Performance Models of Flash-based Solid-State Drives for Real Workloads

Simona Boboila
Northeastern University
360 Huntington Ave.
Boston, MA 02115
simona@ccs.neu.edu

Peter Desnoyers
Northeastern University
360 Huntington Ave.
Boston, MA 02115
pjd@ccs.neu.edu

***Abstract*—There is a wide gap between the potential performance of NAND flash-based solid state drives (SSDs) and their performance in many real-world applications; understanding this gap requires knowledge of their behavior and internal algorithms for various workloads. We develop analytic models for two commonly-used Flash Translation Layer (FTL) algorithms, as used in SSDs, as well as a methodology for applying these models to real-world workloads. We demonstrate the accuracy of these models via simulation, extend this approach to incorporate measurement-based approximations when detailed parameters are unknown, and validate this methodology against real devices.**

I. Introduction

In the last decade NAND flash has grown into a major industry, most recently making inroads in the computing market in the form of solid state drives, or SSDs. Although the performance characteristics of flash devices themselves are fairly straightforward, we know little of how they perform when combined with internal flash translation layer (FTL) algorithms in SSDs and similar devices. Without such information it is difficult to design systems to use these devices most effectively, while efforts to emulate a perfect high-performance device on top of flash so we may ignore these characteristics have to date been imperfect at best.

Most existing work comprises evaluation of proposed FTL algorithms and measurement of available devices[1]. FTL proposals (e.g. [3]–[5]) typically provide trace-driven simulation results for various real-world scenarios, demonstrating their performance vs. previous algorithms. However, with the exception of Park et al. [6] there is little analysis of conditions affecting performance, and thus no guidance as to how to tune workloads for best performance.

On the other end of the spectrum are black box measurements and benchmarks of real devices. An example is uFLIP [7], a set of benchmarks which attempts to capture relevant aspects of SSD behavior. Although heuristically generated, these tests measure key characteristics of SSDs; however they leave unanswered questions such as how to predict real-world performance from benchmark results, or to modify traffic so as to improve throughput.

We approach this problem from the inside out, starting from well-known FTL algorithms, deriving analytic expressions for their behavior under stationary workloads, and validating these models via simulation. We then present and validate a method for applying these models to real, non-ideal workloads. Finally, we extend this methodology to extrapolate from calibrated measurements and thereby predict performance for arbitrary workloads, giving results for both simulation and real devices.

Our analysis targets performance models of SSDs for real workloads. This task is complex due to various workload characteristics and FTL algorithms. Using a piecewise decomposition, we start from the simpler case of well-behaved traffic given specific FTL algorithms. Building up from this, we extend the model to real workloads running on real SSDs.

II. Background

We begin by concisely defining the problem which a flash translation layer must solve, and describing the FTLs analyzed. We omit lower-level details of flash technology; the reader may refer to other works [8], [9] for more details.

Modern NAND flash is organized into pages of 2K to 8K bytes, grouped in erase blocks of $B = 64$, 128, or even 256 pages. Each page must be read as a unit with latency of about 25-200μs, and little penalty for random reads. Pages must be written as a unit, typically taking 4 or 5 times longer than a read, and pages in an erase block must be written from first to last. Pages cannot be over-written; instead the entire erase block must be cleared or *erased* in an operation taking several write times; after erasure the pages in the block may be rewritten.

The role of a flash translation layer is to provide a rewritable disk-like block abstraction on top of these devices. This is done by a mutable mapping of logical addresses to internal, physical pages. In particular, repeated writes to the same logical location occupy different physical addresses, with the logical-to-physical mapping changing for each write.; the old or *invalid* copies will eventually be erased as a block and

[1]But note Hu [1] and Ben-Aroya & Toledo [2], discussed in Section V.

978–1–4577–0428–4/11/$26.00

re-used. In essence an FTL is a log-structured file system, optimized to support a single fixed-length file representing the logical volume seen by the host.

The straightforward (and optimal [1]) FTL mechanism maintains a full map from logical to physical pages. This can be expensive, requiring 32 to 64MB of RAM for a 64GB device (even more for older small-page devices) and long power-on delays to reconstruct such a large map.

Instead, FTLs typically rely on a coarser-grained *block map*, at a granularity of one or more erase blocks. Much like the mapping of a page number and direct use of the offset in a virtual memory system, this approach maps a logical block to a physical block, but relies on the pages within the physical block being in logical order. Since writes are performed one page at a time, this order cannot be perfectly preserved; instead, writes are performed in separate *log blocks* which are page-mapped, and serve as look-aside caches in front of the block-mapped *data blocks*.

In the first scheme we examine, Hybrid Log Block [3] (also referred to as BAST in [4]), each log block is associated with exactly one data block; pages may be written in this log block in arbitrary order. On the first write to a data block a log block is allocated and the page written, *invalidating* the corresponding data block page. The number of log blocks is limited; when this limit is reached an in-use one must be *evicted* for this allocation to succeed. When a log block becomes full or is evicted, it will be merged with its corresponding data block, resulting in a fully ordered and valid data block again. In the best case the log block is written consecutively from beginning to end, and can be switched into place with no overhead; for non-sequential or partial writes it is necessary to *merge* valid pages from the log and data blocks into a third block, possibly incurring significant overhead. In particular, if a log block is merged after n pages have been written, this will result in a *write amplification factor*, or ratio of internal physical writes to external logical writes, of B/n for block size B. For randomly distributed small writes, virtually every write will evict a log block containing a single page, for a write amplification factor of B, or as high as 256.

Fully Associative Sector Translation (FAST) [4] attempts to maximize the utilization of log blocks by allowing pages from multiple data blocks to share a log block. The log blocks are arranged in a queue; at the tail, any pages not invalidated while in the queue are merged with their corresponding data blocks. This performs well for workloads with spatial locality, as pages are invalidated while on the log block queue; in effect the queue functions as a fully associative cache in front of the data blocks (vs. BAST, where associativity is only within single-block sets.). For random traffic, however, very few pages will be invalidated in the queue, resulting in a write amplification factor of nearly B. If all log blocks are empty or invalid, then random writes will be accepted at full speed until the queue is full; at this point expensive merges will be required to free log blocks for new writes, and performance will plummet.

Other FTLs proposed to date include Park's N/K adaptive algorithm [6], which combines elements of Hybrid Log Block and FAST, and Superblock [5], which uses page mapping within groups of data blocks. Analysis of these and other FTLs is not reported in this work.

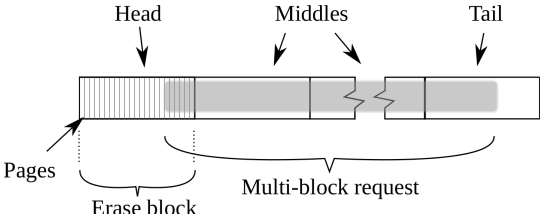


Fig. 1: Division of a request into head, middle, and tail fragments.

Device parameters:	
V_P	number of physical blocks
V_L	number of logical blocks
f	free space ratio ($\frac{V_P - V_L}{V_L}$)
B	erase block size in pages
Workload parameters:	
w	mean write length, in pages
T	total number of writes in pages
Other variables:	
E	number of erasures
M_s, M_p, M_f	number of switch, partial, full merges
F_h, F_m, F_t	number of head, middle, tail fragments
H_l	mean head length in pages
L_f	total number of fragments per log (FAST)
L_v	number of valid fragments per log (FAST)

TABLE I: List of variables

III. Analytical Results

We examine the case of a write-only workload, aligned in starting address and length to the page size. We assume writes are uniformly distributed across the logical address space, with lengths exponentially distributed with mean length w. (See Table I for a full list of variables referenced.)

We then divide each request into *head*, *middle*, and *tail* fragments as shown in Figure 1, where each fragment is the portion of the write falling within a single erase block.

More formally, each request may be represented as a sequence of consecutive pages $(p_0, p_0 + 1, \ldots, p_j, \ldots)$, and fragments are the largest sub-sequences $(p_i, \ldots, p_j)$ such that $\lfloor p_i/B \rfloor = \lfloor p_j/B \rfloor$ where B is the block size in pages. The conditions for head, middle, and tail fragments are:

- Middle: $|(p_i, \ldots, p_j)| = B$ (a full block); otherwise:
- Tail: $p_i \equiv 0 \bmod B$ (starts on block boundary); o.w.:
- Head.

First we approximate fragment distributions:

Heads: Although heads which begin on block boundaries are classified as middles or tails, this number is small for $B \gg 1$; so the number of heads is the number of sequences:

$$F_h = \frac{T}{w} \tag{1}$$

where T is total page writes and w is the mean write length.

Middles: The number of middle fragments in a single request of length N pages is $\lfloor \frac{N-B}{B} \rfloor$ for requests which do

not begin and end on block boundaries. The write length is modeled as an exponentially distributed random variable W of mean w; the total number of middles is thus

$$F_m = \frac{T}{B} \cdot Pr(W \geq B) = \frac{T}{B} \cdot (1 + \frac{1}{w})e^{-\frac{B}{w}} \quad (2)$$

Tails: Similarly, tails are given by:

$$F_t = \frac{T}{B} \cdot Pr(W < B) = \frac{T}{B} \cdot (1 - (1 + \frac{1}{w})e^{-\frac{B}{w}}) \quad (3)$$

For hybrid block-mapped FTLs, the class of a fragment is the primary determinant of its performance, which may be measured in several equivalent ways, e.g. write amplification, number of writes to flash, or number of erasures; we model erasures, but other metrics may be derived directly from these results. We present results below for the number of erasures for a given workload in both Hybrid Log Block and FAST.

A. Hybrid Log Block

As described above, with this scheme each logical block is stored as a fully ordered (i.e. logical page numbers and physical page numbers match) *data block*, and may in addition be associated with a *log block* for unordered update pages to that block. We approximate switch, partial and full merges from survival probabilities of head, middle, and tail fragments in the log list. Since the free space fraction f is comprised of all the log blocks, for $f \leq 1$ it is also the probability of uniform writes to hit a logical block that already has a log block allocated. Switch merges result from middles arriving to data blocks with no allocated log block:

$$M_s = F_m(1 - f) = \frac{T}{B} \cdot (1 + \frac{1}{w})e^{-\frac{B}{w}}(1 - f) \quad (4)$$

Partial merges result from tails arriving to data blocks with no allocated log block in the case where no additional arrivals to this block occur before it is evicted:

$$M_p = F_t(1 - f)^2 = \frac{T}{B} \cdot (1 - (1 + \frac{1}{w})e^{-\frac{B}{w}})(1 - f)^2 \quad (5)$$

Full merges result from middles and tails arriving to data blocks with allocated log blocks (causing fragmentation), and heads that do not have an allocated log block (these are counted once when the log is allocated, since several heads or tails can be mapped to a log, but lead to a single full merge).

$$M_f = (F_m + F_t)f + F_h(1 - f) = \frac{T}{B}f + \frac{T}{w}(1 - f) \quad (6)$$

The total number of erasures for the Hybrid Log scheme is:

$$E = M_s + M_p + 2M_f \quad (7)$$

B. FAST

FAST uses a dedicated log called the sequential block for writes that start on a block boundary. Each middle thus translates to a switch merge, and each tail to a partial merge:

$$M_s = F_m = \frac{T}{B} \cdot (1 + \frac{1}{w})e^{-\frac{B}{w}} \quad (8)$$

$$M_p = F_t = \frac{T}{B} \cdot (1 - (1 + \frac{1}{w})e^{-\frac{B}{w}}) \quad (9)$$

Full merges result from head fragments, and the number of erasures is determined by the number of valid fragments in a log block. Let L_f be the number of fragments per log (both valid and invalid):

$$L_f = \frac{B}{H_l} \quad (10)$$

where H_l is the expected head length. For uniform traffic, there is a $1/B$ chance of the head to start in any page of the block. Each possible head length (from 1 to $B-1$) is achieved with some probability depending on the write length and the start page. We approximate the summation with the integral:

$$H_l = \frac{1}{B}\int_{h=1}^{B-1}(hPr(W > h) + h(B - h)Pr(W = h))dh \quad (11)$$

where, as before, the random variable W models the write length. Solving equation 11 gives H_l:

$$H_l = \frac{1}{B}(e^{-\frac{1}{w}}(wB - w^2 + B - w - 1) + e^{\frac{1-B}{w}}(w^2 - w - B + 1)) \quad (12)$$

We estimate the probability of fragments to remain valid using a survival function of heads in the log list. This gives valid fragments (L_v) per log:

$$L_v = \frac{F_h}{f\,(L_f \cdot F_m + F_h) + \frac{F_h}{L_f}} \quad (13)$$

Since the valid fragments correspond to data blocks containing fragments in the current log to be erased, the number of erasures resulting from full merges is computed from the number of valid fragments per log (adding one erasure for the log block).

$$M_f = \frac{L_v + 1}{L_f} \cdot \frac{T}{w} \quad (14)$$

We substitute L_v in equation 14 to obtain full merges. The total number of erasures is thus:

$$E = M_s + M_p + M_f \quad (15)$$

Validation: We validate the analytic model against simulation results. FlashSim from Pennsylvania State University [10] was used to simulate Hybrid Log and FAST algorithms. FlashSim is integrated with DiskSim [11], and implements several FTL algorithms, including Hybrid Log and FAST schemes analyzed here (also page-mapped FTL and DFTL [12]).

Figure 2 shows the accuracy of the model for uniform writes. Tests are carried out for a volume of 10^7 2K-pages written on 1GB storage capacity with 7% free space, and block size 64. Results are shown for various mean write lengths, and are in terms of *equivalent throughput* $r = \frac{N_{min}}{N_{obs}}$, where N_{min}, N_{obs} are the minimum and observed operation counts. Here the examined operation is erasure, and $N_{min} = 156250$.

C. Prediction of Real Workloads

Real workloads are not well-behaved – they are non-stationary, and writes are not distributed uniformly across the volume but instead show spatial locality. In order to accurately

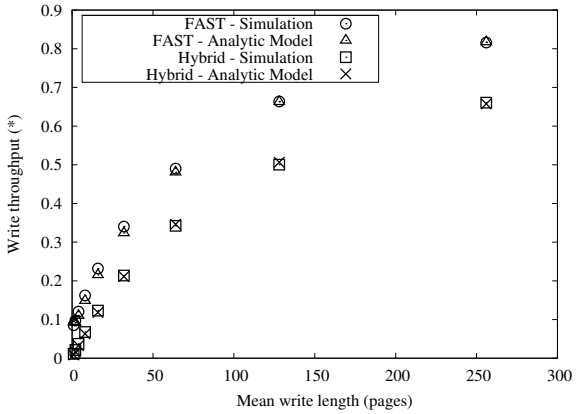


Fig. 2: Analytic model. Validation against simulation results. The 'write throughput' relates full performance to erasure cost: (total page writes / erase block size) / erasures.

predict performance we correct for the bias these factors introduce.

Non-stationarity is corrected for by dividing time into intervals during which behavior is quasi-stationary and analyzing each such interval independently. Non-uniform address distribution, however, is more complex. As seen above, performance is strongly affected by the amount of free space available; concentrating writes onto a smaller section of the volume has the effect of increasing the effective free space. (Consider a logical volume of size V_L (Table I), with k free blocks, for a free space ratio of $\frac{k}{V_L}$ – if write activity is restricted to the first half of the volume, behavior will be that of a volume with $\frac{V_L}{2}$ logical blocks, but still k free blocks, or double the free space ratio.)

To correct for this we integrate across the LBA space of the volume, assuming that each sub-range of the address space consumes a share of free space proportional to the arrival rate to that address range. In practice, we approximate the integral with a summation; thus, the total system performance P_{total} for non-uniform traffic is computed with a double summation across time and address space:

$$P_{total} = \sum_{i} \sum_{j=1}^{M} P(n(i,j), w(i,j), f(i,j)) \qquad (16)$$

M is the number of LBA space intervals, with each interval of size $\frac{V_L}{M}$ over a logical volume of size V_L.
$n(i,j)$ is the number of writes in the (i,j) interval.
$w(i,j)$ is the mean write length in the (i,j) interval.
$f(i,j)$ is the percentage of free space in the (i,j) interval: $f(i,j) = r(i,j) \cdot f \cdot M$, where
$r(i,j) = n(i,j)/n(i)$ is the access rate to the (i,j) interval, and f, as mentioned before, is the percentage of free space for the whole device.

Validation: In Figure 3 we see predictions by this method ("Prediction-Analytical") compared with simulation results for several traces used in previous studies as well [12]–[14]: four from Microsoft Research [13] (rsrch0, prxy0, src11, proj2), two database traces from the UMass Trace Repository [15] (fin1, fin2), and two from Harvard University [14] (dea2, aka deasna2, and lair62b)[2]. The traces were chosen to cover a large number of classes of workload behavior, e.g. database, network, email, research.

Although flash devices exhibit history-dependent performance (i.e. the current state of the log list depends on previous states), performance (e.g. erasures-to-writes ratio) converges in time. All reported measurements were done after convergence. Smaller traces have been processed repeatedly in the same run to ensure convergence, and only the last processing reported.

Performance is estimated using the summation in equation 16. Since finer granularity of the LBA space intervals better captures locality in real workloads, we use intervals of one logical block in size; tests showed that predictions converge to the real values when the number of LBA space intervals is increased. We measure time in units of page writes, with a time interval equal to the number of pages in the log list.

The distribution of free space per integration interval is most challenging, since its accuracy may be affected by FTL implementation details. Following the previous discussion on performance integration, next we give a more detailed description of how the free space distribution was computed for Hybrid Log scheme and FAST. Considering a (i,j) interval indexed in time and LBA space, its share of free space is:

$$free\_space(i,j) = \frac{access\_rate(i,j) \cdot (Z(i) \cdot Logs)}{LBA\_space(i,j)} \qquad (17)$$

where $Z(i)$ is a measure of free space utilization and depends on the FTL algorithm:

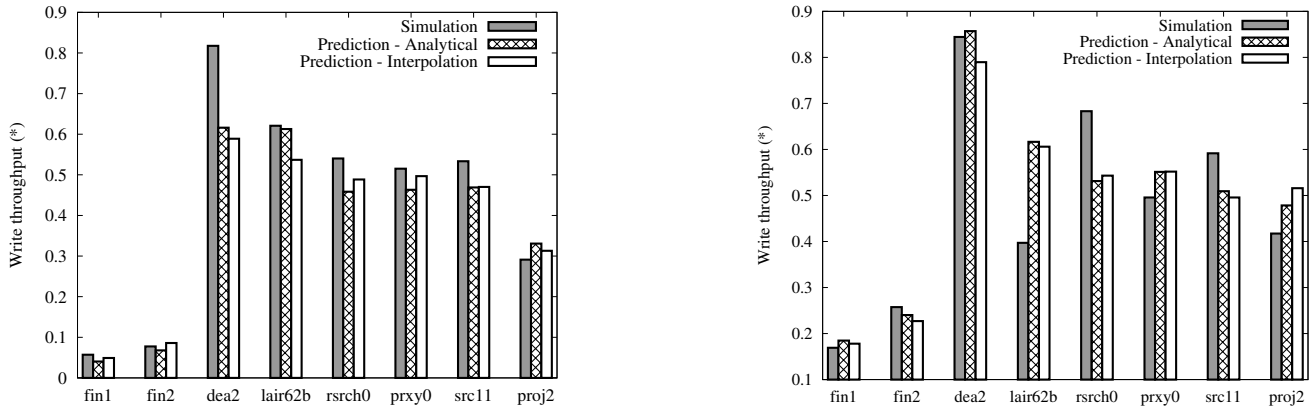
$$Z(i) = \frac{writes(i)}{logs\_used(i) \cdot log\_size} \qquad (18)$$

For FAST, where logs are completely filled sequentially, $Z(i)$ reduces to 1.

The tests reported here use the following configuration: 30GB of storage, with 7% free space, 2K pages, 64-page blocks. FlashSim was used to simulate Hybrid and FAST behavior for each trace and record the number of erasures. The simulation values are compared against the predictions.

While most of the predictions give good accuracy, we observe the following error factors. lair62b trace has a high number of block boundary accesses, translated by FAST into partial merges. FAST attempts to exploit sequentiality by using a new log block every time a boundary write occurs, even if the same data block is accessed. For some workloads, this policy leads to a high number of partial merges, which was observed in the simulation of lair62b with FAST. Unlike the other traces, dea2 has very long write lengths of 100 2K-pages on average and high variance, ranging up to 2000 2K-pages. In this case, free space distribution may be harder to compute, which is shown in Hybrid Log scheme predictions for dea2.

[2]These are file system traces; they were converted to block traces by replay and capture of block level operations.



(a) Hybrid

(b) FAST

Fig. 3: Simulation results. Analytical and interpolation-based predictions on real workloads. The 'write throughput' relates full performance to erasure cost: (total page writes / erase block size) / erasures.

IV. Interpolation from Measurements

The approach used in the previous section relies on the availability of an analytical expression for the FTL performance, but it is possible to approximate this function by measurement of performance for different values of write length and free space fraction, and interpolating from these points. In simulation, where these parameters may be easily modified, this is straightforward. We use FlashSim to generate multiple interpolation points from synthetic data with uniformly distributed start addresses, exponentially distributed write lengths. Interpolation points relate performance (erasures-to-writes ratios) to various mean write lengths and free space fractions. FAST and Hybrid Log Block were simulated with mean write lengths from 1 to 2500 pages and free space ratios from 0.001 to 1000; additional tests on the same workloads have given comparable accuracy for smaller ranges as well.

The interpolation points generated with FlashSim are used in the integration instead of the analytical formulas. We use Matlab to generate an interpolation function from multiple data points. Results are shown in Figure 3 ("Prediction-Interpolation") using the same set of traces and test configuration as above. While performance on most workloads is predicted with good accuracy, again, lair62b FAST and dea2 Hybrid Log schemes give less accurate predictions; the same workload-related aspects discussed for analytical results apply here as well and may affect estimations.

Application of this approach to real devices is similar, except that variation of the free space parameter is more difficult to predict. We do this by testing on varying fractions of the device LBA space, assuming that, as described above, the entire free space of the device will be available even when operating on only a fraction. This does not allow testing of free space ratios less than that of the device itself; however in practice this does not appear to introduce significant error. (These data points would only apply to a small fraction of arrivals, thus diminishing the effect of any errors.)

Tests were performed on two mass market SSDs (Kingston SNV425 64GB and Plextor 64GB), and are shown in Figure 4. We considered the amount of free space to be the difference from the total internal device capacity of $64 \cdot 2^{30}$ B and the size exposed to the host (approx. 7%). Write throughput was measured for uniformly distributed memoryless-length bursts, as modeled above. For Kingston 39 measurements were made with write lengths ranging from 1 to 256 pages, and free space ratios from 0.07 to 100. For Plextor 62 measurements with write lengths from 1 to 256 pages and free space ratios from 0.07 to 1000 were used. We use the FAST-like distribution of free space presented in Section III-C, as other measurements have indicated FAST-like behavior for these devices.

The SSDs' internal algorithms appear to take advantage of locality more than the standard FAST or Hybrid Log Block. We account for this by determining time intervals for each LBA range independently; however, traces with very high locality still present problems, e.g. proj2 where one fifth of the writes access only 1% of the logical address space. Using the actual LBA region size accessed in each integration interval instead of the LBA interval size/range (which differs especially for sparsely distributed writes) improves accuracy for some workloads (e.g. prxy0 on both SSDs; fin1, dea2 on Kingston).

From these results we see that a black-box prediction based on a very general model of FTL behavior results is fairly accurate for a considerable number of cases, spanning a wide range of performance. Other workloads are not as well predicted; further investigation and refinement is ongoing.

V. Prior Work

Analytic models for disk drives and arrays have been studied for many years [16]–[20]. Shiver et al. [21] develop analytic models of disks, where performance prediction for a storage device is a function of both the storage device itself and the workload. Their workload modeling, which attempts to capture temporal and spatial locality in request streams, is analogous to

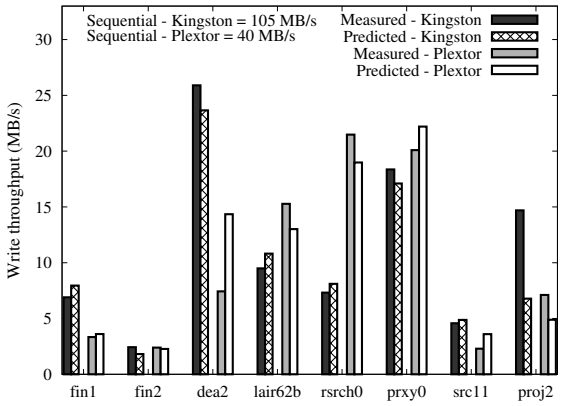


Fig. 4: SSD performance predictions on real workloads. In all but two cases (dea2 for Plextor, proj2 for Kingston) good correspondence between predicted and measured throughput was observed.

our work; however, the models are designed for mechanically-determined disk characteristics, not flash.

FTL performance models have been studied by both Toledo [2] and Hu [1], who derives an analytic model for performance of a page-mapped FTL as a function of the free space ratio, using both optimal and windowed garbage collection. We extend Hu's work by (a) deriving models for the performance of two hybrid block-mapped FTLs, with cleaning policies far different from the windowed policy examined by Hu, (b) modeling performance on larger and mixed write sizes, as is typical of real workloads, and (c) presenting a methodology for applying these models to real workloads.

The extension of this methodology to use interpolated measurement-based performance functions is related to black-box performance models, such as those proposed for disk drives using statistical machine learning techniques [22], [23]. Li and Huang [24] present a black-box performance model for SSDs, which utilizes statistical machine learning algorithms to capture correlations between workload characteristics and observed performance values. However, tests do not use real workloads but generated ones, with a predefined set of access patterns (sequential, random and stride).

A number of recent benchmarking studies have examined performance characteristics of flash devices [25]–[27]. One extensive study is uFLIP [7], a benchmark for understanding flash device performance under various IO patterns; measurements of a wide range of devices are presented.

VI. Conclusion

Although NAND flash offers the potential for far higher performance than magnetic disk, in practice improvements from all but the most expensive solid state drives have been inconsistent at best. Yet to date there has been little understanding of the behavior of flash-based storage under realistic workloads, or of how to optimize systems for its use.

We have developed an analytic model of the performance of two commonly-used flash translation layers, along with a methodology to apply this model to real-world workloads.

References

[1] X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proc. SYSTOR'09*. Haifa, Israel: ACM, 2009, pp. 1–9.

[2] A. Ben-Aroya and S. Toledo, "Competitive analysis of Flash-Memory algorithms," in *Proc. ESA'06*, 2006, pp. 100–111.

[3] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, 2002.

[4] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Emb. Comp. Sys.*, vol. 6, no. 3, 2007.

[5] J. Kang, H. Jo, J. Kim, and J. Lee., "A Superblock-based Flash Translation Layer for NAND Flash Memory," in *Proc. EMSOFT'06*, 2006, pp. 161–170.

[6] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J. Kim, "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," *ACM Trans. Emb. Comp. Sys.*, vol. 7, no. 4, 2008.

[7] L. Bouganim, B. Jonsson, and P. Bonnet, "uFLIP: understanding flash IO patterns," in *CIDR*, Asilomar, California, 2009.

[8] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proc. FAST'10*. USENIX Association, 2010.

[9] M. Sanvido, F. Chu, A. Kulkarni, and R. Selinger, "NAND flash memory and its role in storage architectures," *Proc. IEEE*, vol. 96, no. 11, pp. 1864–1874, 2008.

[10] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A Simulator for NAND Flash-based Solid-State Drives," in *Proc. SIMUL'09*, 2009.

[11] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger, "The DiskSim Simulation Environment Version 4.0 Reference Manual," 2008.

[12] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. ASPLOS'09*. ACM, 2009, pp. 229–240.

[13] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," in *Proc. FAST'08*.

[14] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive NFS Tracing of Email and Research Workloads," in *Proc. FAST'03*, 2003, pp. 203–216.

[15] "OLTP Trace from UMass Trace Repository," http://traces.cs.umass.edu/index.php/Storage/Storage, 2007.

[16] E. K. Lee and R. H. Katz, "An analytic performance model of disk arrays," in *Proc. SIGMETRICS'93*. ACM, 1993, pp. 98–109.

[17] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer*, vol. 27, no. 3, pp. 17 – 28, 1994.

[18] M. Uysal, G. Alvarez, and A. Merchant, "A Modular, Analytical Model for Modern Disk Arrays," in *Proc. MASCOTS'01*.

[19] E. Varki, A. Merchant, J. Xu, , and X. Qiu, "An Analytical Model of Disk Arrays under Synchronous I/O Workloads," Univ. of New Hampshire, Technical Report, 2003.

[20] E. Varki, A. Merchant, J. Xu, and X. Qiu, "Issues and challenges in the performance analysis of real disk arrays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, pp. 559 – 574, June 2004.

[21] E. A. M. Shriver, A. Merchant, and J. Wilkes, "An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering," in *Proc. SIGMETRICS'98*, 1998, pp. 182–191.

[22] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage Device Performance Prediction with CART Models," in *Proc. MASCOTS'04*, 2004, pp. 588–595.

[23] L. Yin, S. Uttamchandani, and R. Katz, "An empirical exploration of black-box performance models for storage systems," in *Proc. MASCOTS'06*, 2006, pp. 433–440.

[24] S. Li and H. Huang, "Black-Box Performance Modeling for Solid-State Drives," in *Proc. MASCOTS'10*, 2010, pp. 391 – 393.

[25] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo, "Characterizing the performance of flash memory storage devices and its impact on algorithm design," in *Experimental Algorithms*, 2008, pp. 208–219.

[26] P. Huang, Y. Chang, T. Kuo, J. Hsieh, and M. Lin, "The Behavior Analysis of Flash-Memory Storage Systems," in *IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008, pp. 529–534.

[27] K. O'Brien, D. C. Salyers, A. D. Striegel, and C. Poellabauer, "Power and performance characteristics of USB flash drives," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2008, pp. 1–4.