Boosting Random Write Performance for Enterprise Flash Storage Systems

The 27th IEEE Symposium on Massive Storage Systems and Technologies, 2011

Tao Xie and Janak Koshia, Department of Computer Science, San Diego State University, San Diego, CA 92182

Introduction

NAND flash memory has been successfully employed in mobile devices like PDAs and laptops. With recent advances in capacity, bandwidth, and durability, NAND flash memory based Solid State Disk (SSD) is starting to replace hard disk drive (HDD) in desktop systems. Integrating SSD into enterprise storage systems, however, is much more challenging. One of the major challenges is that server applications normally demand an exceptional random I/O performance, whereas current SSD performs poorly in random writes. To fundamentally boost random write performance, we propose a new write cache management scheme called EPO (element-level parallelism optimization), which reorders write requests so that element-level parallelism within SSD can be effectively exploited. We evaluate EPO using a validated disk simulator with realistic server-class traces. Experimental results show that EPO noticeably outperforms traditional LRU algorithm and a state-of-the-art flash buffer management scheme BPLRU (block padding least recently used).

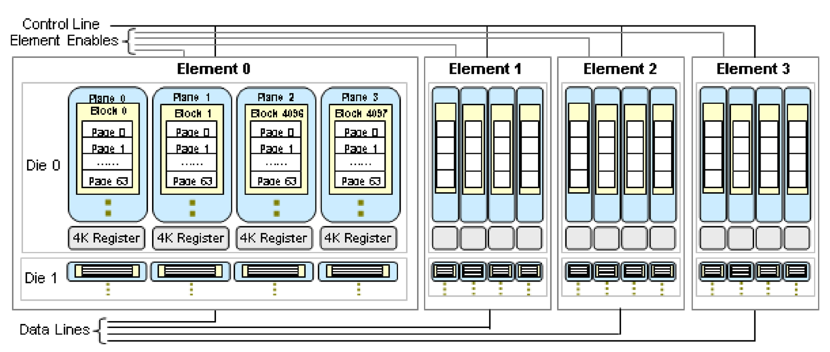
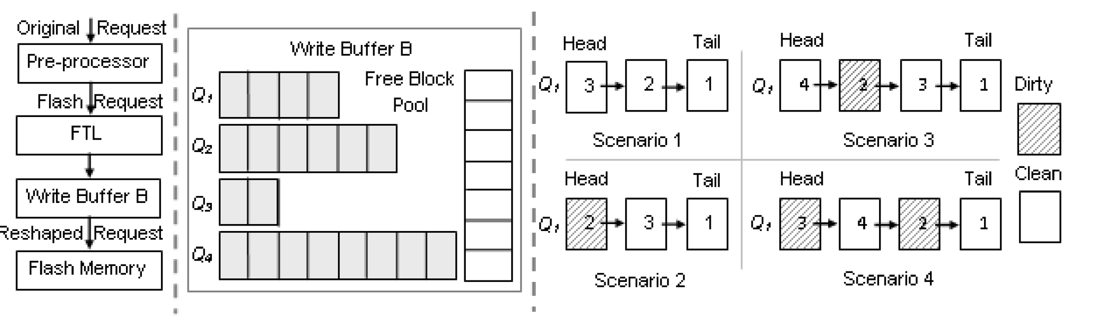


Figure 1. Internal structure of a SSD with four elements.

Methods and Algorithm



(a) (b) (c)

Figure 2. (a) Request processing flow; (b) internal structure of *B*; (c) states of Q_1 in sequence (1, 2, 3, 2, 4, 3).

```
Input: P, a pre-processed write request set; B, a write buffer managed by the EPO scheme
Output: R, a re-shaped write request set that is aware of the element-level parallelism in SSD
1.  Clear B and R; k = 1; e = number of elements in SSD;
    create e queues from Q_1 to Q_e in B
2.  for each request r_k ∈ P do
3.    j = number of pages requested by r_k
4.    Create a temporary array T with j cells
5.    if j > 1 /* the request r_k is a multiple-page request */
6.       Divide the request r_k into j single-page write requests
         and store them in T sequentially
7.    else
8.       Store r_k in T
9.    end if
10.   h = 1
11.   for each single-page write request t_h ∈ T do
12.      i = the element number that t_h targets on and 1 ≤ i ≤ e
13.      Search t_h in the corresponding queue Q_i in B
14.      if the page requested by t_h is found in Q_i
15.        Replace it with t_h and move t_h to the head of Q_i
16.      else
17.        if there is no free space in B to accommodate t_h
18.          for each queue from Q_1 to Q_e in B
19.            Evict the request at the tail to R
20.            Change its arrival time to the arrival time of t_h
21.          end for
22.        end if
23.        Insert t_h at the head of Q_i
24.        The free block pool is increased by e - 1 blocks
24.      end if
25.      h = h + 1
26.   end for
27.   Delete the temporary array T
28.   k = k + 1
29. end for
```

Figure 3. Algorithm of the EPO scheme.

Fig. 2a illustrates the request processing flow of EPO. Similar to BPLRU, EPO only processes write requests. For read requests, it simply forwards them to the FTL. The basic unit in the write buffer B is a block whose size is equal to the size of a flash memory page. Within B, EPO maintains a free block pool and multiple queues (Fig. 2b). Assume that there are only four elements in an SSD, Fig. 2b demonstrates how EPO manages the free block pool and the four queues with each queue corresponding to one element. Fig. 2c shows four different states of Q in sequence(1, 2, 3, 2, 4, 3).

PERFORMANCE EVALUATION

The goal of this experiment is to compare EPO against two well-known cache management algorithms LRU and BPLRU, and to understand the impact of write buffer size on the performance of the four algorithms including NoCache. We tested write buffer size from 4 MB to 32 MB with 48 elements. All simulation experiments are conducted in three stages sequentially*: pre-processing, reshaping, and feeding*.

Parameter	Value (Fixed) – (Varied)
Write buffer capacity (MB)	(8) – (4, 8, 16, 32)
Number of elements	(48) – (16, 32, 48, 64)
Page size (KB)	(4) – (1, 2, 4)
Flash block size (page)	(64)
Element capacity (GB)	(4)
Flash SSD capacity (GB)	(192) – (64, 128, 192, 256)
Block erase latency (µs)	(1500)
Page read latency (µs)	(25)
Page write latency (µs)	(200)
Chip transfer latency per byte (µs)	(0.025)
Number of planes in an element	(8)

We evaluate the four buffer management schemes by running simulations over three real system traces: Financial1, Financial2, and TPC-C, which have been widely used in the literature. We selected those three traces so that the EPO scheme can be evaluated under different degrees of access randomness. Since the simulation times in our experiments are much shorter than the time spans of the traces, we truncate each trace such that only the first 2, 0.65, and 2 million write requests are included for Finanaical1, Financial2, and TPC-C, respectively. The main simulation parameters are shown in left table.

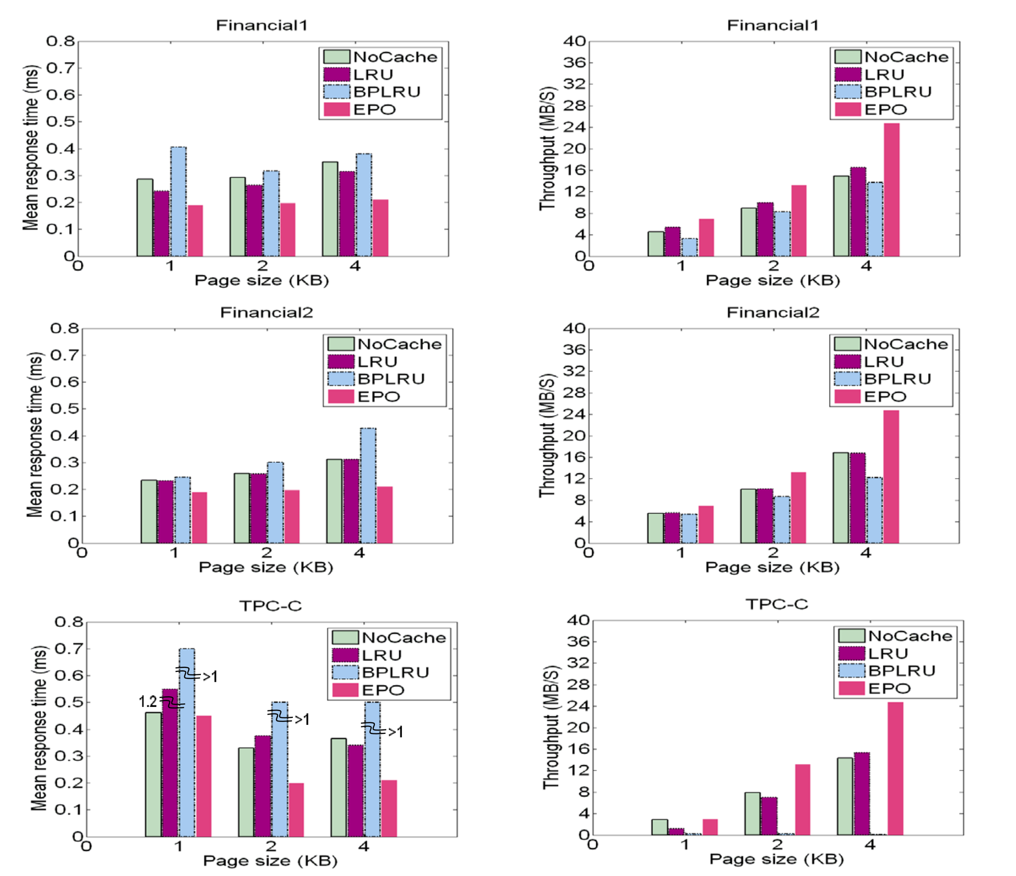


Figure 4. Performance impact of write buffer size on the four schemes.

Fig. 4 shows that the mean response time of all four schemes does not noticeably change when the size of the write buffer increases from 4 MB to 32 MB. This is because the write buffer is still very small considering the large volume of requests from the three server-class workloads. Consequently, the entire write buffer even in its maximal size 32 MB is quickly filled out by arrival requests, and thus, increasing write buffer size does not result in an apparent performance improvement. Still, EPO always outperforms the three existing schemes in all cases for it exploits the element-level concurrency. One interesting observation from Fig. 4 is that increasing the size of write buffer can neither significantly reduce the mean response time nor increase throughput. The rationale behind is that larger buffer size has little impact on a totally random access pattern. To understand the sensitivity of EPO to other parameters, we also measured the performance of EPO when changing the number of elements and page size.

We vary the size of a flash page from 1 KB to 4 KB. Fig. 5 plots the performance of the four algorithms as functions of the size of a flash page. Several important observations can be drawn from Fig. 5. First of all, flash page size has a noticeable impact on the three existing algorithms. Recall that after the pre-processing stage each write request's size is configured to its closest multiples of flash pages and each page is 4 KB. Therefore, when flash page size enlarges to 4 KB, each request needs to write multiple pages rather than a single page. Therefore, the response time of NoCache and LRU increases. The mean response time of EPO, however, only slightly changes because it always splits each multiple-page request into multiple single-page requests (Step 6 in Fig.3). Second, larger page size usually results in a higher throughput. In Financial 1 case, EPO increases the throughput by 4.9 times when flash page size changes from 1 KB to 4 KB. The reason is that larger flash page improves write efficiency and decreases the number of block erasures. Lastly, TPC-C workload is so intensive that all three existing algorithms encounter large mean response times.

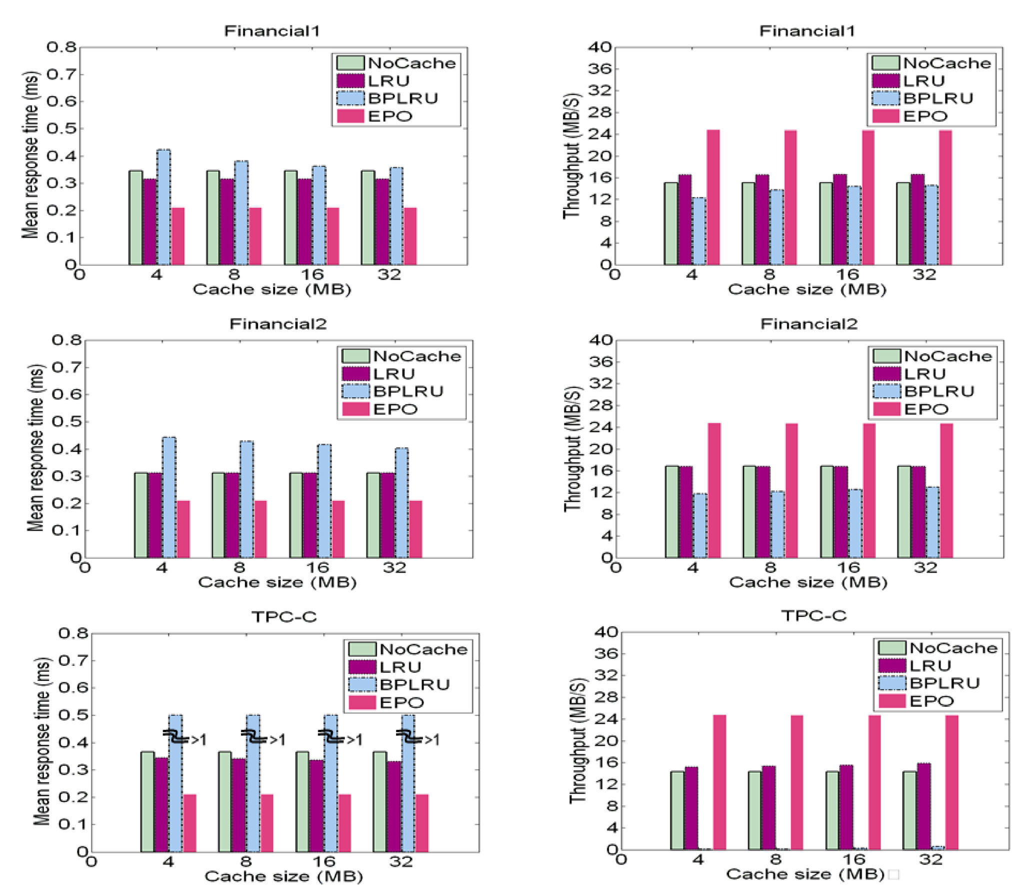


Figure 5. Performance impact of flash page size on the four schemes.

PERFORMANCE EVALUATION

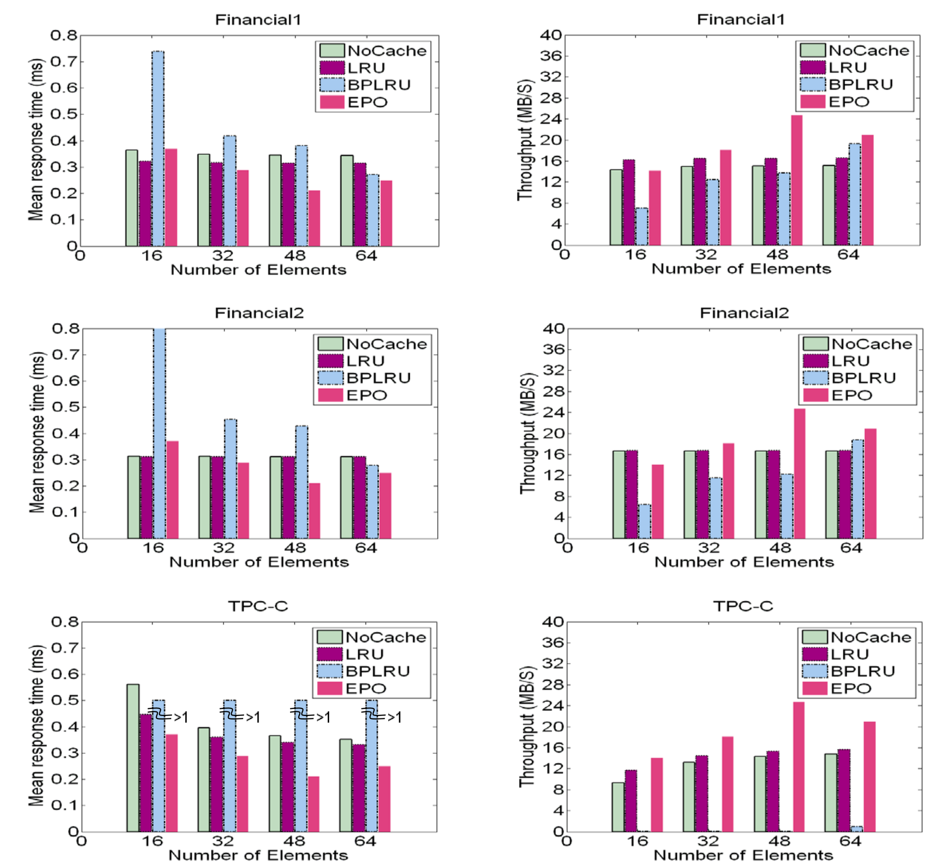


Figure 6. Scalability of the four schemes.

Fig. 6 demonstrates that the scalability of all algorithms including EPO is sensitive to the workloads. In Financial1 and Financial2 cases, increasing the number of elements does bring an apparent improvement in either mean response time or throughput (Fig. 6). After analyzing the two traces we realized that the outcome is expected because both Financial1 and Financial2 workloads have noticeable temporal locality and spatial locality. As a result, a large portion of requests concentrate on a small logical space so that newly added elements cannot receive enough requests to share the entire load. In Financial2 case, compared with NoCache and LRU, EPO on average reduces mean response time by 10.5% and 10.2%, respectively. Compared with NoCache, LRU, and BPLRU, EPO on average improves throughput by 16.6%, 16.2%, and 59.1%, respectively. In TPC-C scenario, EPO significantly outperforms all three existing algorithms in terms of throughput. This is because EPO fully employs the element-level parallelism within an SSD.

Conclusion

In this paper, we address the issue of SSD random write performance in server applications. The basic idea of EPO is to reshape write access pattern by dynamically grouping multiple buffered write requests that target on distinct elements into one batch. EPO exploits the element-level concurrency to significantly shorten mean response time and improve throughput. Although EPO also employs an extra battery-backup RAM buffer inside SSD and reshapes write access pattern, it is orthogonal to current *write requests buffering and reordering* schemes because it seeks to exploit element-level parallelism within SSD, which is a new avenue to solve the SSD random write problem. Comparing with *adding non-volatile RAM (NVRAM) buffer* and *enhanced FTL engine developing* approaches, EPO has several desired advantages. First, its hardware cost is low because of the limited size of RAM buffer used. Second, it does not require any change in the FTL layer, and thus, is easy to be integrated into modern SSDs. Lastly, its low time complexity implies its potential to be implemented in real applications. Experimental results demonstrate that EPO consistently outperforms a state-of-the-art write buffer management scheme BPLRU. It also performs better than the traditional LRU algorithm.

Acknowledgement

This work was sponsored by the US National Science Foundation under grants CNS-0845105, CNS-0834466, and CCF-0702781.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proc. USENIX Annual Technical Conference*, pp. 57-70, 2008.

[2] M. Balakrishnan1, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD Reliability," *Proc. 5th ACM European Conf. Computer Systems*, Paris, France, April 13-16, 2010

[3] S. Boboila and P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis," *Proc. 8th USENIX Conference on File and Storage Technologies (FAST)*, 2010.

[4] L.P. Chang and T.W. Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation," *ACM Transactions on Storage*, Vol. 1, No. 4, pp. 381-418, 2005.

[5] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Enterprise Storage to SSDs: Analysis of Tradeoffs," *Proc. 4th ACM European Conf. on Computer Systems*, 2009.