Analytic Modeling of SSD Write Performance

Peter Desnoyers

Northeastern University
Boston, Massachusetts, USA
pjd@ccs.neu.edu

Abstract

Solid state drives (SSDs) update data by writing a new copy, rather than overwriting old data, causing prior copies of the same data to be *invalidated*. These writes are performed in units of *pages*, while space is reclaimed in units of multi-page *erase blocks*, necessitating copying of any remaining valid pages in the block before reclamation. The efficiency of this cleaning process greatly affects performance under random workloads; in particular, in SSDs the write bottleneck is typically internal media throughput, and *write amplification* due to additional internal copying directly reduces application throughput.

We present the first precise closed-form solution for write amplification under greedy cleaning for uniformly distributed random traffic, and validate its accuracy via simulation. In addition we also present the first models which predict performance degradation for both LRU cleaning and greedy cleaning under simple non-uniform traffic conditions; simulation results show the first model to be exact and the second to be accurate within 2%. We extend the LRU model to arbitrary combinations of random traffic, and demonstrate its use in predicting cleaning performance for real-world workloads.

The utility of these analytic models lies in their amenability to optimization techniques not feasible in simulation. We examine the strategy of separating "hot" and "cold" data, showing that for our traffic model, such separation eliminates any loss in performance due to non-uniform traffic. We show how a system which separates hot and cold data may shift free space from cold to hot data in order to achieve improved performance, and how numeric methods may be used with our model to find the optimum operating point, which approaches a write amplification of 1.0 for increasingly skewed traffic. We examine on-line methods for achieving this optimal operating point, and show that a control strategy based on our model achieves near-optimal performance for a number of real-world block traces.

Categories and Subject Descriptors B.3.3 [*Memory Structures*]: Performance Analysis and Design Aids—formal models, simulation; D.4.2 [*Storage Management*]: Garbage collection

General Terms Design, Performance, Algorithms

Keywords Solid State Drives, Solid State Storage Systems, Write Amplification, Flash Memory

SYSTOR '12 June 4–6 2012, Haifa, Israel.

1. Introduction

Log-structured file systems (LFSs) and SSD Flash Translation Layers (FTLs) both rely on a *cleaning* mechanism to consolidate free space and make it available for subsequent modification. In each case modifications are made in fixed-sized units (flash *pages* or file system *blocks*), and these writes are made in an *out-of-place* fashion—i.e. rather than modifying existing data, the previous data is *invalidated* and a new copy written. Storage is arranged in larger fixed-size units (flash *erase blocks* or LFS *segments*) which are written sequentially, destroying (either by over-writing or prior erasure) any previous data in the block. When all blocks on the media have been filled, it is necessary to select and *clean* blocks which may be used for further writes, by copying any remaining valid data so that it is not lost.

We measure the performance of this cleaning process in terms of the Write Amplification Factor, A: the ratio of total internal writes (including copying due to cleaning) to externally-requested writes. We note that this cleaning process is similar to the well-studied problem of garbage collection in programming languages [17]; however the constraints of fixed mutation and collection units in cleaning, along with perfect knowledge of storage liveness, result in a problem with substantially different solutions and behavior.

Cleaning mechanisms have been extensively studied in the context of log-structured file systems [4, 22, 29] and flash translation layers [12, 13, 21], but to date almost all work has focused on experimental validation of new algorithms via trace-driven simulation, and few analytic results have been published. We present models with closed-form results for LRU and greedy cleaning under conditions of uniformly-distributed random arrivals. We extend this work to examine the case of traffic with temporal locality, which has not been addressed in the literature to date, and present analytic models with simple numerical solutions which accurately predict degradation of cleaning performance for both LRU and greedy cleaning in the presence of simple non-uniform traffic. This approach is then applied to more complex traffic models, and shown to predict performance of simple cleaning algorithms for real-world traces.

Finally, we use our analytic results to provide insight into the well-known technique of separating "hot" and "cold" data in systems based on block cleaning. We show that simple separation of hot and cold data is insufficient to achieve performance gains possible in the presence of locality, and demonstrate how cleaning strategies informed by our models can achieve improvements of 3x or more in the presence of locality, where naïve cleaning would suffer a degradation of 2x. We construct a cleaning strategy which is optimal for our hot/cold traffic model, and present trace-driven simulation results demonstrating its applicability to real-world workloads.

2. Problem Statement

We consider the case where $U \cdot N_p$ pages of data are stored on $T \cdot N_p$ pages of media, in blocks of N_p pages each; this would correspond, for instance, to a flash translation layer with U logical blocks and T physical blocks, with a block size of N_p pages. A key metric is the degree of *over-provisioning*, where T exceeds U, the minimum amount of media needed to store the retained data. We express this as the *spare factor* $S_f = \frac{T-U}{T}$, or equivalently as the *over-provisioning factor* $\alpha = \frac{T}{U}$, the ratio by which total storage exceeds user-visible data.

At any time one block is designated as the *write frontier*; pages are written sequentially into this block until it is full, after which it is placed in the *storage pool* and a new frontier selected from the *free list*. When the free list drops below a low watermark $w \ll T$, blocks are selected and cleaned until the free list has reached w again. This is done by (1) selecting a block to clean, (2) copying any remaining valid pages within the block (we may assume without loss of generality that they are copied to the write frontier), and then (3) adding the block to the free list. If the block selected for cleaning contains N_V valid pages and $N_I = N_p - N_V$ invalid pages, the total gain in free pages will be N_I; the process will need to be repeated until N_p invalid pages have been reclaimed, increasing the length of the free list by one block.

We note that this formulation of the problem describes what is termed a *fully page-mapped* flash translation layer, as opposed to many algorithms in use which map the majority of storage in units of blocks, typically to conserve mapping table space in RAM. Existing work has already provided models for performance of several of these *block-mapped* translation layers [5], and such models lack generality as they are specific to the class of algorithm modeled. In addition, we focus on page-mapped algorithms for their efficiency, as they have been shown experimentally [14] to hold the potential for highest performance, and when used with greedy cleaning have been proven [16] to be optimal for uniform random traffic.

In this work we are concerned with cleaning efficiency, which we express as *write amplification*, the ratio of total storage writes (including data copied during cleaning) to external data written to the system. In doing so we ignore the cost of the read operation, which is significant in a disk-based file system but less so in flash-based ones; the modification to include read overhead is straightforward. [26] In addition we ignore the cost of erasure in flash-based systems; although this is expensive, typically taking 10x as long as writing a page, in contemporary devices where N_p is 64, 128, or even 256, the amortized cost of one erasure per N_p writes is not high.

Finally, we ignore the issue of wear leveling in flash-based devices where blocks have limited write/erase limits. *Static wear-leveling* algorithms used to maximize the endurance of SSDs cause additional internal writes, thus contributing to write amplification; an example is Ban's randomized algorithm [2]. We note that Ban's algorithm, as an example, copies entire erase blocks and thus incurs no additional fragmentation; if the destination of this copy is substituted for the original block, the operation of a cleaning algorithm such as LRU or Greedy will be unaffected, and the performance impact will be limited to any copying performed by the wear leveling algorithm itself.

3. Uniform Traffic Case

We first examine the case where traffic is uniformly distributed—i.e. an individual write is equally likely to invalidate any one of the N_pU pages of valid data.

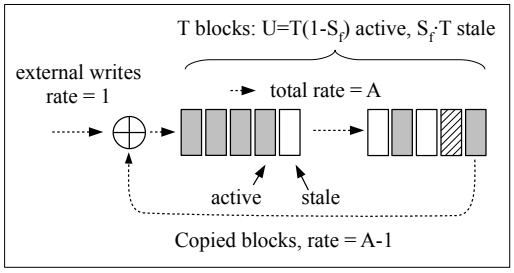

Figure 1. Model of LRU block cleaning. External writes enter at rate 1; blocks move through the queue at rate A, and are invalidated at a rate of $\frac{1}{U}$; at the end of the queue they thus remain valid with probability $\left(1 - \frac{1}{U}\right)^{T/A}$, which must also be equal to $\frac{1}{A-1}$.

3.1 LRU Cleaning

The simplest policy is LRU cleaning, where the least-recently-written block is selected for cleaning. (Since pages are only written once, this policy might equivalently be termed FIFO cleaning, as LRU and FIFO are identical in the absence of multiple accesses.) A diagram of LRU cleaning may be seen in Figure 1; after a block is fully written it enters the *LRU queue*, and upon reaching the end it is selected for cleaning, with any remaining valid pages copied back to the current write frontier.

If we assume $N_p = 1$ and that external writes arrive with a constant rate of 1 write per unit time, then blocks move from the front of the queue to the end with a velocity of A due to write amplification. While a block is on the queue the chance of it being invalidated by a particular new write is $1/U$, and since the queue is of length T, the expected number of such writes which occur during its journey is $\frac{T}{A}$; the odds of it surviving are thus $S = (1 - \frac{1}{U})^{\frac{T}{A}}$. The survival rate S then gives us the write amplification $A = \frac{1}{1-S}$, resulting in the following equation:

$$\frac{1}{1 - (1 - \frac{1}{U})^{\frac{T}{A}}} = A \tag{1}$$

If we define T in terms of the over-provisioning ratio α, $T = \alpha U$ and let $t = \frac{\alpha}{A}$, as $U \to \infty$ this converges to:

$$\frac{1}{1 - e^{-t}} = \frac{\alpha}{t} \tag{2}$$

Simulation results show that this convergence is quite rapid; for a volume of more than 1000 pages (i.e. 4 MiB with 4K pages) the effect of absolute size is negligible. Equation 2 can be solved in terms of Lambert's W function[1]:

$$t = \alpha + W(-\alpha e^{-\alpha}) \tag{3}$$

and

$$A = \frac{\alpha}{t} = \frac{\alpha}{\alpha + W(-\alpha e^{-\alpha})} \tag{4}$$

which may be seen as the top line in the graph in Figure 4.

In Table 1 simulation results for LRU cleaning, using 10^6 pages, are compared to analytic results from Equations 3 and 4; essentially exact correspondence is seen. For $T \gg 1$ the effect of block size N_p is negligible; this has been validated in simulation[2] and may be seen by constructing an equivalent system with $(T', U') = (T, U) \cdot N_p$

[1] Lambert's W function, sometimes known as ProductLog, is the inverse of xe^x; i.e. $W(y) = x | xe^x = y$

[2] Across a range of $0.04 \leq S_f \leq 0.19$, with 3200000 pages, A varied by a relative factor of 0.001 or less from $N_p = 1$ to $N_p = 256$.

S_f	Eq. 4	simulation	
0.03	16.837	16.835	±0.0036
0.07	7.318	7.317	±0.0020
0.11	4.725	4.725	±0.0013
0.17	3.129	3.129	±0.0008
0.23	2.371	2.371	±0.0008

Table 1. Comparison of LRU cleaning performance from Equation 4 with simulated results for $U = 10^6$; 95% confidence intervals are given. Correspondence is seen to be exact.

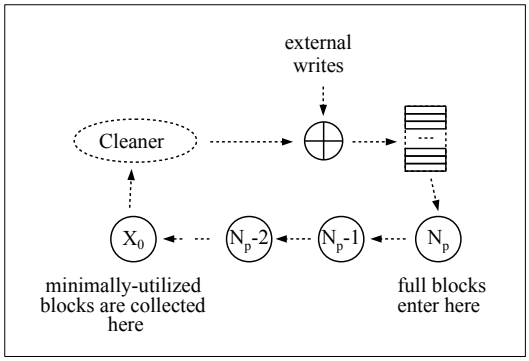

Figure 2. Markov model for a single block with greedy cleaning: block state is the number of remaining valid pages.

and $N_p' = 1$, which will function identically except in the case of writes to the last block in the original system, which occur at the negligible rate of $1/T$.

We note that Equations 3 and 4 are independent of the block size N_p. This independence has been validated in simulation, and may be shown more formally (for the case where $U, T \gg 1$) by comparing a system with block size N_p to an equivalent one with $N_p' = 1$ and $(T', U') = (T, U) \cdot N_p$. In each system the order in which pages are written and recycled is fixed; the only difference between the two is that in the first system a block of N_p pages is recycled at once, at the same point where the first of the N_p pages would be reclaimed in the $N_p' = 1$ system. In the $N_p' = 1$ system these remaining pages would be subject to invalidation for up to N_p additional steps; the rate at which such invalidations occur is no more than $1/T$, or negligible by our assumptions.

For validating this and other models in this paper we have constructed a high-speed simulator [3], consisting of approximately 1000 lines of C and Python code. This simulator ignores page contents, but is designed to be highly configurable and to enabled detailed instrumentation of behavior such as block occupancy. In the simplest case (LRU cleaning, write amplification statistics only) we are able to simulate the equivalent of a 256 GiB SSD at a rate of 3 million simulated writes per second, while for greedy cleaning a 128 GiB SSD may be simulated at 0.5 million simulated writes per second.

3.2 Greedy Cleaning

The greedy cleaning algorithm reclaims the block with the most invalid pages, thus minimizing the copying necessary; for uniform random traffic this has been shown to be optimal (see Hu [16]). In Figure 2 we see a simple Markov model for the behavior of a single block under greedy cleaning, numbering states by the number of valid pages in the block, $0 \cdots N_p$. If we again define the arrival rate as 1 page per unit time, then blocks arrive at state N_p with rate $\frac{1}{N_p}$, and as writes to a block arrive at a rate proportional to the remaining valid pages, the transition rate from state i to $i-1$ is $\frac{i}{UN_p}$.

[3] Available under an open-source license from `www.ccs.neu.edu/~pjd/ftlsim`

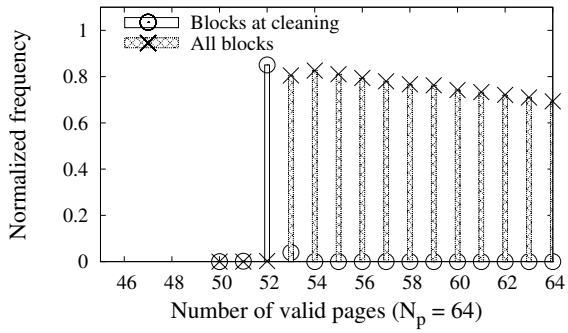

Figure 3. Greedy cleaning: distribution of block states (i.e. number of valid pages) overall and when selected for cleaning. Simulation parameters: $S_f = 0.089$, $N_p = 64$, $U = 5 \times 10^4$

The distribution of states seen during simulation, as well as that of blocks selected for cleaning, is shown in Figure 3. We note that there is a minimal state X_0 below which occupancy is negligible; once a block in state X_0 is written to, it enters a small pool of blocks eligible for greedy cleaning; the odds are small that it will receive a second write before being selected.

Letting X_0 denote the lowest-numbered state with non-negligible occupancy (53 in the figure above) we solve the balance equations as follows. First, for states $i = X_0 \cdots N_p$ we have transitions only from state i to $i-1$, with rate proportional to the number of valid pages i; letting f_i be the fraction of blocks in state i, we have

$$i \cdot f_i = k \tag{5}$$

for some constant k. If we sum $i \cdot f_i$ across all blocks we have the mean fraction of valid pages per block, S_f:

$$\sum_{i=X_0}^{N_p} i \cdot f_i = \sum_{i=X_0}^{N_p} k = (1 - S_f) \tag{6}$$

and thus

$$k = \frac{N_p(1 - S_f)}{N_p - (X_0 - 1)} \tag{7}$$

The sum of all state probabilities is 1:

$$\sum_{i=X_0}^{N_p} f_i = \sum_{i=X_0}^{N_p} \frac{k}{i} = 1 \tag{8}$$

which we approximate as an integral with continuity correction:

$$\int_{i=X_0-\frac{1}{2}}^{N_p+\frac{1}{2}} \frac{k}{i} di = 1 \tag{9}$$

and thus

$$k = \frac{1}{\log(2N_p + 1) - \log(2X_0 - 1)} \tag{10}$$

From Equations 7 and 10 we derive the equilibrium value of X_0:

S_f	$A(S_f)$	$A(simulated)$
0.03	13.624	13.631±0.002
0.05	8.872	8.870±0.001
0.07	6.623	6.625±0.001
0.11	4.430	4.432±0.001
0.17	3.002	3.002± <0.0005

Table 2. Analytic ($A(S_f)$) vs. simulated values for write amplification with greedy cleaning. $N_p = 64$, $U = 100000$, simulation results given with 95% confidence intervals. Correspondence is exact to 3 decimal places for values of S_f greater than 0.17.

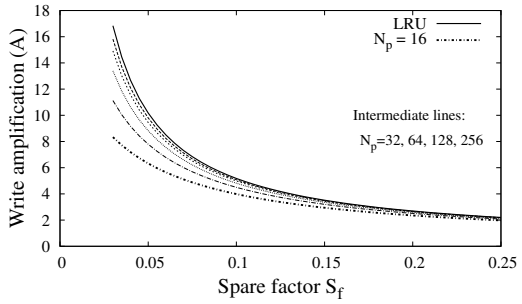

Figure 4. Greedy cleaning performance for varying block sizes, converging to the performance of LRU cleaning for large values of N_p.

$$X_0 = \frac{1}{2} - \frac{2N_p}{\alpha} W\left(-(1+\frac{1}{2N_p})\alpha e^{-(1+\frac{1}{2N_p}\alpha}\right) \quad (11)$$

and noting that reclaimed blocks have $X_0 - 1$ pages active out of N_p, we obtain the write amplification:

$$A = \frac{N_p}{N_p - (X_0 - 1)} \quad (12)$$

In Table 2 we see simulation results compared to Equations 11 and 12 for $N_p = 64$. We note that for very small values of S_f, the number of blocks reserved for the free list becomes large enough in relation to the number of spare blocks that it affects the results slightly, even for realistically-sized volumes. As an example, the simulation results in Table 2 are from a system equivalent to a 25 or 50 GiB SSD, depending on page size; without correction for free list size, the predicted write amplification for $S_f = 0.03$ drops from 13.624 to 13.393.

Unlike LRU, the efficiency of greedy cleaning is dependent on N_p, decreasing as the block size increases. In Figure 4 we see performance for values of N_p ranging from 16, the erase block size in early flash devices, to 256, the largest erase block size in production today. (typical values are 64 for SLC and 128 for MLC) As N_p increases, we see performance converge to that of LRU cleaning, with a more rapid convergence for higher values of S_f.

3.3 Windowed Greedy

Prior work by Hu [16] has examined the *windowed greedy* cleaning algorithm, a hybrid of LRU and Greedy where the search for a block with minimal valid pages is restricted to a *greedy window* of the w oldest blocks in the LRU queue, in order to reduce computational cost. We question the benefit of this algorithm, as it appears to give the write amplification of LRU cleaning with the computational complexity of Greedy cleaning.

S_f	LRU	Windowed Greedy	
0.04	12.6712	12.469	±0.0042
0.06	8.5070	8.396	±0.0027
0.08	6.4261	6.356	±0.0027
0.11	4.7254	4.682	±0.0017
0.14	3.7554	3.727	±0.0019

Table 3. LRU vs. Windowed Greedy performance for U=50000, N_p=64, w=500. With a window of 1% of the device size, we see a relative performance improvement of approximately 1%.

		Fraction of LBA space			
		mds_0	prxy_0	rsrch_0	rsrch_2
Fraction	70%	4.20%	5.60%	13.1%	59.2%
of	50%	0.70%	0.30%	1.20%	32.0%
Traffic	20%	0.00%	0.12%	0.01%	0.30%
	10%	0.00%	0.06%	0.00%	0.00%

Table 4. Locality in sample traces from Microsoft Research [23]. E.g. in the *prxy_0* trace, 50% of the traffic was written to 0.3% of the LBA space.

Using the same argument used previously to show that LRU cleaning costs are independent of N_p, we note that given equivalent systems with LRU and Windowed Greedy cleaning, the only differences in operation between the two will be when invalidations occur in any of the w blocks in the greedy window. The rate at which these invalidations occur will be negligible unless $w = O(T)$; this may be seen in Table 3, where LRU is compared in simulation to Windowed Greedy with $U = 50000$, $N_p = 64$ (the lowest value seen in current devices), and $w = 500$.

The advantage of Windowed Greedy in computational cost is marginal, as well, as may be seen in the algorithm used in our simulator. Since the number of valid pages in a block can only range over $0 \cdots N_p$, it is straightforward to keep $N_p + 1$ bins of block references; on write an $O(1)$ operation moves a reference from bin i to bin $i - 1$, while finding the minimal non-empty bin to select a block for cleaning is in the worst case $O(N_p)$, and typically $O(1)$.

4. Non-Uniform Traffic Case

Unlike the uniformly distributed workload assumed above, real storage workloads typically display significant amounts of both temporal and spatial locality. As an illustration, in Table 4 we see statistics on write operations in several of the widely used Microsoft Research storage traces [23]; in many cases 50% or more of the writes are destined to one percent or fewer of the storage. In this section we present the first analytic examination of the effect of non-uniform write distribution on cleaning performance, deriving an accurate analytic model for LRU cleaning and an approximate model for Greedy.

Temporal locality refers to the correlation between observance (or non-observance) of an address during an interval and the expected time until that address is next seen. The simplest model of temporal locality ignores any variations over time, and is thus equivalent to a non-uniform random distribution of operations over the volume address space. We examine this case, using Rosenblum's hot/cold data model [26] as shown in Figure 5. Some fraction f of the address space is "hot"; a fraction r of the overall arrival rate is distributed uniformly and randomly across these pages. Conversely, the remaining $1 - f$ of data is "cold", and fraction $1 - r$ of the incoming writes are uniformly distributed across this space.

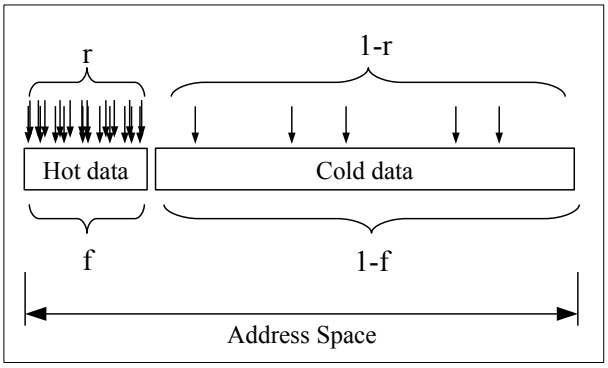

Figure 5. Hot/Cold data model used for analysis. The fraction of the total write rate destined for "hot" pages is r; these hot pages represent a fraction f of the overall address space—e.g. 80% of writes destined to 20% of the address space..

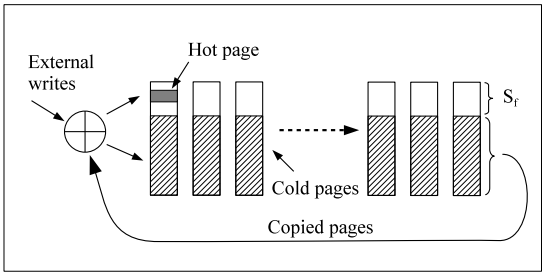

Figure 6. Limiting behavior for hot/cold traffic with LRU cleaning: $f \to 0$ (i.e. one hot page) and $r \to 1$ (no writes to cold pages).

We note that this is implicitly a "closed-world" model, comprised of UN_p data pages which are repeatedly over-written at various rates. As such this model directly applies to block device FTLs but is less applicable to log-structured file system cleaning, where data is frequently deleted rather than over-written. Extensions to this model for such transient data are under investigation.

4.1 LRU Cleaning

LRU cleaning is known to suffer degradation in performance when there is significant temporal locality [27, 30]. We can gain some insight by considering the limiting case where a single page is repeatedly re-written—i.e. $f \simeq 0$ and $r = 1$, as seen in Figure 6. In this case we see that recycled blocks contain $N_p(1 - S_f)$ cold pages; this is in fact the population average, and thus no better than if blocks had been selected randomly. Because LRU cleaning forces all blocks to be cleaned at the same age, hot data must wait too long to be cleaned (in this case it is invalidated and thus ready to reclaim almost immediately) while cold data does not wait long enough–in this limiting case it should in fact never need cleaning.

In Figure 7 we see the balance equations for LRU cleaning with hot/cold traffic depicted graphically. The total rate at which pages enter the top of the queue is A, giving:

$$A = H' + r + (1 - r) + C' \tag{13}$$

Using $\alpha = \frac{T}{U}$ as before, we have:

$$A = 1 + \frac{r}{e^{\frac{r}{f}\frac{\alpha}{A}}} + \frac{1 - r}{e^{\frac{(1-r)}{(1-f)}\frac{\alpha}{A}} - 1} \tag{14}$$

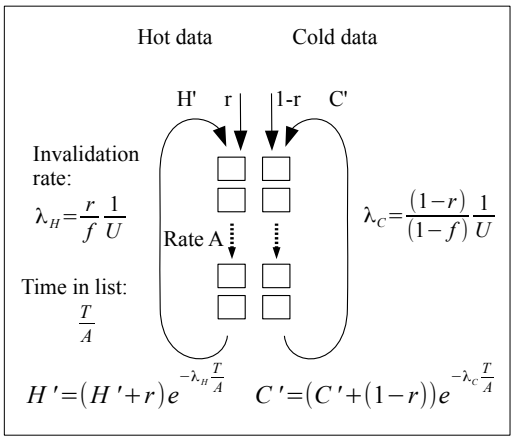

Figure 7. Balance equations for LRU cleaning with hot/cold traffic. At each step (i.e. external write) the odds of a hot page being invalidated are $\frac{r}{fU}$; during a block's passage through the list the number of external writes is $\frac{T}{A}$.

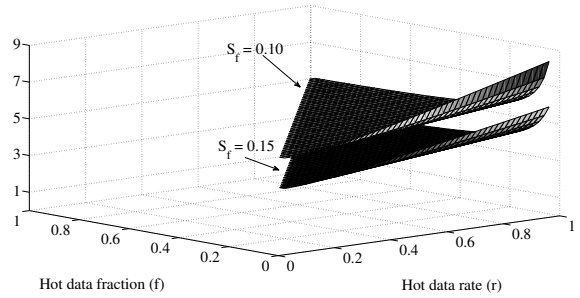

Figure 8. Performance of LRU cleaning with mixed hot/cold traffic for different values of r and f, for $S_f = 0.1$ (upper graph) and $S_f = 0.15$ (lower one). As $r \to 1$ and $f \to 0$, write amplification increases from 5.2 to 8.4 and from 3.5 to 5.9, respectively.

S_f	r	f	A (computed)	A (simulated
0.03	0.90	0.05	19.064	19.065 ± 0.002
0.07	0.8	0.2	7.682	7.681 ± 0.001
0.07	0.90	0.05	9.240	9.240 ± 0.0007
0.11	0.8	0.2	5.083	5.083 ± 0.0008
0.11	0.90	0.05	6.409	6.409 ± 0.0005
0.20	0.8	0.2	3.035	3.034 ± 0.0006
0.20	0.90	0.05	3.973	3.972 ± 0.002

Table 5. Analytic ($A(S_f)$) vs. simulated values for write amplification with LRU cleaning with 3×10^6 pages; simulation results given with 95% confidence intervals.

This equation is easily and efficiently solved numerically[4]; in Table 5 we see computed values compared to simulation results for several values of S_f, r, and f, with essentially exact correspondence. To give a better picture of cleaning behavior as locality increases, in Figure 8 we see A plotted against values of r and f for $S_f = 0.1$

[4] MATLAB code for these and other numeric computations in this paper is available under an open-source license at `www.ccs.neu.edu/~pjd/ftlsim`

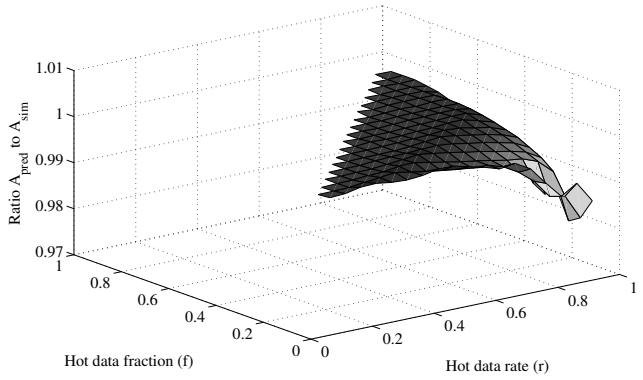

Figure 9. Ratio of predicted Greedy performance A_{pred} (Eq. 16) to simulated performance A_{sim} for $S_f = 0.10$, $N_p = 64$ across the range of values for r and f. For highly skewed data, relative error remains less than 2%.

S_f	N_p	r	f	Eq. 16	A (sim.)	rel. error
0.03	32	0.9	0.05	13.199	13.433	1.77%
0.07	64	0.9	0.05	8.461	8.608	1.74%
0.07	128	0.8	0.20	7.302	7.325	0.31%
0.11	64	0.9	0.05	6.058	6.112	0.89%
0.11	32	0.8	0.20	4.509	4.537	0.62%
0.20	64	0.9	0.05	3.845	3.826	-0.49%
0.20	128	0.8	0.20	2.984	2.992	0.27%

Table 6. Analytic ($A(S_f)$) vs. simulated values for write amplification with Greedy cleaning and 10^5 blocks, showing relative accuracy of $< 1\%$ in all except extreme cases. All simulation 95% confidence intervals are < 0.0025.

and $S_f = 0.15$ (i.e. $\alpha = 1.111$ and $\alpha = 1.177$). We only consider cases where the hot pages are in fact hot—i.e. $r \geq f$—and thus the values plotted begin at the diagonal $r = f$, or no locality, and approach the corner where $r = 1$ and $f = 0$. As we approach the corner of the graph performance degrades; the increase in write amplification is fairly linear in r, but is relatively unaffected for most values of f, rising sharply as f nears zero. Although only two values of S_f are shown here, behavior relative to the uniform case is similar for other values of S_f.

4.2 Greedy Cleaning

Performance for greedy cleaning with hot/cold traffic may be approximated very closely by noting that Equations 11 and 12 are equivalent to:

$$A = \frac{A_{LRU}(\alpha')}{1 + \frac{1}{2N_p}} \tag{15}$$

where $\alpha' = (1 + \frac{1}{2N_p})\alpha$ and $A_{LRU}(\alpha)$ is the LRU cleaning throughput for a given over-provisioning ratio from Equation 14.

Based on this result we approximate Greedy performance with hot/cold data using the same adjustment to $A_{LRU}(\alpha, r, f)$, the write amplification for LRU cleaning under the same conditions:

$$A_{Greedy}(\alpha, r, f) = \frac{A_{LRU}((1 + \frac{1}{2N_p})\alpha, r, f)}{1 + \frac{1}{2N_p}} \tag{16}$$

In Figure 9 we see Equation 16 compared with simulation results for a single configuration, with $S_f = 0.10$ and $N_p = 64$. In Table 6 we see a sampling of cases for different values of S_f and N_p; again

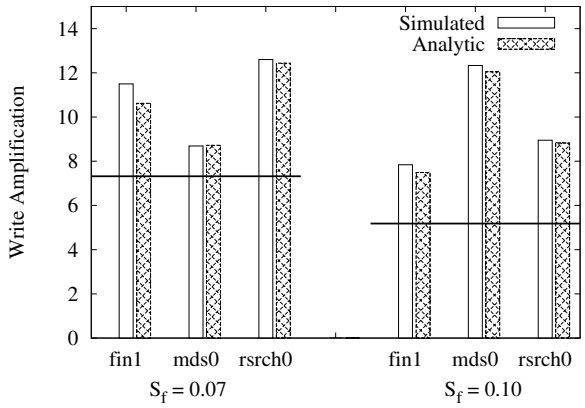

Figure 10. Predicted and simulated performance for selected traces with LRU cleaning and spare factors of 0.07 and 0.10. Horizontal lines show uniform traffic write amplification for comparison.

agreement is close but not exact, with error increasing slightly for small values of N_p and high degrees of locality.

4.3 Real-World Traffic

Equation 14 may be generalized for the case of k traffic classes, each with rate $r_i 1 \leq i \leq k$ and occupying fraction f_i of the LBA space, where $\sum r_i = \sum f_i = 1$:

$$R_i = (R_i + r_i)\, e^{-\frac{r_i}{f_i}\frac{\alpha}{A}} \tag{17}$$

$$= \frac{e^{-\frac{r_i}{f_i}\frac{\alpha}{A}}}{1 - e^{-\frac{r_i}{f_i}\frac{\alpha}{A}}} r_i \tag{18}$$

and

$$1 + \sum R_i = A \tag{19}$$

In applying this model to real-world traffic, we first note that the page-mapped LRU and greedy algorithms analyzed in this work are unaffected by spatial locality—there is no dependency in the algorithm on the actual values of the logical addresses. In addition, LRU cleaning appears to be fairly insensitive to temporal correlation: for instance, randomizing the order of writes in the traces examined causes in only small changes in LRU write amplification, indicating that results are primarily determined by the frequency of arrivals for each address. We may therefore estimate cleaning performance by measuring workload statistics and using Equations 18 and 19.

In Figure 10 we see predicted and simulated performance for LRU cleaning and three block traces: the `fin1` OLTP trace from the UMass Trace Repository [28] and the `mds0` and `rsrch0` traces from Microsoft Research [23]. In each case per-page access frequencies were grouped by quintile (i.e. those addresses responsible for the top 20% of accesses, then the next 20%, etc.) and used to construct a 5-part model for Equations 18 and 19. Prediction accuracy is seen to be quite close in almost all cases.

5. Hot/Cold Data Separation

It is well known that separating hot and cold data can increase flash translation layer performance [9, 15, 20]. To examine this, we begin by separating the problem of *identifying* hot data from *handling* it, and assume that the FTL has perfect knowledge of which data pages are hot and which are cold. Simulation results have shown that for all tested values of S_f, r, and f, with our simple hot/cold traffic model, separating hot and cold data into separate blocks and using

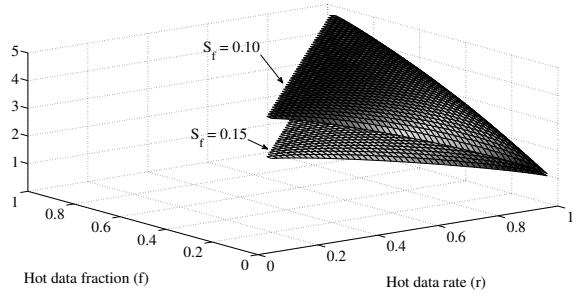

Figure 11. Write amplification with hot/cold separation and optimal division of free space, greedy garbage collection, $N_p = 64$.

S_f	N_p	r	f	A (computed)	A (simulated
0.07	64	0.90	0.05	2.325	2.335
0.07	128	0.80	0.20	4.693	4.823
0.11	32	0.80	0.20	2.919	2.991
0.11	64	0.90	0.05	1.760	1.762
0.20	64	0.90	0.05	1.311	1.312
0.20	128	0.80	0.20	1.966	2.008

Table 7. Optimal greedy cleaning for hot/cold data—computed vs. simulated performance.

global greedy cleaning results in performance that is statistically identical to that for uniformly distributed traffic. In effect, the hot and cold blocks form essentially two independent flash translation layers, as hot pages are written to blocks going into one pool, and cold pages go into a separate pool. Due to greedy cleaning, hot and cold blocks will be selected for cleaning with the same number of invalid pages, thus incurring the same write amplification. Since write amplification is a monotonic function of S_f, they will each have the same spare fraction, which must then be the same as the total spare fraction $\frac{T-U}{T}$, resulting in performance equal to that of the uniformly-distributed traffic case.

We can do better, however, by reserving a higher share of free space for the hot traffic, decreasing write amplification for hot blocks while increasing it for cold blocks. Since there are more writes to hot pages, the reduction in write amplification for these pages will more than compensate for any increase in amplification suffered by cold writes. Again the limiting case of $f \simeq 0$ and $r \simeq 1$ is illustrative; given several blocks reserved for hot traffic, the single hot page may be re-written repeatedly with no additional copying of either hot or cold data.

We can frame this as a simple optimization problem: if $A(\alpha)$ is the write amplification for uniform traffic with over-provisioning factor α, then we want to minimize:

$$A_{total} = r \cdot A(\alpha_h) + (1-r) \cdot A(\alpha_c) \tag{20}$$

where α_h and α_c are the over-provisioning factors for hot and cold data resulting from some unequal division of free space. In particular, assume that of the entire free space $T - U$ we assign some fraction p to the hot traffic, giving

$$\alpha_h = \frac{p(\alpha-1)+f}{f} \tag{21}$$

$$\alpha_c = \frac{(1-p)(\alpha-1)+(1-f)}{(1-f)} \tag{22}$$

Using the performance function for greedy cleaning from Equation 12 and optimizing numerically, we obtain results shown in

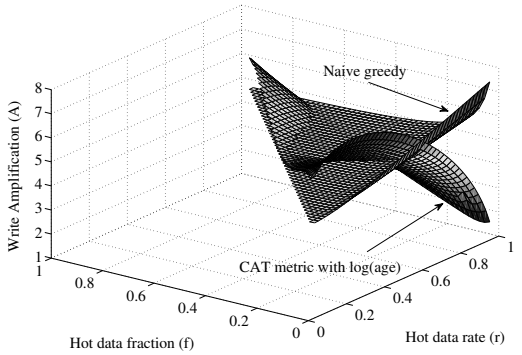

Figure 12. Write amplification with hot/cold separation and CAT metric for block selection (using $log_2(age)$) compared with naïve greedy collection, $N_p = 64, S_f = 0.1$.

Figure 11. As hot/cold disparity (i.e. spatial locality) increases, write amplification decreases towards 1. For example, for $S_f = 0.1$ and $N_p = 64$, with uniform arrivals $A = 4.82$; if 90% of the traffic is restricted to 5% of the address space, naïve greedy cleaning gives $A = 6.24$ while hot/cold separation with an optimal division of free space yields $A = 1.86$. Given a desired assignment of free space between hot and cold blocks, we may construct a cleaner which greedily chooses either a cold block or a hot block so as to maintain the specified free space division. We have simulated this, using knowledge of the synthetic workload to divide hot and cold pages perfectly, and specifying the free space assignment determined by our optimization; sample results are shown in Table 7 and shown to correspond to our analytic results.

In each case the optimal assignment results in a lower write amplification for hot data than for cold, meaning that cold blocks selected for cleaning will have few free pages than hot blocks selected. (In the case above where $S_f = 0.1$ and $N_p = 64$, hot blocks will have about 14 valid pages when cleaned, while cold blocks will be selected when 55 blocks are still valid and only 9 are free.) A number of heuristic methods have been proposed to optimally select the next block for cleaning; one of the earliest and most well-known of these is the Cost/Age/Times (CAT) metric due to Chiang [10].

CAT selects the block which minimizes the function

$$\frac{u}{1-u}\frac{1}{f(age)}t \tag{23}$$

where u is the block utilization, t is the number of times the block has been erased, and $f(age)$ is a non-decreasing function of the time since the block was written. (It was not given explicitly in the original CAT paper, but has been approximated by some as $f(age) = log_2(age)$ in more recent work. [8] Without hot/cold separation CAT is unlikely to provide significant advantage unless hot and cold arrivals are highly correlated; however we examine a modification we term CAT_{hc} where hot and cold data blocks are written to separate write frontiers, and a variant of the CAT rule is used to choose whether to select the next block for cleaning from the hot or cold pool.

In the simple hot/cold data model, if we let L be the number of blocks in the hot or cold pool:

$$L_h = Up(\alpha-1) + Uf \tag{24}$$

$$L_c = U(1-p)(\alpha-1) + U(1-f) \tag{25}$$

the approximate age at which a block is reclaimed is $\frac{L}{A}$, and since $\frac{1}{1-u} = A - 1$ and $t \propto rA_h$ (or $(1-r)A_c$), using $log_2(age)$ and

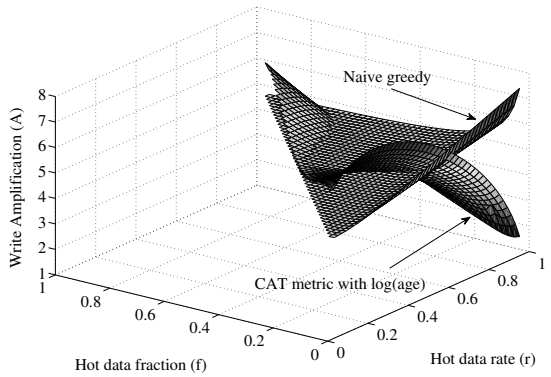

Figure 13. Calculated write amplification: CAT_{hc} cleaning with logarithmic age function, compared to naïve greedy cleaning, $N_p = 64, S_f = 0.1$.

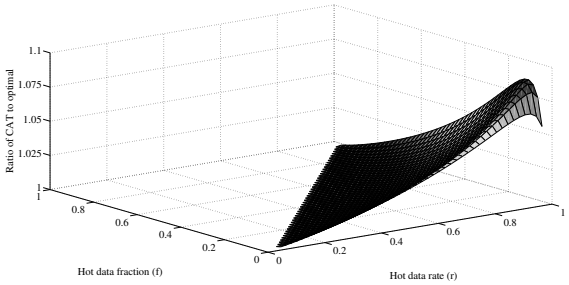

Figure 14. Calculated write amplification: CAT_{hc} cleaning with linear age function, as ratio to optimal, $N_p = 64, S_f = 0.1$.

factoring out U the decision to select a hot or cold block is based on which of the following is less:

$$\frac{(A_h - 1)A_h}{log_2(f + p(\alpha - 1)) - log_2(A_h)} \cdot r \tag{26}$$

or

$$\frac{(A_c - 1)A_c}{log_2((1 - f) + p(\alpha - 1)) - log_2(A_c)} \cdot (1 - r) \tag{27}$$

In equilibrium the two will be equal, allowing us to solve numerically for p. The result may be seen in Figure 14, compared to greedy collection without any hot/cold data separation. We note that as $f \to 0$ and $r \to 1$ it achieves significant improvements, but for certain traffic mixes it actually degrades performance.

A straightforward modification is to use age directly rather than transforming it: i.e. $f(a) = a$. This gives us the following expressions which will be equal in equilibrium:

$$\begin{aligned} &\frac{(A_h - 1)A_h}{p(\alpha - 1) + f} \cdot rA_h \\ &\frac{(A_c - 1)A_c}{p(\alpha - 1) + (1 - f)} \cdot (1 - r)A_c \end{aligned} \tag{28}$$

The result is quite close to optimal; in Figure 14 we see the ratio of write amplification for this version of CAT_{hc} to optimal for $S_f = 0.1, N_p = 64$. In the worst case linear CAT_{hc} results in about 9% higher write amplification, for a write amplification of 1.59 (vs. 1.46) for $r = 0.98, f = 0.08$.

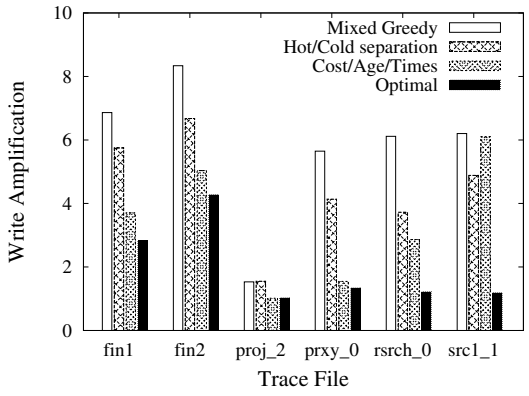

Figure 15. Write amplification for write-only storage traces using global greedy, greedy with hot/cold separation, linear CAT metric hot/cold separation, and optimal cleaning; $N_p = 64, S_f = 0.07$.

5.1 Optimal Online Hot/Cold Cleaning

We can do better, however, by examining the marginal effect on write amplification of removing a block from either the hot or cold pool. If we let $\omega = W(-\alpha e^{\alpha})$ for brevity, the derivative of the greedy performance function is:

$$dA = \frac{\alpha(\omega + \alpha)(N_p)^2}{(1 + \omega)(\alpha + \omega N_p + \alpha N_p)^2} d\alpha \tag{29}$$

If we let L be the number of blocks in a pool (hot or cold) and f be the number of free (i.e. invalid) pages in that pool, then we have:

$$\begin{aligned} \alpha &= \frac{LN_p}{LN_p - f} \\ d\alpha &= -\frac{N_p}{LN_p + f} \end{aligned} \tag{30}$$

Each time a block is to be selected for cleaning, we may then select it from the hot or cold pool according to which is less: $r\alpha_h d\alpha_h$ or $(1 - r)\alpha_c d\alpha_c$. Given perfect knowledge of hot vs. cold pages, it is clear that this will result in the optimal division of free space that minimizes Equation 20; we show below that even with probabilistic identification of access frequency it can converge in many cases to near-optimal performance.

6. Experimental Results

We have modified the simulator used in prior sections to incorporate a simple recency-based hot/cold data separation algorithm. Write sequence numbers are used rather than actual timestamps, and an exponentially weighted moving average R of recency (i.e. the number of writes since a page was last overwritten by an external write) is kept. If cold pages receive two writes in a row with inter-arrival times less than a cutoff (arbitrarily chosen as $0.7R$) they are considered hot; if a hot page is cleaned and has not been overwritten in $1.4R$ they are demoted to cold. At startup all pages are considered cold, and the first write to any page is ignored as no recency information is available. Performance of this classifier on synthetic hot/cold workloads is almost perfect; in cases where hot and cold arrival rates are sufficiently close enough to "confuse" the classifier, the degree of spatial locality is small enough that the effect on write amplification is negligible.

Traces were quantized into aligned 4K page accesses, and simulated device size was determined by the largest LBA referenced in a trace. The first 10^6 accesses from a trace were simulated to "warm up" the FTL algorithm, ensuring some degree of fragmentation;

statistics collection began after the end of the warmup period. Two OLTP traces from the UMass Trace Repository [28] were used (*fin1* and *fin2*) as well as a number of more general-purpose computing traces from Microsoft Research [23]: home (*rsrch_0*) and project (*proj_2*) directories, a source code control server (*src1_1*) and a web proxy (*prxy_0*).

In Figure 15 we see results for these traces, simulated with $S_f = 0.07$ and $N_P = 64$. The algorithms simulated are:

- Greedy: the naïve greedy cleaning algorithm, with no hot/cold separation.
- Hot/cold separation: separation of hot and cold pages into two pools, with global greedy cleaning.
- CAT_{hc}: hot/cold separation with cleaning decisions based on Equation 28.
- Optimal: hot/cold separation, cleaning based on Equation 30.

On the general-computing workloads, optimal cleaning is seen to result in a write amplification of less than 2, even with this small spare factor, with naïve greedy cleaning between $2\times$ and $4\times$ worse. The OLTP traces have significantly less spatial locality, resulting in poorer performance, although still a factor of 2 better than naïve greedy.

7. Prior Work

Several researchers, starting with Rosenblum [26], have examined the performance of block cleaning in the context of log-structured file systems. Robinson, in a 1996 paper [25], derives results equivalent to Equation 4 as an approximation to greedy cleaning performance; for typical LFS values of N_p, which may be 1000 or more, the approximation is very close. Unfortunately Robinson's paper appears to be unknown to the FTL research community, and is not cited by any of the works below, despite describing a closer approximation to greedy cleaning than all but one of them.

The earliest general performance model of FTL garbage collection known to the author is Ben-Aroya and Toledo's 2006 work, which proved hard analytic bounds for a simplified case [3]. Despite the significance of this result, however, the assumptions used in their proof prevent their model from being used to predict performance in typical configurations. In particular, they assume that free space ($T - U$ in the above analysis) is a single block, causing severe performance degradation.

Subsequent to this, Baek et al. [1] identified free space ("utilization") as a key performance parameter, providing analytic results for block cleaning performance. However, the other main parameter of their model ("uniformity") is a measure of system state, itself the result of workload and cleaning algorithm. The problem has thus been transformed from one of predicting performance of an algorithm to one of predicting the device state resulting from that algorithm, but no solution is given for the transformed problem.

Boboila and Desnoyers have published mathematic models [5] for performance of several hybrid page/block-mapped FTLs; however this work does not extend to fully page-mapped FTLs. Hu's 2009 paper [16] provides a compute-intensive procedure (typically taking more CPU time than accurate simulations) for numerically approximating the performance of the Greedy algorithm, with fairly good correspondence between predicted and simulated results for $S_f > 0.2$. Hu begins by using Windowed Greedy as an approximation of Greedy; we argue in Section 3.3 above that this approximation is unfounded, as for uniform traffic Windowed Greedy in fact performs identically to LRU.

Hu's approximation has large errors for values of S_f below 0.2, however; this range includes virtually all consumer SSDs,

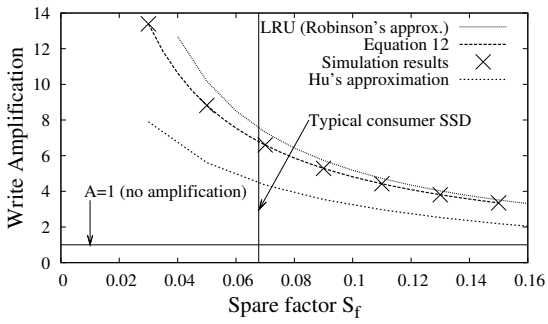

Figure 16. Hu's approximation and Robinson's LRU model compared to Greedy cleaning simulation and analytic results (Equation 12), $N_p = 64$.

which tend to have free space ratios from under 7% (e.g. the Intel device examined above, with $S_f = 0.069$) to about 13% (e.g. $U = 60 \times 10^9$ for a drive with 64 GiB of raw capacity gives $S_f = 12.7\%$). In Figure 16 we see Hu's approximation evaluated for lower S_f values and $N_p = 64$, compared with Equation 12, results from simulation, as well as LRU cleaning performance from Robinson's model. We see that Hu's approximation is not only less precise than our greedy cleaning model, but is in fact significantly less accurate than Robinson's LRU cleaning results; however we have not determined which step in Hu's derivation introduces this inaccuracy.

Bux has described a Markov model of greedy cleaning performance [6] for very small devices; the complexity of this model makes it unfeasible for realistic systems. Most recently, Bux and Iliadis [7] describe results which appear similar to our model for Greedy cleaning, with very similar results; however no closed-form solution is described and it is unclear if their approach may be extended to non-uniform and realistic workloads. In particular, we note that our optimal controller relies on the differentiability of the close-form performance function, an extension which is not possible with the Bux and Iliadis solution.

Efforts to identify and exploit spatial locality in cleaning again begin with log-structured file systems, including work by Rosenblum [27] using the simple two-parameter hot/cold traffic model used in this work, and proposing separation of hot and cold data at cleaning time and a *cost/benefit* calculation for cleaning decisions as an improvement over greedy cleaning. Chiang in turn proposed the Cost/Age/Times metric [11] as well as a multiple-pool method for separating hot and cold data [10]. We note that the cost/age/times metric examined in this work differs from Chiang's in two ways: first, in the use of a linear time function, and secondly by using greedy selection within the hot and cold pools, so that CAT is only used to compare one hot and one cold block. Chang proposed a *dual-pool* algorithm [9] for separating hot and cold data, similar to Chiang's multiple pools; however later versions of the dual-pool algorithm [8] emphasize wear leveling rather than FTL performance, and the pools are used group physical blocks (rather than logical addresses) according to their write/erase cycle counts.

These methods have all used cleaning-time decisions between one of two write frontiers to implicitly group hot and cold data. Kim and Lee [19] assume that the identity of hot and cold data is known exactly by the file system, while Jung [18] uses OS process information to inform hot/cold data separation. Hsieh [15] uses counting Bloom filters to track access frequency of data pages, while Park and Du [24] use rotating Bloom filters to track recency instead of frequency.

We believe that our optimal cleaning strategy is relatively independent of the mechanism used to track hot and cold pages; Park and Du's Bloom filter-based recency estimator is likely to be a good substitute for our resource-intensive algorithm, and other tracking mechanisms may work as well. Finally, and most significantly, unlike prior work, our work provides models for predicting the effect of spatial locality on performance, and for parameterizing an FTL to achieve optimal performance in the presence of locality.

8. Conclusions

In this work we have presented an exact, close-form model for block cleaning performance of the greedy collection algorithm under uniformly distributed traffic, and of LRU cleaning with a simple hot/cold data mix as well as more complex workloads; in addition we provide a highly accurate approximation for greedy cleaning with hot and cold data. We validate the prediations of these models in simulation, on both synthetic data and (in the case of LRU cleaning) real workloads. In addition we use these models to derive new insight into the behavior of cleaning algorithms when data is segregated by update frequency ("hot/cold data separation"), providing a control algorithm which is optimal for simple traffic mixes and demonstrates high performance for real-world workloads. We believe this work represents a new approach to the issue of Flash Translation Layer performance, based on a fundamental understanding of how actual and achievable performance is determined by workload statistics.

9. Acknowledgements

The author would like to thank the anonymous reviewers for their feedback, and Ravi Sundaram for helping start me on this line of inquiry. This work was supported in part by a Faculty Award from the IBM Corporation.

References

[1] S. Baek, J. Choi, D. Lee, and S. H. Noh. Model and validation of block cleaning cost for flash memory. In *Int'l Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 46–54. Springer-Verlag, 2007.

[2] A. Ban. Wear leveling of static areas in flash memory. United States Patent 6,732,221, 2004.

[3] A. Ben-Aroya and S. Toledo. Competitive analysis of Flash-Memory algorithms. In *Algorithms - ESA 2006*, pages 100–111. 2006.

[4] T. Blackwell, J. Harris, and M. Seltzer. Heuristic cleaning algorithms in log-structured file systems. *Proceedings of the 1995 USENIX Technical Conference*, pages 277—288, 1995.

[5] S. Boboila and P. Desnoyers. Performance models of flash-based Solid-State drives for real workloads. In *IEEE Symposium on Massive Storage Systems and Technologies*, June 2011.

[6] W. Bux. Performance evaluation of the write operation in flash-based solid-state drives. IBM Research Report RZ 3757, IBM, Nov. 2009.

[7] W. Bux and I. Iliadis. Performance of greedy garbage collection in flash-based solid-state drives. *Performance Evaluation*, 67(11):1172–1186, Nov. 2010. ISSN 0166-5316.

[8] L. Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *ACM Symposium on Applied Computing*, pages 1126–1130, Seoul, Korea, 2007. ACM.

[9] L. Chang and T. Kuo. An adaptive striping architecture for flash memory storage systems of embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, page 187. IEEE Computer Society, 2002.

[10] M. Chiang and R. Chang. Cleaning policies in mobile computers using flash memory. *Journal of Systems Software*, 48(3):213–231, 1999.

[11] M.-L. Chiang, P. C. H. Lee, and R. chuan Chang. Managing flash memory in personal communication devices. In *Int'l Symposium on Consumer Electronics (ISCE'97*, pages 177–182, 1997.

[12] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. A survey of flash translation layers. *Journal of Systems Architecture*, 55(5-6):332–343, 2009.

[13] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2):138–163, 2005.

[14] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 229–240, Washington, DC, 2009. ACM.

[15] J. Hsieh, T. Kuo, and L. Chang. Efficient identification of hot data for flash memory storage systems. *ACM Trans. on Storage*, 2(1):22–40, 2006.

[16] X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *SYSTOR 2009: The Israeli Experimental Systems Conference*, pages 1–9, 2009.

[17] R. Jones, A. Hosking, and E. Moss. *The Garbage Collection Handbook: The Art of Automatic Memory Management*. Chapman and Hall/CRC, 1 edition, Aug. 2011.

[18] S. Jung, Y. Lee, and Y. Song. A process-aware hot/cold identification scheme for flash memory storage systems. *IEEE Transactions on Consumer Electronics*, 56(2):339 –347, May 2010.

[19] H. J. Kim and S. G. Lee. An effective flash memory manager for reliable flash memory space management. *IEICE Transactions on Information and Systems*, 85(6):950–964, 2002.

[20] H. Lee, H. Yun, and D. Lee. HFTL: hybrid flash translation layer based on hot data identification for flash memory. *IEEE Transactions on Consumer Electronics*, 55(4):2005–2011, 2009.

[21] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6(3):18, 2007.

[22] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 238–251, Saint Malo, France, 1997. ACM.

[23] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: practical power management for enterprise storage. In *USENIX Conference on File and Storage Technologies*, pages 1–15. USENIX Association, 2008.

[24] D. Park and D. Du. Poster: Hot data identification for flash memory using multiple bloom filters. In *USENIX Conference on File and Storage Technologies*, 2011.

[25] J. T. Robinson. Analysis of steady-state segment storage utilizations in a log-structured file system with least-utilized segment cleaning. *SIGOPS Operating Systems Review*, 30(4):29–32, Oct. 1996.

[26] M. Rosenblum. *The Design and Implementation of a Log-structured File System*. PhD thesis, University of California at Berkeley, 1992.

[27] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *ACM Symposium on Operating Systems Principles*, pages 1–15, 1991.

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

[29] J. Wang and Y. Hu. WOLF - a novel reordering write buffer to boost the performance of Log-Structured file systems. In *Proceedings of the Conference on File and Storage Technologies*, pages 47–60. USENIX Association, 2002.

[30] M. Wu and W. Zwaenepoel. eNVy: a non-volatile, main memory storage system. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 86–97, 1994.