manner not considering the overhead that it incurs. For example, in Linux ext4 TRIM is either turned on or off. When turned on the TRIM command is issued at every journal commit point.



(a) TRIM Overhead (b) SeT Results

Figure 1: Experimental Data

Motivation: Observing Trim Overhead

Figure 1(a) shows the execution time of a Postmark benchmark on an off-the-counter 60GB SSD at a particular file system utilization setting. Utilization is set for each data point by first formating the ext4 file system on the SSD, then trimming all empty space of the file system, then aging the file system to the fullest, that is, 100% utilization, then deleting files randomly down to the desired utilization, and then sending TRIM operations for the blocks of the deleted files. Then, the file system is formatted once more to reset the file system uti-

*Student, Presenter

lization to 0 to make sure we start from the exact same file system utilization. Finally, we mount the file system with TRIM set to on or off. Then, we ran the Postmark benchmark by generating 4250 files of file size ranging between 0.1 to 3MB, and then executing 20,000 transactions.

The result contrasts the execution of the benchmark on the Linux ext4 file system with TRIM turned on and off; when turned on, we simply use the default TRIM setting. It shows making use of the TRIM command may show inferior performance compared to when not using it. The main reason behind this observation is the overhead involved when issuing the TRIM command.

The Selective TRIM Policy

The question we need to answer is how to make good use of the TRIM command. That is, issuing TRIM incurs overhead, so it should be used sparingly and judiciously. However, we want to make use of TRIM to reap the benefits it has promised. *Selective* TRIM *(SeT)*, which is the policy that we propose, considers the utilization of the system and the workload characteristics in turning TRIM on and off.

Development of the SeT policy is motivated by another observation of Figure 1(a). Observe from the results, when TRIM is turned on, how the slope is steepest between utilization 90% and 100%, and how this differs from lower utilization ranges. From the standpoint of the operating system, the difference between high utilization and low utilization is essentially the reduction in utilization that one could attain by making use of TRIM. Hence, when TRIM is employed at high utilization, the performance gains (the difference on the y-axis) is large, while when utilization is low, there is hardly any benefit. Hence, in SeT, we keep TRIM on when utilization is high, that is, higher than the *high watermark*, and kept turned off when low, that is, lower than the *low watermark*. Both watermarks are controllable knobs.

For the range between the two watermarks, we take into account the activities of the workload. For this, the operating system keeps track of the utilization of the file system. If utilization keeps increasing, this mean that files and/or directories are being created and expanded rather than being deleted. Issuing TRIM in such situations may negatively affect their activities. Hence, in this case, we turn TRIM off. On the other hand, if utilization keeps decreasing, then this means that file deletions are dominant. Such activities may be delayed if necessary. Hence, TRIM is turned on.

Experimental Results

For our experiments, we added about 400 lines of code to the Linux ext4 file system to implement the SeT policy. To capture the adaptability of the SeT policy, we vary the utilization setting as the Postmark benchmark is executed. Our experiment goes through a sequence of utilization setup and transaction execution sequences. That is, in the utilization setup stage, utilization of the file system is adjusted to 60%, 40%, 70%, and 60%. After each utilization setup stage, we execute the Postmark benchmark.

The results are shown in Figure 1(b). For each graph, the numbers in parentheses designate the low and high watermarks. We observe that always keeping TRIM on (rightmost bar labeled TRIM on) performs better than always keeping it off (leftmost bar labeled TRIM off). This is because utilization of the SSD is high and also because in forcefully adjusting utilization in our experimental setup, considerable space is being deleted, making the situation ripe for TRIM.

The SeT policy, regardless of the watermark values, outperforms the Linux default TRIM policy, whether TRIM is kept on or off. SeT(0,100) refers to the situation where only the workload dynamics are considered. The middle 5 bar graphs (SeT(40,100) to SeT(40,60)) show that performance degrades as the high watermark is decreased. This shows that keeping TRIM on all the time even when utilization is high has negative effects.

Acknowledgement This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. R0A-2007-000-20071-0) and by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2009-0085883).

References

[1] M. Saxena and M. M. Swift. FlashVM: Virtual Memory Management on Flash. In *Proceedings of the 2010 USENIX Annual Technical Conference*, 2010.

[2] F. Shu. Notification of Deleted Data Proposal for ATA-ACS2. http://t13.org/, 2007.