How Many MLCs Should Impersonate SLCs to Optimize SSD Performance?

Wei Wang
San Diego State University
5500 Campanile Dr
San Diego, CA 92182
wei.wang5@sandisk.com

Wen Pan
San Diego State University
5500 Campanile Dr
San Diego, CA 92182
wenwen412@gmail.com

Tao Xie
San Diego State University
5500 Campanile Dr
San Diego, CA 92182
txie@mail.sdsu.edu

Deng Zhou
San Diego State University
5500 Campanile Dr
San Diego, CA 92182
dzhou.ustc@gmail.com

ABSTRACT

Since an MLC (multi-level cell) can be used in an SLC (single-level cell) mode, an MLC-based flash SSD typically uses a fixed small portion (called log partition) in the SLC mode to accommodate hot data so that its overall performance can be improved. In this paper, we show that a fixed capacity of a log partition without considering workload characteristics can lead to an unexpected overall performance degradation. Contrary to intuition, we notice that blindly enlarging the capacity of a log partition would also result in worse performance due to the increased garbage collection cost in a data partition, which serves cold data. How many MLCs should impersonate SLCs under a particular workload to achieve an optimized performance is still an open question. To answer this question, we first measure write costs on each partition and their impact on the overall performance of an SSD. Next, a hardware-validated write cost model is built. Based on the model, we demonstrate that for each workload there always exists an optimal partitioning scheme. Further, to verify the effectiveness of our workload-aware dynamic partitioning strategy, we implement an FTL (flash translation layer) called BROMS (Best Ratio Of MLC to SLC), which adaptively adjusts the capacities of two partitions according to the workload characteristics. Experimental results from a hardware platform show that BROMS outperforms a fixed partitioning scheme by up to 86%.

CCS Concepts

• **Information systems** → **Flash memory;** • **Hardware** → **External storage;** • **Software and its engineering** → *Secondary storage;*

MEMSYS 2016 October 3–6, 2016, Washington, DC, USA
© 2016 ACM. ISBN 978-1-4503-4305-3/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2989081.2989095

Keywords

NAND, Solid State Disk, MLC, SLC, Partitioning

1. INTRODUCTION

Emerging storage devices like NAND flash memory (hereafter, flash memory) based SSDs are being deployed on large-scale supercomputers such as Gordon [13] and Tsubame2 [1] due to their desirable features like high I/O performance and low energy consumption. Typically, flash memory devices fall into two categories, SLC and MLC, based on the number of bits that a memory cell can store. An SLC flash memory cell can store only 1 bit data, whereas 2 or more bits can be stored in an MLC memory cell. An SLC memory device is more robust and delivers a much better performance[1] [6,17]. For example, while a typical SLC flash memory can tolerate $\sim 100k$ program/erase (P/E) cycles with a $\sim 200\mu s$ programming delay, a 2-bit MLC can only survive $\sim 10K$ P/E cycles with a programming latency more than $1,000\mu s$ [6].

An MLC flash cell can work in an SLC mode by selectively using its fast pages [6,7] (see Section 2). In other words, an MLC flash cell can impersonate an SLC flash cell, and thus, obtain an SLC-like performance at the cost of 50% capacity loss [7]. Obviously, the performance-capacity trade-off is worthwhile considering that more than 3 times performance improvement and the increasingly lower capacity per dollar of MLC flash [6]. Inspired by this beneficial trade-off, previous work [4, 7, 8] proposes to break down an MLC SSD into a small log partition and a large data partition. In the log partition, an MLC acts as an SLC so that a higher performance can be delivered. In the data partition, MLC memory cells work in their normal mode to provide a higher density but lower performance. Popular data are dispatched to the log partition, whereas unpopular data are assigned to the data partition. Therefore, a log partition is also called a hot partition and a data partition is called a cold partition. In the remainder of this paper, log partition and hot partition as well as data partition and cold partition are used interchangeably. The overall effect is that an MLC-SLC partitioned MLC SSD can deliver performance close to that

[1]In this work we only investigate 2-bit MLC flash, which is currently the dominant type of MLC flash.

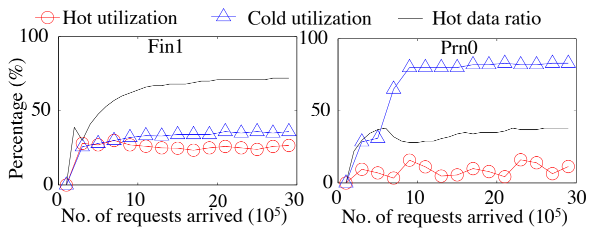


Figure 1: Partition utilizations in Fin1 and Prn0.

of an SLC SSD while keeping a capacity near to that of an MLC SSD. The capacity of a hot partition is usually small (typically 5% $\sim$ 20 %) and is fixed to minimize the capacity loss [4,7]. However, a fixed partitioning for all applications largely overlooks the varying workload conditions among different applications, which could significantly affect an SSD's performance.

Flash memory uses garbage collection (GC) to reclaim invalid pages [16]. A partition with a higher utilization (i.e., the ratio between the size of valid data and the capacity of the partition) tends to have a higher GC cost because a large number of valid pages need to be transferred to a new location. As a result, the overall performance degrades. To understand the impact of workload characteristics on the performance of an SSD with a fixed MLC-SLC partitioning, we conduct a group of experiments on an emulated MLC-SLC SSD under 16 real-world traces [2,12]. The total capacity of the SSD is set to 64 Gb, among which 10% is configured as a hot partition. If a request whose size is less than 4 KB, it is classified as a hot data [4]. Otherwise, it is viewed as a cold data[2]. We find that different workloads running on a fixed partitioning configuration exhibit distinctive partition utilizations (see the differences between the two traces in Fig. 1). Different utilizations lead to various levels of performance. The conclusion is that a fixed partitioning cannot fit all 16 workloads. Besides, even under a particular workload, partition utilization (i.e., an indicator of performance) may noticeably vary over time (see the Prn0 workload in Fig. 1). The implication is that for a particular workload a fixed partitioning results in an unstable performance along the time. Due to the space limit, we only provide results of Fin1 [2] and Prn0 [12] in Fig. 1.

Motivated by our discoveries as well as analyses on the previous work, in this paper we first build a write cost model based on an analysis of garbage collection overhead in two partitions. Next, an FPGA platform is built to validate the accuracy of the model. The validated write cost model reveals that for each workload there always exists a best partitioning scheme that can deliver an optimal performance. Based on the model, we further develop a *dynamic partitioning* method, which optimizes the overall performance of a partitioned SSD via adjusting the capacities of its two partitions on the fly. It performs periodic evaluation on an SSD's overall performance according to current workload characteristics and partition utilizations. The evaluation results are used to guide how a GC operation is performed and how to re-allocate two partitions so that the overall performance is maximized.

[2]Our workload-aware dynamic MLC-SLC partitioning method does not rely on a particular hot-cold data classification strategy. How to find an optimal hot-cold data classification strategy is out of the scope of this research.

Finally, to verify the effectiveness of the workload-aware dynamic partitioning method, we integrate it into a new FTL called BROMS (Best Ratio Of MLC to SLC), which adaptively adjusts the capacities of two partitions according to the workload characteristics. Experimental results from a hardware platform demonstrate that BROMS improves the performance of a partitioned SSD up to the optimal off-line performance of a fixed partitioning. In this research, we make the following contributions:

- We analyze the overall write cost of a partitioned SSD. A write cost model is built accordingly to reveal the relationship between an SSD's overall performance and partition utilizations.
- We use a hardware evaluation platform to evaluate the accuracy of the write cost model. We demonstrate that the errors between the values from the theoretical model and the results from the hardware platform are reasonable and the model is feasible.
- We propose a dynamic partitioning method to tune an SSD's partitioning toward an optimal performance on-the-fly. Multiple GC methods are provided to facilitate the adjustment of the capacities of two partitions.
- We develop a new FTL called BROMS that integrates the dynamic partitioning method. BROMS is implemented in a Linux kernel running on an SSD evaluation platform. We evaluate its performance on a hardware platform using 16 real-world traces. Experimental results demonstrate that BROMS can improve the overall performance in terms of mean response time by up to 86% compared with a fixed partitioning approach.

The rest of the paper is organized as follows. In the next section, we introduce the basic knowledge of flash memory and FTL. In Section 3, we analyze various costs. The write cost model is developed in Section 4. Section 5 evaluates the model on a hardware platform. The BROMS FTL is presented in Section 6. Its performance is evaluated in Section 7. Section 8 discusses the related work. Finally, we conclude our work with a summary and future direction in Section 9.

2. BACKGROUND

A flash memory consists of one or more flash dies with each having multiple planes. Each plane contains thousands of blocks and each block typically has 64 to 256 pages. The size of one page is usually 2-8 KB. Each memory cell of an MLC flash can store 2 bits, which are logically separated into a least significant bit (LSB) page and a most significant bit (MSB) page. While reads and writes are performed on page level, erase is carried out on a block. programming an LSB page is straightforward, whereas programming an MSB requires an extra read to its corresponding LSB page and a refined length of programming voltage step [6,17]. Hence, programming an LSB page is much faster than programming an MSB page [6,17]. Usually, an LSB page is also called a *fast page*, while an MSB page is referred to as a *slow page*. The number of fast pages is equal to the number of slow pages. Due to the significant performance difference between an LSB page and an MSB page, previous studies [6–8] suggest that the MSB-LSB performance discrepancy can be exploited to improve the performance and endurance of SSDs.

Flash memory cannot be programmed before it is erased, which is known as the *erase-before-write* problem. A flash translation layer (FTL) hides such problem by adopting an *out-of-place* method [5]. When a write request arrives, an FTL selects a pre-erased page for it and then marks the page with the old data as invalid. The invalid pages will be erased and reclaimed through a garbage collection operation. In addition, an FTL provides a wear-leveling mechanism to ensure that all cells in a flash memory experience a roughly equal programming/erase (P/E) cycle number. An FTL also manages a mapping table that translates a logical address from host to a physical address in flash memory.

3. ANALYSIS OF VARIOUS COSTS

A modern SSD normally provides some user-invisible over-provisioned blocks for bad block management and garbage collection. When an SSD is running out of free blocks, a GC process moves valid pages of victim blocks to the over-provisioned blocks and reclaims invalid pages. In a partitioned SSD, these over-provisioned blocks can be used by both partitions. Intuitively, a larger hot partition can provide a higher overall performance because most of data are served by fast pages. However, a large hot partition shrinks the capacity of the cold partition in an capacity-fixed SSD, which may increase the GC overhead of the cold partition. As a result, the overall performance may be reduced. To better understand the overall performance of a partitioned SSD, in this section we estimate various costs of a partitioned SSD from the partition utilization point of view.

3.1 Definitions and Assumptions

Since read traffic can be significantly reduced due to the extensive use of caching, write performance becomes the most important factor that affects the overall performance of many storage systems [18]. Especially, flash memory write latency can be 10 times higher than its read latency, which makes the impact of read on the overall performance of an SSD trivial. Furthermore, the read latency of SLC is similar to that of MLC [7]. Hence, to simplify our analysis we only model the write cost of a partitioned SSD. Experimental results on a hardware platform demonstrates that this simplified model is feasible and the errors between it and the hardware system is reasonable (see Section 5).

Fig. 2 shows a typical MLC SSD hot-cold partitioning. θ percent of total data are classified as hot data and sent to the hot partition. The hot/cold data classifier works ideally under a certain hot/cold data definition (e.g., data size). We assume that all flash memory devices are manufactured ideally so that there is no variation of write speed and reliability among all pages. A full page-mapping [5] is used in FTL to get an optimal address translation performance. Let us assume that each block contains N_p pages so that a block used in a hot partition consists of $N_p/2$ pages (i.e., the number of fast pages).

3.2 Garbage Collection Cost

Unlike the garbage collection cost of an unpartitioned SSD [5,18], in a partitioned SSD (see Fig. 2) each partition maintains its own garbage collection due to different space management schemes and data updating frequencies [7, 8]. Hence, the garbage collection cost models for an unpartitioned SSD [5,18] are no longer applicable. Furthermore, in a partitioned SSD, an adjustment of capacity of one partition results in a change of garbage collection cost on both partitions because the capacity increment of one partition comes from the capacity decrease in the other partition.

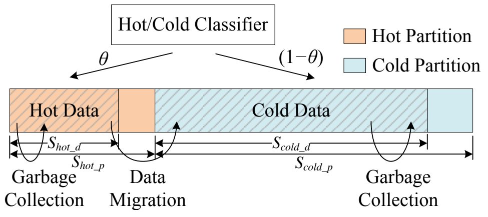


Figure 2: An example of partitioned MLC SSD.

To derive the model for a partitioned SSD, we first analyze the GC cost for each partition. Next, the interplay between the hot partition utilization and cold partition utilization is studied to obtain an overall GC cost model. The cost of a garbage collection operation can be modeled from different angles. Desnoyers built a GC cost model for an unpartitioned SSD from a write amplification perspective [5]. It does not take MLC-SLC partitioning into consideration [5], and thus, it is not suitable for this research. We study the GC cost in terms of partition utilization, which serves as a bridge between the two partitions. A high utilization results in an increased garbage collection overhead as more valid pages have to be copied out from a victim block during a garbage collection. Thus, the garbage collection cost increases along with capacity utilization.

We assume that in a steady state the utilization of the hot partition and cold partition are μ_h and μ_c, respectively. μ_h and μ_c can be defined as follows:

$$\mu_h = \frac{S_{hot\_d}}{S_{hot\_p}}; \mu_c = \frac{S_{cold\_d}}{S_{cold\_p}}, \tag{1}$$

where the S_{hot_d} and S_{cold_d} are the total sizes of hot data and cold data stored in each partition, respectively. The S_{hot_p} and S_{cold_p} are the capacities of hot partition and cold partition, respectively. If β percent of the SSD's total capacity is configured as hot partition, the capacities of the two partitions can be derived as: $S_{hot\_p} = \beta S_{tot}$ and $S_{cold\_p} = (1-\beta)S_{tot}$, where S_{tot} is the total storage capacity. Clearly, μ_h and μ_c are the average capacity utilizations.

In the hot partition, we assume that a block erase cost and a page copy cost are C_e and C_{hot_pc}, respectively. The C_{hot_pc} contains a page read and a fast page write. During a garbage collection, all the valid pages in a victim block have to be first copied out to a free block. Therefore, the garbage collection cost of hot partition can be derived as:

$$C_{hot\_gc} = \lceil \mu_h \cdot \frac{N_p}{2} \rceil \cdot C_{hot\_pc} + C_e. \tag{2}$$

Similarly, the garbage collection cost in the cold partition can be defined as:

$$C_{cold\_gc} = \lceil \mu_c \cdot N_p \rceil \cdot C_{cold\_pc} + C_e. \tag{3}$$

3.3 Page Write Cost

After a garbage collection in the hot partition, there are $\lfloor (1-\mu_h) \cdot \frac{N_p}{2} \rfloor$ available pages left in the free block. These pages can be used to serve upcoming write requests. The garbage collection cost is evenly distributed among these requests. Therefore, for each request in the host partition, its page write cost, C_{hot_pw}, is:

$$C_{hot\_pw} = \frac{C_{hot\_gc}}{\lfloor (1-\mu_h) \cdot N_p/2 \rfloor} + C_{pf}, \tag{4}$$

where the C_{pf} is the fast page write cost.

For the same reason, the page write cost for cold partition can be defined as:

$$C_{cold\_pw} = \frac{C_{cold\_gc}}{\lfloor (1-\mu_c) \cdot N_p \rfloor} + C_{pa}, \tag{5}$$

where the C_{pa} is the average page write cost of a block.

Clearly, the page write cost is a function of the utilization. Moreover, the page write cost in a partition may vary due to different workloads and partition capacities. The average utilizations, μ_h and μ_c, used in the above equations are the worst case value. This is because that a garbage collection typically chooses a victim block with the most invalid pages (i.e., the block with the lowest space utilization) to minimize it cost. Therefore, the utilization of a victim block, $\widetilde{\mu}_h$ or $\widetilde{\mu}_c$, could be lower than the worst case value (i.e., $\widetilde{\mu}_h \leq \mu_h$ or $\widetilde{\mu}_c \leq \mu_c$). The relationship between the utilization of a victim block and the worst case value has been derived as Equation (6) by [9]. It is proven to be true for both LFS and flash memory [9]. We reiterate it here just for completeness.

$$\mu_h = \frac{\widetilde{\mu}_h - 1}{ln(\widetilde{\mu}_h)}, \mu_c = \frac{\widetilde{\mu}_c - 1}{ln(\widetilde{\mu}_c)}. \tag{6}$$

By applying $\widetilde{\mu}_c$ and $\widetilde{\mu}_h$ to Equation (2), (3), (4), and (5) the page write cost in the hot partition and cold partition can be evaluated, respectively.

3.4 Data Migration Cost

It is understood that the hot/cold data classifier may make a mistake due to the fluctuation of a workload. A portion of cold or hot data could be incorrectly tagged. Cold data are rarely updated. Hence, when they are sent to a hot partition, the blocks that store them may not be updated as frequently as others. These infrequently used hot partition blocks decrease the available space in hot partition. Therefore, a data migration scheme is employed in the partitioned SSD so that the mis-dispatched data can be moved back to the right partition (see Fig. 2).

For a page of hot data that has been mis-distributed onto the cold partition, the mis-distributing issue can be easily corrected without a data migration process. The hot data can be left on the cold partition until its update arrives, at which moment the update can be simply directed to the hot partition. For a page of mis-dispatched cold data, a data migration process is required to move it back to the cold partition because the chance of updating it later is small. Therefore, the cost of a page migration only covers the overhead of migrating a page of cold data from the hot partition to the cold partition. Usually a time-stamp based detecting method is used to find a mis-dispatching [7]. A detecting process consumes processor time. Clearly, no matter how an SSD is partitioned, the cost of detecting a mis-dispatching is constant under a certain workload with a particular hot/cold classification standard. We do not include this constant software cost into the model as the speed of a modern processor is as much as three orders of magnitude higher than that of a flash write operation. Moreover, experimental results shown in Section 4.3 demonstrate that the amount of mis-dispatched data do not affect an optimal partitioning configuration. Therefore, the software detecting cost can be safely ignored.

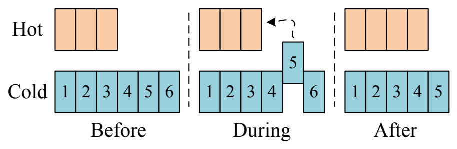


Figure 3: An example of dynamic partitioning.

In a data migration process a piece of data is first read and then it is written onto another partition. Thus, the data migration cost of an SSD, C_m, is derived as follows:

$$C_m = C_r + C_{cold\_pw}, \tag{7}$$

where C_r and C_{cold_pw} are the page read and write cost on cold partition.

4. THE WRITE COST MODEL

4.1 A Dynamic Partitioning Method

As having a fixed hot/cold partitioning for all applications performs sub-optimally, we propose a dynamic partitioning method that adjusts the capacities of two partitions on block-level to improve the overall performance of an SSD. One or multiple blocks in the cold partition can be dynamically transferred to the hot partition and vice versa. As a result, the capacities of the two partitions are re-sized (see Section 6). At any given time, only one transfer can happen. After a cold block joins the hot partition, it is immediately transformed to a hot block as it behaves like other blocks in the hot partition. A hot block only utilizes its fast pages, whereas a cold block uses its all pages. Fig. 3 illustrates an example of a cold-to-hot dynamic partitioning process, during which the block 5 in the cold partition is transformed to a hot block. This section examines the relationship between two partitions under the dynamic partitioning method.

The initial hot and cold partition capacities are βS_{tot} and $(1-\beta)S_{tot}$, respectively (see Section 3.2). Let the steady hot and cold partition utilizations be μ_h and μ_c, respectively. Assume that multiple blocks in the cold partition are transferred to the hot partition, and thus, the hot partition capacity is increased by ΔS. Meanwhile, the space of the cold partition is reduced by $2\Delta S$ (cold partition uses both MSB and LSB pages). After re-partitioning, the new hot partition utilization μ'_h and the cold partition utilization μ'_c can be derived by Equation (8) and (9), respectively:

$$\mu'_h = \frac{\beta \mu_h S_{tot}}{\beta S_{tot} + \Delta S}, \tag{8}$$

$$\mu'_c = \frac{(1-\beta)\mu_h S_{tot}}{(1-\beta)S_{tot} - 2\Delta S}. \tag{9}$$

The Equation (8) can be rearranged as:

$$\Delta S = \frac{\beta \mu_h S_{tot} - \beta \mu'_h S_{tot}}{\mu'_h}. \tag{10}$$

Therefore, the relationship between μ'_h and μ'_c can be derived by combining Equation (9) and (10):

$$\mu'_c = \frac{(1-\beta)\mu_h \mu'_h}{(1-\beta)\mu'_h - 2\beta\mu_h + 2\beta\mu'_h} \tag{11}$$

The Equation (11) illustrates the relationship of changed partition utilizations in a dynamic partitioning process. The partitioning process, however, must follow one restriction: at any given time, the capacity of a partition must be larger than its data size. For example, as shown in Fig. 2 the smallest hot partition capacity is S_{hot_d}. Hence, the lower and upper bound of μ'_h are:

$$\frac{2\beta\mu_h}{1+\beta-(1-\beta)\mu_c} < \mu'_h < 1. \quad (12)$$

4.2 Write Cost Model

Assume that θ percent of total data are classified as hot data. Also, assume that λ percent of classified hot data are mis-dispatched. By combining equations (4), (5), and (7), the write cost of an patitioned SSD can be represented as Equation (13), which is a function of μ'_h and μ'_c.

$$C'_{overall} = \theta C'_{hot\_pw} + (1-\theta)C'_{cold\_pw} + \lambda\theta C'_m \quad (13)$$

After applying Equation (11) to Equation (13), $C'_{overall}$ becomes a function of μ'_h. Note that Equation (13) is the write cost model that will be used to derive an optimal partitioning under a certain workload. For example, assume that under a certain workload 30% data are hot data (i.e., $\theta = 0.3$) and the proportion of mis-dispatched data is 1% (i.e., $\lambda = 0.01$). Also, assume that in a steady state the initial hot partition utilization and cold partition utilization are all 0.8 (i.e., $\mu_h = 0.8$ and $\mu_c = 0.8$). Thus, the ratio of hot partition capacity to cold partition capacity is 3/7. According to Equation (12), μ_h can be changed from its lower bound 0.65 to its upper bound 1. For an MLC SSD (see the legend of Fig. 4 for its parameters), its overall write cost is shown in Fig. 4. It is obvious that the write cost of the SSD exhibits a bathtub-shaped curve as μ_h changes. An optimal point that delivers the lowest write cost is found when μ_h is 0.87. From left to right, μ_h increases from 0.65 to 1 (i.e., the hot partition capacity is continuously shrunk for the size of hot data remains unchanged). Near its lower bound (i.e., 0.65) almost all free blocks are given to the hot partition. As a result, due to the lack of free blocks the cold partition suffers from an extremely high GC overhead, which increases the overall write cost. When μ_h increases from 0.65 to the point P_1, free blocks are transferred from the hot partition to the cold portion, which lowers the GC overhead in the cold partition. As the GC overhead of the cold partition is the major contributor to the overall write cost in this μ_h range, the overall write cost decreases quickly. From point P_1 to P_2, the overall write cost changes gently because the two partitions both have enough free blocks to perform garbage collections. Therefore, partitioning does not noticeably affect the overall write cost. The width between P_1 and P_2 will be shortened if more data are stored because the number of free blocks is reduced. After the point P_2 the GC overhead in the hot partition increases rapidly because most of free blocks have been transferred to the cold partition.

4.3 Model Feasibility Checking

To find the optimal point by the write cost model, parameters including hot data ratio, partition utilization, partition capacity, and flash memory specifications are essential. For a particular SSD, the flash memory specifications are constant. Other parameters, however, change over time. Hence, an SSD must be able to keep track of its working status so that the parameters can be dynamically obtained. Status tracking can be simply done by modifying an FTL.

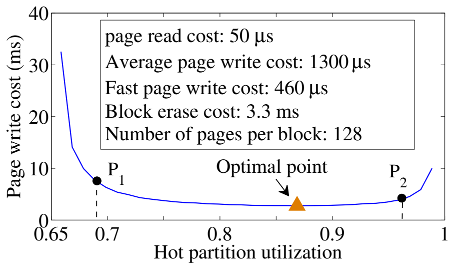


Figure 4: An example of overall write cost.

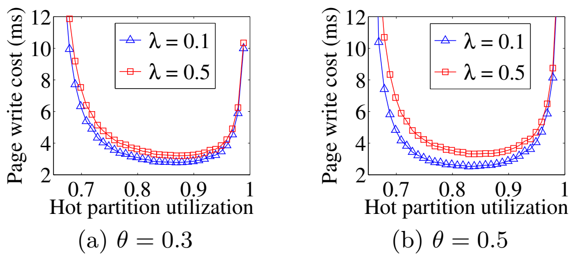


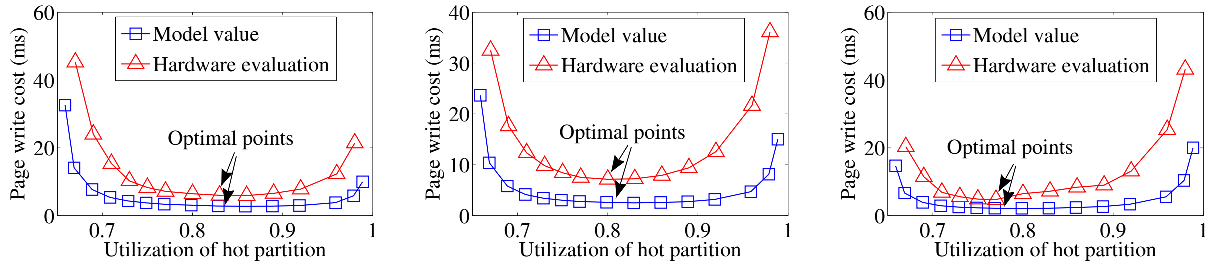
Figure 5: Impact of mis-dispatched data.

Intuitively, when the total capacity of an SSD is fixed increasing its capacity of the hot partition should always lead to a better overall performance because more number of fast pages can be used. However, for a workload with a small size of hot data (e.g., 30% is hot data as shown in Fig. 4), our write cost model shows that increasing the hot partition capacity results in a performance degradation, which is counter-intuitive. The reason is two-fold. First, giving the hot partition more free blocks than needed cannot further improve its performance due to a small size of hot data. Secondly, enlarging the capacity of the host partition shrinks the capacity of the cold partition, which reduces the number of free blocks that it can use. As a result, the GC overhead in the cold partition noticeably increases. Overall, the SSD's performance is degraded.

Mis-dispatching degrades the overall performance because extra cost is needed to perform data migration. We vary the portion of mis-dispatched data in hot partition and evaluate their impact on page write cost under two different hot data ratios (see Fig. 5). A larger value of λ indicates more mis-dispatched cold data. Two conclusions can be directly drawn from Fig. 5: (1) The partitioning that delivers the lowest page write cost is not affected by the amount of mis-dispatched data. When $\theta = 0.3$ the optimal points are all at the place where μ_h is equal to 0.87. When $\theta = 0.5$ the optimal points are all found when μ_h is 0.82. (2) The impact of mis-dispatched data is larger when μ_h is smaller. For example, in the case of $\theta = 0.3$ as the amount of mis-dispatched data enlarges the write cost only increases by 4.9% when μ_h is 0.96, whereas the write cost increases by 18.1% when μ_h is 0.68.

5. MODEL VERIFICATION

5.1 Evaluation Platform



(a) Results under ST1. (b) Results under ST2. (c) Results under ST3.

Figure 6: Performance comparisons between model value and evaluation results.

Table 1: Characteristics of synthetic workloads.

Name	*Total Req.*	*Updates*	*≤ 4 KB Req.*
ST1	10,485,233	80%	30%
ST2	10,485,233	80%	50%
ST3	10,485,233	80%	70%

The evaluation platform consists of an FPGA parent board [19] and a flash daughter board [3]. The daughter board is designed to connect multiple raw flash devices to the FPGA board with maximum 4 independent channels. The performance and capacity of an SSD can vary under different hardware configurations. Since we only evaluate the accuracy of our model, in our experiments a 64 Gb MLC flash memory [11] on one channel is used to simplify and accelerate evaluation process. A system including an Microblaze processor and an ONFI 2.0 compatible flash controller is implemented on the FPGA. Further, a Linux with kernel version 3.0 is installed on top of it. A flash controller driver that provides I/O interfaces is implemented. Our evaluation platform is essentially an SSD emulator because a software FTL can be integrated so that it can manage raw flash chips through the controller. The well-known ComboFTL [7] that manages partitioned SSDs is implemented.

5.2 System Configurations and Workloads

10% of total capacity is reserved for garbage collection and bad block management. The rest part of the flash is broken down into two partitions. we assume that by default 30% of the rest part is used by the hot partition. We also assume that in a steady state each partition is used 80% (i.e., utilization is 0.8).

ComboFTL distributes requests according to their sizes. As suggested by [4, 7], 4 KB is used in our experiments to classify hot and cold data. Three synthetic workloads (see Table 1) are generated to mimic different I/O scenarios.

5.3 Evaluation Results

Fig. 6 shows the hardware evaluation results and the model prediction values under the three workloads. Two observations can be made: (1) Results given by the hardware platform exhibit the same trend in performance change as that of model values. The overall system performance decreases when the hot partition capacity is either increased or decreased; (2) The hardware evaluation results are consistently larger than the values given by the model. The gap between the two curves becomes larger when μ_h leaves the optimal point that delivers the lowest page write cost. The differences between the model values and the evaluation results are mainly due to the software cost on the evaluation platform. An FTL manages several data structures in memory and regulates multiple functionalities like garbage collection. When the hot partition capacity is shrunk, the garbage collection overhead is increased as more valid pages need to be relocated to a new place. Accordingly, the associated software cost is increased, which further enlarges the gap between the two curves. Similarly, the gap is also enlarged when the hot partition capacity is increased.

The optimal point of the hardware evaluation curve and the optimal point of the model curve are at the same μ_h value in all the three scenarios. Therefore, the write cost model is accurate and can be used to find a partitioning configuration that delivers an optimal performance. For different FTLs and systems, the errors between the real performance and the model may vary. This, however, has no impact on finding the optimal point by the model.

6. THE BROMS FTL

In this section, a dynamic partitioning method that employs the write cost model is integrated into an FTL called BROMS (Best Ratio of MLC to SLC) to help an SSD achieve its optimal performance under various workloads. In BROMS, we choose the page-mapping table as an example to manage the two partitions in an MLC SSD. However, other types of mapping methods can also be used. To separate the two partitions, an FTL has to acquire prior knowledge about the layout of fast and slow pages within a block. Since such layout is fixed when a chip is designed and does not change over time, it is easy for an SSD designer to configure the layout information in an FTL.

The general architecture of BROMS is similar to that described in [7, 8]. Thus, we will focus on the enhancement parts of BROMS and omit descriptions of already known implementation techniques such as bad block management and wear-leveling control. Fig. 7 shows the high-level architecture of BROMS. Each request to the SSD enters the *request queue*. Next, the *request classifier* determines which partition is used to serve a request according to a request classification method. Note that the hot/cold data classification strategy is out of the scope of this research because the model does not depend on it. In BROMS a size-based classification strategy [4, 7] is applied as an example. Other techniques to determine the popularity of a request can also

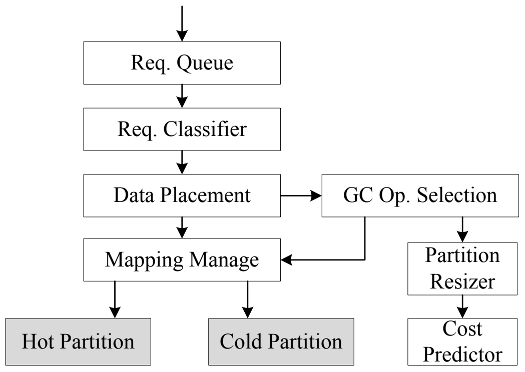


Figure 7: Architecture of BROMS.

be used in this FTL.

BROMS manages the capacities of two partitions in a dynamic way. Adjusting the capacity of a partition is simply done by changing the maximum number of blocks that the partition can use. For each partition, BROMS maintains a partition head pointer, which points to a list of blocks. The length of the list determines the capacity of a partition. Switching a block from cold partition to hot partition can be done by deleting a block from the cold partition linked list and then inserting it to the hot partition linked list. The *data placement* module consults with the internal data structures such as mapping table to decide where to place a request and whether a GC operation should be invoked. If the size of free space in a partition reaches a predefined threshold, a GC is invoked to reclaim used blocks.

The *GC operation selection* module is responsible for executing GC operations. Conventionally, a GC operation chooses the block that contains the largest number of invalid pages as a victim block to minimize its overhead. When several blocks happen to have the same number of invalid pages the block that has experienced the smallest number of erase operations is chosen to lower the discrepancies of P/E cycles among blocks. Each of the two partitions has its own GC threshold and their GCs are invoked independently (see Fig. 2). Unlike a conventional GC, BROMS introduces two novel modules, *partition resizer* and *cost predictor*, to help an SSD perform GC operations judiciously. A GC operation consists of two processes: valid data moving and block erasing. After the victim block is erased, it is put back to the free block pool on the same partition to accommodate upcoming requests. In order to re-partitioning an SSD on-the-fly, three new types of GC operations are provided to replace the traditional "*intra-partition*" GC: (1) A GC operation does not reclaim any used blocks. It simply grabs an erased block from the other partition. In this scenario, in each partition its capacity and utilization are all changed. (2) Valid data in the victim block are moved to the other partition. As a result, the utilizations in two partitions will be changed after the GC. However, the capacity of each partition remains unchanged. (3) A GC operation moves valid data within one partition and does not change the capacities of two partitions. This just likes a conventional GC operation and does not change the utilization of each partition.

The basic idea of GC selection is that before a GC is executed an after-GC performance evaluation for each GC option will be performed. The one that gives the lowest write cost will be chosen and executed. The *partition resizer* estimates the above three choices by consulting with the *cost predictor* module. The write cost model illustrated in Section 4 is implemented in the *Cost Predictor*. It takes current space utilizations as well as hot/cold data ratio as inputs and estimates the performance change based on the choice of a GC operation. After comparing the estimated costs of the three GC choices, the *GC operation selection* chooses the GC operation that leads to a better performance.

Table 2: Flash specifications [11].

Parameter	*Value*
Page size	8,192 + 448(OOB) bytes
Pages per block	256
Blocks per plane	2,048
Planes per die	2
Chips per channel	1
Channel number	2
Read latency	$75\mu s$
Write latency	$1{,}300\mu s$
Erase latency	$3.8ms$

7. EVALUATION OF BROMS

7.1 Experimental Setup

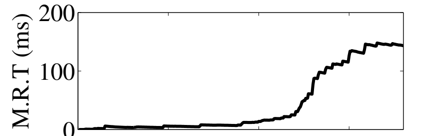
To fully understand the impact of the dynamic partitioning method on performance, we implement BROMS in an SSD emulator, which is built on the Xilinx xupv5-lx110t FPGA evaluation board [19]. The FPGA board connects to a 2-channel flash daughter board. One MLC flash device [11] is attached to a channel. Details of the platform is presented in Section 5.1.

In our experiments, we use 8 GB MLC flash memory devices [11], which are the largest parts in hand at the time of this research. We reserve 10% of the total number of blocks to form an over-provisioned space. Initially, 30% of the rest flash capacity is allocated for the hot partition, while others form the cold partition. The detailed flash specifications is shown in Table 2.

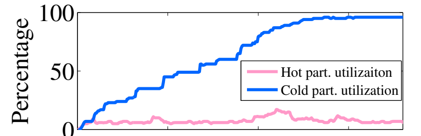
The ComboFTL [7] serves as a baseline and is also implemented. Sixteen real-world traces are selected to evaluate the two FTLs. Fin1 and Fin2 are taken from OLTP applications running at two large financial institutions [2], whereas the other 14 traces that have different enterprise workload patterns are from the MSR Cambridge traces [12]. Although most of the traces are collected from storage systems with a large capacity, their footprints are small. Further, we use a *modulo* function to re-map their addresses to our platform's address space so that each trace can fit in our evaluation platform. A size based data classification method is used and its default value is set to 4 KB.

7.2 Experimental Results

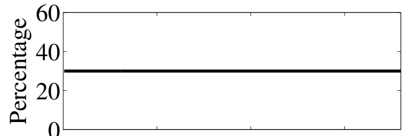
Fig. 8 shows the performance of the two FTLs in terms of mean response time. All the results are normalized to that of ComboFTL. It is clear that BROMS outperforms ComboFTL under almost all workloads except in the case of *hm_1*, *mds_0*, and *mds_1*. Under the *hm_0* and *prn_0* workloads, BROMS decreases the mean response time by 82.5% and 86.3%, respectively. In a steady state, the utilizations of cold partition in ComboFTL under *hm_0* and *prn_0*



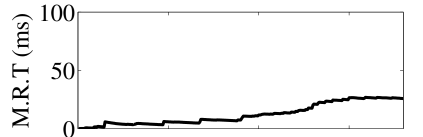
(a) ComboFTL MRT.



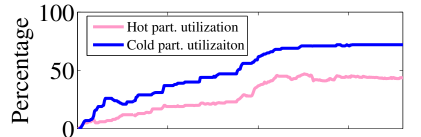
(b) ComboFTL utilization



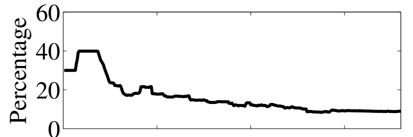
(c) ComboFTL hot size



(d) BROMS MRT.



(e) BROMS utilization



(f) BROMS hot size

Figure 9: A comparison between ComboFTL and BROMS under *hm_0*.

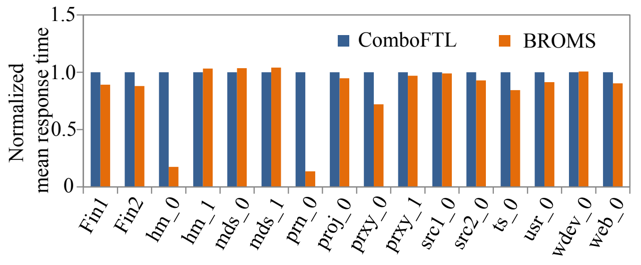


Figure 8: Performance comparisons between the two FTLs.

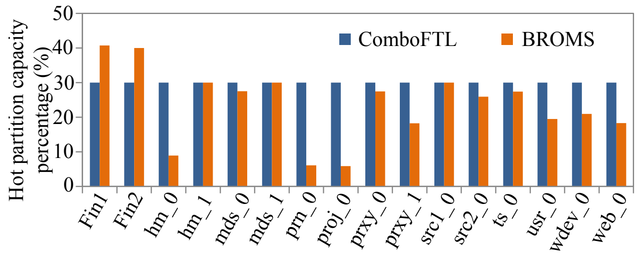


Figure 10: Hot partition capacities under the two FTLs.

are 96% and 83%, respectively, whereas the utilizations of hot partition is 0.06% in *hm_0* and 0.09% in *prn_0*. The high utilization in cold partition results in a high garbage collection overhead even though a lot of empty blocks still exist in hot partition. BROMS, however, is aware of such variation and intentionally transfers empty blocks from hot partition to cold partition so that the garbage collection overhead in cold partition can be reduced.

For the *hm_1*, *mds_1*, and *src1_0* traces, the footprints of them are too small so that no any garbage collection is invoked. In this case, the BROMS works almost in the same way as ComboFTL and no big difference in performance is shown between BROMS and ComboFTL. For *mds_0*, updates exhibit a very high spatial locality. Hence, almost no valid pages need to be moved during a garbage collection. The number of garbage collections is the only factor that affects the overall performance. In this scenario, the number of garbage collections is almost the same under the two FTLs, and thus, the performance of them is similar.

Fig. 10 illustrates the capacities of the hot partition in a steady state in ComboFTL and BROMS, respectively. For *Fin1* and *Fin2*, the capacity of hot partition increases to accommodate a larger hot data size (in *Fin1* and *Fin2*, more than 70% of requests are classified as hot data). In *hm_0*, *prn_0* and *proj_0*, the capacity of hot partition is largely shrunk (i.e., from 30% to 8.9%, 6%, and 5.8%, respectively) so that the GC overhead in cold partition is reduced.

We take *hm_0* as an example to illustrate how the dynamic partitioning method can improve the overall performance of an SSD. Fig. 9 compares the performance, partition utilization, and hot partition capacity between ComboFTL and BROMS. In ComboFTL, the partition capacity remains unchanged throughout the entire experiment. The hot partition utilization stays at a low level (around 10%). The mean response time increases moderately at the begining of the experiments. However, as the valid data in cold partition increases consistently, the cold partition utilization is pushed to 96% (see Fig. 9b). In this scenario, garbage collection in cold partition has to move a large number of valid pages even through plenty of empty blocks exist in the hot partition. Hence, the mean response time increases rapidly as shown in Fig. 9a.

Fig. 9f shows the change of hot partition capacity in BROMS. When the utilization of cold partition is very low, BROMS intentionally moves some blocks from cold partition to hot partition so that the garbage collection overhead in hot partition is reduced. As the cold partition utilization continuously increases, blocks in the hot partition are transferred to the cold partition to reduce garbage collection overhead. Hence, the utilization of the cold partition keeps at a reasonable level. Although the hot partition utilization is increased compared with ComboFTL, the overall GC overhead decreases. Thus, the mean response time keeps at a lower level (see Fig. 9d).

The overhead of a GC in the cold partition is much larger than that of a GC in the hot partition. The hot partition only uses fast pages so that 50% of its pages are empty. Besides, a fast page has a better programming performance, which reduces the overhead of a page moving. Hence, the dynamic partitioning method is more effective in the scenario that the cold partition utilization is originally high, such as *hm_0* and *prn_0*. In *Fin1* and *Fin2*, most of their data are classified as hot data. Therefore, enlarging the hot partition only improves performance by 10.7% and 12%, respectively. (see Fig. 8).

Using only fast pages in an MLC can also improve its endurance [8]. Therefore, a partitioned SSD is more reliable than an unpartitioned SSD. However, the endurance improvement in a hot partition leads to an uneven wear out between the two partitions [8]. The problem is addressed by a dynamic mapping method [8], which can be readily integrated into BROMS.

8. RELATED WORK

In order to gain the complementary merits of hard disk drives (HDDs) and flash SSDs, Oh *et al.* proposed an HDD-SSD hybrid system [14]. By analyzing the garbage cost in an SSD, they found that balancing the two spaces appropriately helps improve the performance. In this research, we use a similar method to analyze GCs cost in an MLC-based SSD. Due to the decreasing price of an SSD, SSDs are increasingly replacing HDDs in various computing platforms. There are two types of flash memory: MLC and SLC. Compared with MLC flash, SLC flash has a much better I/O performance, endurance, and reliability. However, it is more expensive than SLC flash [6]. Therefore, how to achieve an SLC-like performance from an MLC SSD becomes an attractive research topic. Grupp *et al.* proposed a scheme called RTF (return to fast) that uses fast pages for programming while slow pages are only used for garbage collection [6]. Unlike [6],

How to achieve an SLC-like performance from an MLC SSD becomes an attractive research topic. Chang proposed a hybrid SSD combining one SLC device and one MLC device [4]. While the SLC part serves frequent updates, the MLC part receives less popular data. However, since each flash device has a fixed capacity, a dynamic partitioning between the two devices is not applicable.

As modern MLC flash can work in an SLC mode that delivers an SLC-like performance, recently a few research projects [7, 8] have been conducted on an MLC-based SSD where part of its MLCs act as SLCs. It can be viewed as an MLC-SLC logically partitioned SSD. Im *et al.* developed the ComboFTL that manages an MLC-based SSD in two partitions (i.e., an SLC partition and an MLC partition). However, it only used a fixed partitioning approach without considering changing workload characteristics [7]. On the other hand, Jimenez *et al.* focused on how to improve the lifetime of an MLC-SLC partitioned SSD [8]. Still, they adopted a fixed partitioning method.

Park *et al.* [15] and Lee *et al.* [10] propose an MixedFTL and FlexFS for SLC/MLC combined flash memory, respectively. These two studies exploit the hotness of a piece of data. Data will be first written to the SLC region and then is moved into MLC region if there is no updates in a long time period. Hence, SLC partition suffers from a heavy write workload and a high migration cost.

9. CONCLUSIONS

MLC-based SSDs have been widely used in supercomputers [1, 13]. In order to gain a better performance, a portion of an MLC SSD is used in an SLC mode and serves hot data only [7, 8]. To understand the impact of a partitioning method on the performance of an SSD, we first conduct a group of experiments on an emulated MLC-SLC partitioned SSD using 16 real-world traces. We find that different workloads running on a fixed partitioning configuration lead to various levels of performance. Besides, even under a particular workload, performance may noticeably vary over time. The conclusion is that a fixed partitioning normally leads to an inferior performance. To address this problem, we analyze the garbage collection cost in both partitions and then build a write cost model for an SSD. We demonstrate that for each workload there always exists a best partition configuration that offers optimal overall performance. Next, we develop a dynamic partitioning method to adjust the capacities of two partitions on-the-fly so that the performance can be tuned towards an optimal point. Finally, we implement a new FTL called BROMS to demonstrate the effectiveness of the dynamic partitioning method. Hardware-based experimental results show that BROMS improves the overall performance by up to 86% compared with ComboFTL.

In current work, we apply a fixed step size (i.e., one block) to change partition capacity, which results in a slow response to the cost change. In the future work, we will analyze the impact of step size on the performance. Further, how to fast respond to the data size change will also be addressed.

10. ACKNOWLEDGMENTS

We would like to thank Steven Swanson's Non-volatile Systems Laboratory for providing us with the Ming II flash daughter board. This work is sponsored by the U.S. National Science Foundation under grant CNS-1320738.

11. REFERENCES

[1] Top500 supercomputer sites. URL http://www. top500. org/ (2014)

[2] Bates, K., McNutt, B.: OLTP application I/O (2007), http: //traces.cs.umass.edu/index.php/Storage/Storage, last checked: 05.09.2014

[3] Bunker, T., Wei, M., Swanson, S.J.: Ming ii: A flexible platform for nand flash-based research. Tech. rep., Department of CSE, UCSD (2012)

[4] Chang, L.P.: Hybrid solid-state disks: Combining heterogeneous nand flash in large ssds. In: Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific. ASPDAC'08 (March 2008)

[5] Desnoyers, P.: Analytic models of ssd write performance. Trans. Storage 10(2), 8:1–8:25 (2014), http://doi.acm.org/10.1145/2577384

[6] Grupp, L.M., Davis, J.D., Swanson, S.: The harey tortoise: Managing heterogeneous write performance in ssds. In: USENIX ATC. Berkeley, CA (2013)

[7] Im, S., Shin, D.: ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. Journal of Systems Architecture 56(12) (2010)

[8] Jimenez, X., Novo, D., Ienne, P.: Software controlled cell bit-density to improve nand flash lifetime. In: Design Automation Conference,. DAC'12 (June 2012)

[9] Kwon, H., Kim, E., Choi, J., Lee, D., Noh, S.H.: Janus-ftl: Finding the optimal point on the spectrum between page and block mapping schemes. In: Proceedings of ACM EMSOFT. pp. 169–178. New York, NY, USA (2010)

[10] Lee, S., Ha, K., Zhang, K., Kim, J., Kim, J.: Flexfs: a flexible flash file system for mlc nand flash memory. In: USENIX ATC, USA (2009)

[11] Micron: Nand datasheet, MT29F64G08CEACA (2014), http://www.micron.com/products/nand-flash/mlc-nand

[12] Narayanan, D., Donnelly et al., A.: Write off-loading: Practical power management for enterprise storage. Trans. Storage 4(3), 10:1–10:23 (Nov 2008)

[13] Norman, M.L., Snavely, A.: Accelerating data-intensive science with gordon and dash. In: Proc. of the 2010 TeraGrid Conf. p. 14. ACM (2010)

[14] Oh, Y., Choi et al., J.: Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In: Proceedings of the 10th USENIX conference on File and Storage Technologies. FAST'12 (2012)

[15] Park, S.H., Park et al., J.W.: A mixed flash translation layer structure for slc-mlc combined flash memory system. Proc. of SPEED 8 (2008)

[16] Wang, W., Xie, T.: Pcftl: A plane-centric flash translation layer utilizing copy-back operations. IEEE Transactions on Parallel and Distributed Systems 26(12), 3420–3432 (Dec 2015)

[17] Wang, W., Xie, T., Zhou, D.: Understanding the impact of threshold voltage on mlc flash memory performance and reliability. In: Proceedings of the 28th ACM International Conference on Supercomputing. ICS '14, New York, NY, USA (2014)

[18] Wang, W., Zhao, Y., Bunt, R.: Hylog: A high performance approach to managing disk layout. FAST'04, vol. 4 (2004)

[19] Xilinx: Xilinx xupv5-lx110t development system (2014), http://www.xilinx.com/univ/xupv5-lx110t.htm

All links were last followed on October 5, 2015.