Flash SSD Oriented IO Management for Data Intensive Applications

Yongkun Wang† Kazuo Goda‡ Miyuki Nakano‡ Masaru Kitsuregawa‡
† *Student, Graduate School of Information Science and Technology*
‡ *Institute of Industrial Science*
The University of Tokyo
{*yongkun,kgoda,miyuki,kitsure*}*@tkl.iis.u-tokyo.ac.jp*

1 Introduction

Flash SSDs are being incorporated into enterprise storage for achieving high performance in data intensive applications. When deployment to such an environment is considered, a major concern of flash SSD is its slow random writes. A random write to flash SSD usually takes several milliseconds, which are unbearably long in comparison with fast random reads (taking several to tens of microseconds). One essential research topic is to mitigate such poor performance of random writes. Many researchers have studied performance improvement techniques in different storage software layers such as DBMS storage engine[1], file system[2], device driver[3] and flash SSD controller called FTL[4]. Their work has successfully provided substantial write performance improvement, but they focused on a specific layer through IO path. There are yet few studies to explore overall IO management for achieving potential performance of flash SSD.

In this report, we present our basic idea of IO management for flash SSD. The key point is to utilize runtime application information to maximize write scheduling opportunities for improving write performance. In many environments, most of data writes issued by data intensive applications do not have to be flushed to the storage device immediately. Rather they can be deferred for some time. So some scheduling opportunities are allowed. However, these opportunities are not explicitly informed to IO path. The data intensive applications often expect their issued writes to be flushed to the storage device at a certain time point. That is, without this knowledge, IO path often misses such write scheduling opportunities. In contrast, if this knowledge is conveyed to IO management, write sequence could be fully optimized. The system can understand how long it can defer the given writes. This is beneficial to schedule the write sequence in order to improve the throughput. In the next section, we take a scenario of database systems for introducing the proposed idea.

2 Flash SSD Oriented IO Management

Many database systems allow write requests to secondary storage to be deferred and then flushed to the storage in a batch. Such write deferring has benefits of providing scheduling opportunities for reducing IO cost of the writes at run time. As writes are deferred longer, scheduling benefits could be larger, giving higher performance. But, database systems need to guarantee that all the writes older than a last checkpoint are reflected onto the secondary storage. The available deferring window is strictly limited by database checkpointing. Checkpoint is crucial information for write scheduling when we try to improve the IO performance along IO path.

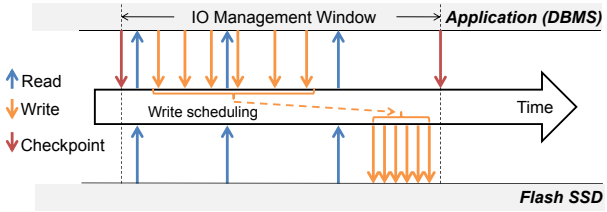


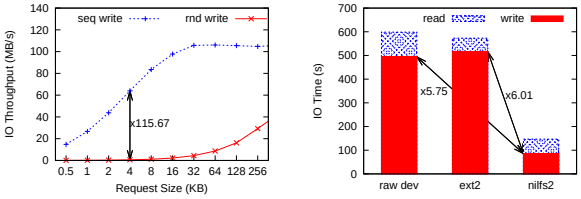
Figure 1: Flash SSD oriented IO management with checkpoint information

Available write deferring window (i.e. IO management window) can be defined as illustrated in Figure 1. Each window starts when a checkpoint finishes and the window ends when the next checkpoint starts. Writes can be deferred and scheduled at run time within the window and then reflected in a batch manner to improve performance. Let us summarize three major techniques on the deferred writes.

- **Write coalescing** merges overlapping write requests. This helps to reduce the amount of writes.
- **Write converting** translates random access address into a sequential order, like a log-structured manner. This can pack more writes into flash SSD block (erase unit) so as to reduce the number of slow erase operations.
- **Write aligning** arranges write requests along block boundary. This also works to reduce unnecessary erase operations.

3 Preliminary Experiments

First, we studied IO performance of existing IO techniques on flash SSDs. Due to the page limitation, we briefly show a result of TPC-C benchmark, standard online transaction processing benchmark, on a Mtron PRO 7500 32GB SLC SSD. A fixed number of TPC-C transactions were ran on



(a) IO Throughput: sequential write vs. random write

(b) IO Replay of Comm. DBMS on SSD with 80MB DBMS buffer

Figure 2: Gap of basic write performance of flash SSD and obtainable performance in DBMS

three types of volume configurations (using raw device, using ext2fs and using nilfs2) on commercial DBMS X and Linux operating system on a Intel 2GHz server with 2GB memory. In each configuration case, we traced all IO events during the transaction execution, and afterwards, we replayed those IO traces on the same SSD to measure pure IO time that were taken for executing the TPC-C transactions. Figure 2 summarizes the results. As a reference, Figure 2 also depicts basic performance of random writes and sequential writes that we obtained using a micro benchmark on the same SSD. Nilfs2, a typical LFS implementation on Linux, could improve write performance x5.75 and x6.01 in comparison with raw device and ext2fs configuration respectively. This was because of Nilfs2's copy-on-write feature, which tries to convert slow random writes to fast sequential writes. Note that there is still some performance improvement room for this SSD device. The basic performance graph shows that such conversion could potentially improve throughput up to x115.67. Although current IO techniques can improve IO performance on flash SSDs substantially, they cannot reach to the potential performance.

Based on this observation, we had a further experiment to study the potential performance improvement of the proposed flash SSD oriented IO management. Similarly, we conducted a trace driven experiment. We implemented IO management techniques such as write coalescing, write converting and write aligning, on top of our IO replaying environment, and tested IO performance of such techniques on the environment. IO trace files were generated by TPC-C execution on raw device configuration. For studying the effects of checkpoint, we tested two checkpoint intervals (30 and 300 seconds) for TPC-C execution. Several buffer size configurations were also studied for IO management. Figure 3 summarizes the results. Major findings are summarized below.

- Write coalescing could reduce write time by reducing the amount of writes. The merging effect became larger with longer checkpoint intervals and larger buffer sizes.
- Write converting could successfully translate random accesses into a sequential order. Thus significant improvement of write performance was observed.
- Write aligning could reduce the write time around 50% even for improved write sequence by write converting.
- Simple write deferring using IO management window increased total write time. This was possibly due to flash SSD's bathtub effect[5].

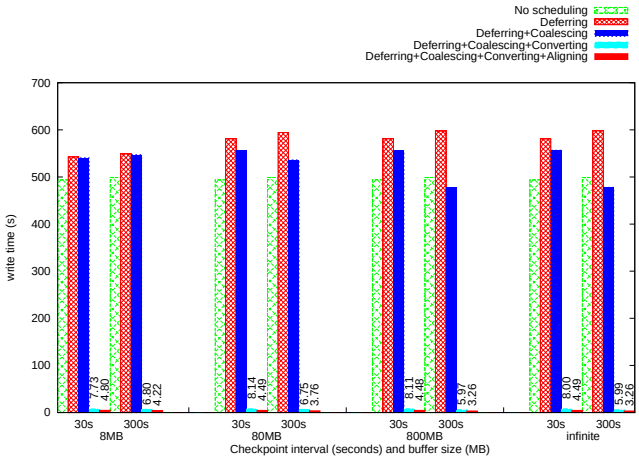


Figure 3: Online Scheduling with varied buffer size limits

Interestingly, overall performance improvement was not so sensitive to write buffer sizes. 80MB buffer could obtain performance improvement that was comparable with experimentally assumed infinite buffer.

4 Conclusion

We have proposed flash SSD oriented IO management for data intensive applications. By utilizing database checkpoint information, the scheduling opportunities can be fully maximized. Preliminary experiment shows that our proposal could achieve potential performance benefit of flash SSD. We continue to explore further topics such as efficient runtime read/write scheduling to balance checkpoint flushing overhead and implementation of the proposed IO management to current storage software stacks.

References

[1] S.-W. Lee and B. Moon. Design of flash-based DBMS: an in-page logging approach. In Proc. of SIGMOD, pp. 55-66, 2007.

[2] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li. DFS: A File System for Virtualized Flash Storage. In Proc. of FAST, pp. 85-100, 2010.

[3] Y.-R. Kim, K.-Y. Whang, and I.-Y. Song. Page differential logging: an efficient and DBMS independent approach for storing data into flash memory. In Proc. of SIGMOD, pp. 363-374, 2010.

[4] H.-J. Choi, S. H. Lim, and K. H. Park. JFTL: A flash translation layer based on a journal remapping for flash memory. ACM TOS, vol. 4, no. 4, 2009.

[5] R. Freitas and L. Chiu. Solid-State Storage: Technology, Design and Applications. FAST2010 Tutorial, `http://www.usenix.org/events/fast10/tutorials/T2.pdf`, 2010.