Polymorphic Mapping for Tera-scale Solid State Drives

Sungmin Park[1] , Jaehyuk Cha[1], Youjip Won[1], Sungroh Yoon[2], Jongmoo Choi[3], Sooyong Kang[1]
[1]*Div. of Comp. Sci. & Eng., Hanyang University, Korea, {syrilo, chajh, yjwon, sykang}@hanyang.ac.kr*
[2]*School of Electrical Engineering, Korea University, Korea, sryoon@korea.ac.kr*
[3]*School of Computer Science and Engineering, Dankook University, Korea, choijm@dankook.ac.kr*

Abstract[1]

Recently, NAND flash Memory based SSD (solid state drive) is widely used in desktop and enterprise severs as well as portable devices. SSD, which already has hundreds of gigabyte of capacity, is expected to has tera-scale capacity in a near future. Since contemporary high-end SSDs use page mapped FTL for better performance, they require tens or hundreds of megabyte of DRAM for mapping table store. However, since tera-scale SSDs need gigabytes of DRAM for mapping table store, the mapping table management itself is a new challenge for tera-scale SSDs. In this paper, we propose a novel caching-based mapping scheme, Polymorphic Mapping, for tera-scale SSDs, which subsumes wide range of mapping granularities from page to block or range.

1. Introduction

Since SSD, unlike HDD, has no mechanical parts, it has many merits such as low power consumption, low noise, high speed and shock resistance. Because of these characteristics, it is widely used in desktop PCs and enterprise servers as well as portable devices. Recently, due to the improved packing density and new types of NAND flash chip such as MLC, TLC and QLC, the capacity of SSDs increases continuously. Assuming the subsequent capacity improvement, it is expected that the tera-scale SSDs will be introduced to the market in a near future.

SSD requires a special software layer, FTL, which enables users to use SSD as an ordinary block device such as a hard disk. Lots of FTLs have been developed for a decade, which can be categorized into block mapping, hybrid mapping and page mapping FTLs. Among them, contemporary SSDs mainly use page-mapped FTL to increase the random write performance. However, original page-mapped FTL requires a large-sized DRAM for mapping table store [1]. For example, 128 GB SSD needs at least 128 MB DRAM for mapping table, which makes it easy to project the required DRAM space for future tera-scale SSDs. To remedy this problem, DFTL[2] proposed to caching only part of the mapping table in a fast volatile memory and store the entire mapping table in the Flash memory. DFTL uses a small-sized SRAM for caching space and exploits temporal locality of workload to maintain high cache hit ratio. However, it does not consider either the spatial locality of write accesses or caching space scalability. If the spatial locality is high, large number of mapping entries can be merged into much smaller number of entries, which can be more efficiently managed like in huge page table of operating systems or extents map in file systems. When the spatial locality is low, updated entries can be logged, as in database system, instead of updating mapping table entries in flash memory pages, instantaneously. Also, since the SRAM is such an expensive memory, it cannot be used for tera-scale SSDs which may require large sized cache space for entire working set store. Even if we use DFTL for large sized DRAM, instead of SRAM, its caching structure is not practical because of large search overhead.

2. Polymorphic Mapping

In this paper, we propose a novel mapping scheme for tera-scale SSDs, Polymorphic mapping, which exploits both the spatial locality of write accesses and cache space scalability. Figure 1 shows the data structures of Polymorphic mapping. Polymorphic mapping uses different mapping granularities according to the write pattern: Direct mapping, Extents mapping, and Page mapping. The entire logical address space of SSD is partitioned into a set of contiguous fixed-sized (4MB) regions of which mapping information is stored in a root data structure, Global Table Directory (GTD). If a region is written sequentially, the PPN field of the corresponding entry in GTD directly points to the physical address of its first data page in the flash memory, and the state bits are set to 00 (Direct mapping). For non-sequentially written regions, a mapping table (of which the address is stored in the PPN field) is assigned to each of them, and the state bits in GTD are set to 01. The initial form of the mapping table is Extents mapping. The number of entries in the Extents map increas-

[1] This work was supported by the IT R&D program of MKE/KEIT No. KI10035202, Development of Core Technologies for Next Generation Hyper MLC NAND based SSD.

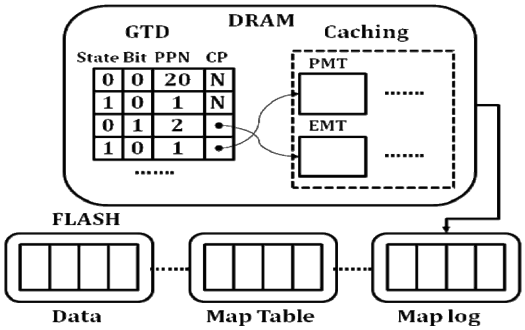

Figure 1 Structures of Polymorphic mapping

es whenever random write occurs to the region. If the size of Extents map table becomes larger than the predefined threshold (half of a page), the region transforms to a Page-mapped region, in which case the state bits in GTD becomes 10. In this way, Polymorphic mapping maintains three kinds of mapping structures according to the write pattern (sequential: Direct mapping, random: Page mapping, mixed: Extents mapping).

In pursuit of caching scalability, CP (cache pointer) which points either cached PMT (Page Mapping Table) or cached EMT (Extents Mapping Table) is used. CP field helps to translate address within O(1) regardless of the cache size. For update of a mapping entry, we use map log. An updated mapping entry is logged into a fixed location of flash memory, which is updated to the original mapping table, periodically.

3. Performance Evaluation

We performed trace-driven simulation to evaluate the performance of the Polymorphic mapping. We used four kinds of real world workloads: WI, WU-1, WU-2, and OLTP. WI was collected during installing windows and programs. WU-1 and WU-2 were extracted while running windows applications during one month. WI, WU-1 and WU-2 show high sequentiality while OLTP generates mainly random requests. We measured cache hit ratio and flash access overhead during address translation phase and normalized them to that of the DFTL.

In Figure 2, line graphs show cache hit ratio when cache size varies from 32KB to 2MB, and bar graphs represent flash read/write overhead when cache size is 512KB. As we can see from the figure, Polymorphic mapping outperforms DFTL in most cases. In particular, Polymorphic mapping shows far better performance than DFTL in sequential workloads. They show little performance difference in random workload, in which Polymorphic mapping behaves like a mere page mapping with caching. As cache miss invokes flash read/write operations, the cache hit ratio is inverse proportional to the address translation overhead.

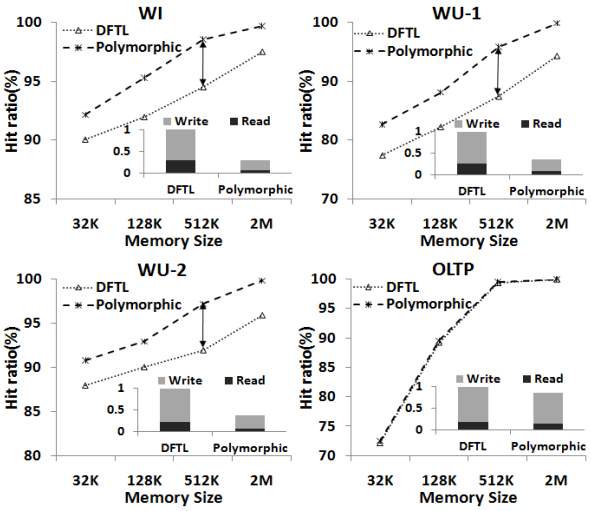

Figure 2 Performance comparison

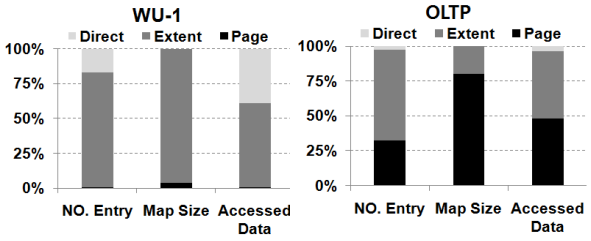

Figure 3 Analysis of Polymorphic mapping

Figure 3 shows the characteristics of the Polymorphic mapping in WU-1 and OLTP. In WU-1, 40% of data are accessed through only 20% of mapping entries (direct mapping). It shows why Polymorphic mapping outperforms DFTL in sequential workloads.

4. Conclusion

In this paper, we propose a Polymorphic mapping, which exploits different mapping granularities for different access patterns, for tera-scale SSDs. It naturally adapts well to the workload characteristics and outperforms by up to 67% in comparison to the DFTL during address translation phase.

References

[1] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., M. Manasse, and Panigraphy, R., Design Tradeoffs for SSD Performance. In Proceedings of the USENIX Annual Technical Conference (Boston, MA, June 2008). USENIX 2008.

[2] GUPTA, A., KIM, Y., AND URGAONKAR, B., DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In Proceedings of ASPLOS'09