Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

SSD Stress Relief

Semester Thesis

Maxim Mattheeuws

mmaxim@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georg Bachmeier
Prof. Dr. Roger Wattenhofer

January 2, 2017

Acknowledgements

All my heartful thanks to Prof. Dr. Roger Wattenhofer, head professor of the Distributed Computing Group at ETH Zürich and to Georg Bachmeier, my supervisor, for their invaluable support; to Alan Kohler for his good vibes and connections; to Carlo Del Don from the department of Computational Science and Engineering for helping me to exorcise the occult inner workings of Matlab.

Abstract

This paper investigates important issues specific to the usage of Solid-State Drives (SSDs). It focuses on how to simultaneously optimize wear-leveling and write amplification while considering the over-provisioning factors. Subsequently, we propose a new approach for the computation of a score necessary for block reclaims. Furthermore, we compare the results of Desnoyers (2012) [1] to real-life data sets. This leads to our proposal for a new model of write traffic.

Contents

Chapter 1

Problem Statement

1.1 Description

SSDs are becoming the go-to hardware for storage. Unlike Hard Disk Drives, they raise particular issues due to the combination of their NAND flash architecture, where writing is page-wise but deleting is block-wise, and the fact that writing to SSD cells causes deterioration.

More specifically, an SSD is composed of m blocks, each one of them consisting of n pages. The pages are the smallest unit of datum. A block usually contains a mix of valid pages (relevant data) and invalid pages (redundant data).

When the SSD is full, meaning that no blocks have free pages available, it is impossible to delete only the subset of invalid pages on a block. The block needs to be deleted in its entirety, due to the architectural constraints of NAND flash memory. The valid pages are copied to a buffer and then rewritten on the SSD after the block has been wiped clean. The rewriting of valid pages increases write amplification, which is the ratio of pages written on the SSD to the amount of page writes issued by the operating system. Thus the total amount of writes increases.

As writing on the SSD causes deterioration, the process of selecting a block for reclaim is not a simple one. It is important not only to minimize the write amplification, which means selecting the block with the smallest amount of valid pages that will need to be rewritten, but also to optimize the wear-leveling, which means using all m blocks as evenly as possible. For example, if block $m1$ is systematically the block containing the smallest number of valid pages compared to all the other blocks, for wear-leveling reasons, it still should not necessarily be chosen systematically for reclaim.

Also, the over-provisioning factor from 7 to 10% incorporated by manufactures leaves room to carry out diverse subdivision strategies. By over-provisioning, the operating system is shown less space available than there physically is on the SSD. The aim is to give more room to the SSD to manage the blocks in order to obtain blocks as homogeneous as possible in terms of data temperature, with the effect of minimizing the write amplification and optimizing wear-leveling when reclaiming blocks.

Thus, the choice of the block to be reclaimed is crucial and difficult as it has to optimize both wear-leveling and write amplification. A widespread practice consists in assigning scores to the blocks, based on a variety of parameters.

In this paper, we will review various practices and existing scores for different traces and how to improve the utilization of the over-provisioning factor. We will also propose a new score and a new type of write traffic model.

1.2 Technical Terms

In order to avoid confusion and in effort to provide a consistent notation, the most important terminology is listed below.

Page: A page is the smallest unit of a datum. In this work, it consists of 512 bytes.

Block: In an SSD, pages are written on bigger units, called blocks. The amount of blocks in the SSD is denoted as m. The assumption is made that a block consists of 512 pages [2], thus setting $n = 512$. Desnoyers uses the notation N_p for the amount of pages per block.

Reclaim: When the SSD is full, a block must be chosen to be reused. This process is called a block reclaim and exists due to the fact that only block-wise deletion is possible in NAND flash devices.

Write Amplification (denoted as A) happens when a block is reclaimed and still has valid pages on it. In an SSD, these pages are copied to a buffer before being rewritten on the SSD after the block has been wiped clean. Thus, a datum which the Operating System required to save once can de facto be written several times on the drive. The perfect algorithm would keep the write amplification as close to 1 as possible, which is the main goal alongside wear-leveling.

Wear-Leveling is the other goal the perfect algorithm should achieve on an SSD. It consists of using the drive evenly.

Over-Provisioning Factor is the ratio between how many pages fit on the SSD and the biggest address the Operating System is allowed to write to. It is denoted as $\alpha = \frac{m \cdot n}{q}$.

Chapter 2

Current Practice

Essentially, we can model an SSD as a non-linear function with the input as a vector containing the write addresses and output being wear-leveling, inferred from the variance of the number of reclaims of each block, and the write amplification generated, inferred from the total amount of reclaims. We want to minimize both outputs and thus alter a process in the SSD: the score computation. Depending on the input vector, the manner of computing a certain score can lead to the goal of minimizing wear-leveling and/or write amplification or do the opposite; in heuristics, there is always a way to make a score perform poorly or excellently by changing the input vector. Below follows a review of the existing scores.

2.1 Scores

Once the SSD has been filled up, a method needs to be determined to choose which block to reclaim. Apart from LRU, the common approach is to choose the block with the *highest* score, delete all its content while copying the valid pages to a buffer and then writing them back to the same block. Thus, new pages can be written on the SSD until full and the process reiterates. This section discusses several possible computations of scores.

2.1.1 LRU

One intuitive way to choose a block is to do it Round-Robin-style, meaning that for the l-th reclaim, block $l \mod m$ is chosen (m is the amount of blocks on the SSD). LRU, which stands for Least Recently Used, does exactly that. It is easy to implement and good for wear-leveling, but can perform poorly for write amplification.

2.1.2 GreedyVariance

Similar to LRU, this score minimizes the wear-leveling without taking write amplification into account. One main difference is its flexibility as there is no strict Round Robin:

$$\text{score} = \frac{1}{\textit{lifecycles}} \tag{2.1}$$

lifecycles stands for how many times a block has been reclaimed.

2.1.3 GreedyReclaim

In order to minimize write amplification, the intuitive answer is to choose the block with as many invalid pages on it as possible. The score may perform badly for wear-leveling.

$$\text{score} = \textit{invalid} \tag{2.2}$$

invalid denotes the amount of invalid pages on a block; *valid* is the amount of valid pages on a block.

2.1.4 CAT

A famous score that minimizes both wear-leveling and write amplification is the Cost-Age-Times score computation proposed by Chiang [3]. Viewed as the state-of-the-art score, it is computed in the following manner:

$$\text{score} = \frac{\textit{invalid} \cdot \log_2(\textit{age})}{\textit{valid} \cdot \textit{lifecycles}} \tag{2.3}$$

Every time a block is reclaimed, a global reclaim counter is incremented and its value is marked on the reclaimed block. This means essentially that every block has a time stamp. The difference between this value and the current reclaim counter is the *age* of the block.

2.1.5 CICL

The idea in CICL [4] is to have a function determining if the policy should focus more on wear-leveling or write amplification depending on the current state of the flash device. The function is computed as follows

$$\lambda = \frac{\textit{maxerasures} - \textit{minerasures}}{\textit{maxerasures}} \tag{2.4}$$

Therefore, $0 \leq \lambda \leq 1$. It is important to remember that λ is a function of the whole SSD, opposed to the score which is assigned to individual blocks.

maxerasures is the highest *lifecycles* value found on the entire SSD, *minerasures* the lowest. Then the score is computed by setting

$$\text{score} = (1 - \lambda) \cdot \frac{\mathit{valid}}{\mathit{valid} + \mathit{invalid}} + \lambda \cdot \frac{\mathit{lifecycles}}{1 + \mathit{maxerasures}} \tag{2.5}$$

Caveat: Computed this way, the block with the *lowest* score is eligible for reclaim, as opposed to the other scores.

2.2 Over-Provisioning Factors

Apart from the scores, it is crucial to consider the traffic of address writes on an SSD. In real-life data sets, addresses written in the SSD have different probabilities and distributions. One common approach to categorize these writes is to subdivide them by their overall occurrences. If a page is written only once on the SSD, it is said to be *static*.

Then there is *cold* and *hot* data, which are subdivided arbitrarily. For this work, the assumption is made that there is a threshold t for every data set. If the address occurs less than t and more than once in the set, it is cold; else it is hot. t is computed by taking the square root of the arithmetic mean of the top 1% highest occurring address counts.

A sensible idea is then to subdivide the SSD as well and adapt the relative over-provisioning factors of the static, cold and hot pool denoted as α_s, α_c and α_h respectively. α_s should be set to 1 as static data can be stored on minimal space. Dealing with the other two factors, however, is not trivial. One possible way is to assume uniform distribution in the respective hot and cold pools, like Desnoyers did.

2.2.1 Desnoyers (2012)

In his paper [1], Desnoyers derives the mathematical tool based on simplifications (uniform write distribution using GreedyReclaim) to find the ideal over-provisioning factors for hot and cold data pools. Defining X_0 as the lowest amount of valid pages on blocks eligible for reclaim and N_p the amount of pages on a block, write amplification should be

$$A = \frac{N_p}{N_p - (X_0 - 1)} \tag{2.6}$$

with

$$X_0 = \frac{1}{2} - \frac{2N_p}{\alpha} W\left(-1(1 + \frac{1}{2N_p})\alpha e^{-\left(1+\frac{1}{2N_p}\right)\alpha}\right) \tag{2.7}$$

where $W(x)$ is the Lambert function. Now setting f as the fraction of the address space occupied by hot pages and r as the fraction of the rate of hot pages occurring, a fraction p $(0 \leq p \leq 1)$ can be found numerically by minimizing

$$A_{\text{total}} = r \cdot A(\alpha_h) + (1-r) \cdot A(\alpha_c) \tag{2.8}$$

with

$$\alpha_h = \frac{p(\alpha - 1) + f}{f} \tag{2.9}$$

$$\alpha_c = \frac{(1-p)(\alpha - 1) + (1-f)}{(1-f)} \tag{2.10}$$

Chapter 3

Implementation

3.1 SSD

In order to visualize and analyze the specificity of an SSD, a Matlab model has been developed and implemented. As standards in the industry are kept secret, it is difficult to evaluate the accuracy of the model in comparison with cutting edge technology, but fundamentally the model simulates a fully functioning NAND flash device where a state expressing the validity of the hypothetical datum is kept instead of the datum itself.

The actual SSD is a matrix of m rows and n columns, where m equals the amount of blocks and n the amount of pages per block. On initialization, all entries are set to 0, standing for *free*. The SSD is filled block-wise with incoming pages. To keep track of the location of these pages, a mapping matrix of q rows and 2 columns is initialized, where q is the highest address made available to the Operating System. Setting the address as the index, the first column denotes the block and the second column the page on which the data of the address has been written. At every incoming write a 1 standing for *valid* is set on a formerly free entry of the SSD matrix. It is checked through the mapping matrix if the address has already been written on the SSD. If yes, the relevant entry on the SSD is invalidated by setting it to -1 before updating the address in the mapping matrix.

Once the SSD is full (no entries are 0), a score vector of the size m is generated in order to assign a score to each block. On the next incoming address, the model chooses the block with the highest score *which is not full with valid pages only.* It proceeds to set all *invalid* entries in the SSD matrix to *free.* It then increments the respective entry in a life cycle vector, which also has the size m as only block-wise deleting is possible. It updates the block's score and fills it up with new incoming pages. This procedure allows not having to update the mapping matrix. The program then reiterates until all the write instructions have been executed. Introducing metric vectors of the size m like the life cycle vector is necessary to compute the scores.

3.2 Traces

In this work, two type of traces are examined.

On the one hand, traces from a Microsoft data center [5] are treated to fit the format read by the SSD model. The raw data has the form

```
[Time Stamp, Host Name, Disk Number, Type, Offset, Size, Response Time]
```

and is saved in a Comma Separated Value file. Every entry is filtered according to its respective disk; stripped off its time stamp, host name and response time; offset and size which were given in bytes are converted to pages consisting of 512 bytes. Then the individual page write is generated by adding as many pages to the offset as there are available in size. Eventually, a mapping takes place in order to minimize the address span, due to the fact that it is not contiguous.

On the other hand, for research purposes, we also generated synthetic traces. Two particular examples are uniform writes, used in literature, and linearly slanted writes, subsequently discussed.

Chapter 4

Experimental Results

4.1 SSD Subdivision

The over-provisioning factors computed for the `CAMRESHMSA01-lvm1` (short: 01_1) Microsoft Data set using Desnoyers' findings and running the SSD model leads to a very good result. Not only are the over-provisioning factors close to the actual optimum for GreedyReclaim, they also coincide for all other scores. In order to find the best subdivision, a brute force algorithm has simulated all possible subdivisions. Its result can be seen in figure 4.1. Table 4.1 also shows that, in case of GreedyReclaim, the Desnoyers subdivision generates only $2864 + 1302 = 4166$ total reclaims against $2584 + 5206 = 7790$ for standard subdivision and 5680 for no subdivision. Noteworthy: In this case, no subdivision is better than a standard subdivision.

However, for the Data set `CAMRESHMSA03-lvm1.csv` (short: 03_1) the results are less than optimal. Interestingly, the graph generated by the algorithm is quite different. For this data set, in the case of $\alpha = 1.07$, Desnoyers tells us to choose 354 hot blocks. Also, no subdivision leads to 14232 reclaims in total, compared to Desnoyers' $29581 + 99 = 29680$ as seen in Table 4.1 and the optimal 15683 found by brute force in figure 4.2, meaning that no subdivision is the best option to minimize reclaims.

This counter-intuitive result can be interpreted as a poor choice for our assumption of hot and cold subdivision regarding the second data set. The point is that the hot and cold dichotomy is arbitrary. Depending on the data set, it will be necessary to create several temperature pools to approximate uniformity in the respective traffic pool.

4.1.1 Linslant: A new approach

In uniform traffic, it is assumed that each of i pages is written j times. In order to approximate real-life data, the common approach is to do a superposition of uniform traffic, usually with just two pools (hot and cold).

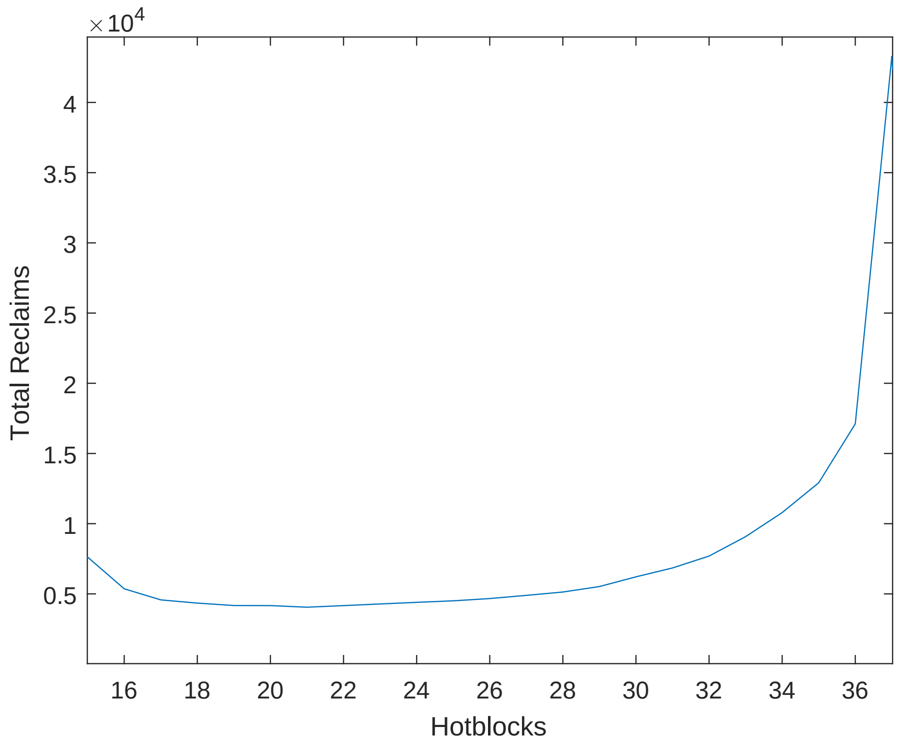

Figure 4.1: Subdivisions with GreedyReclaim for `CAMRESHMSA01-lvm1`. On the x-axis, the number of blocks conceded to the hot pool is displayed; on the y-axis, the amount of total reclaims is shown. Desnoyers' subdivision works well as he tells to assign 20 blocks to the hot pool, thus minimizing the total amount of reclaims.

A quite intuitive opposition to uniform traffic can be the following: assume traffic, where address a occurs j_a times, j_a being unique to the address. This means that the uniform model would need as many temperature pools as occurring addresses. To generalize, think of equally big pools of addresses occurring j_p times, meaning we have uniform traffic in each pool. Now let the the amount of pools go to infinity. This model has been engineered during this work and called *linearly slanted* or short: *Linslant* traffic.

In order to visualize the nature of the data sets, a Matlab script called SpectralAnalysis has been developed and implemented. It generates a plot where the x axis represents the number of times an address has been found in a data set, and the y axis denotes the amount of addresses written the same number of times. In this representation, uniform traffic is a spike, and Linslant traffic a constant.

A possible interpretation is to say that perfectly uniform traffic is a scaled Gaussian distribution with mean j and variance 0. But when the variance in

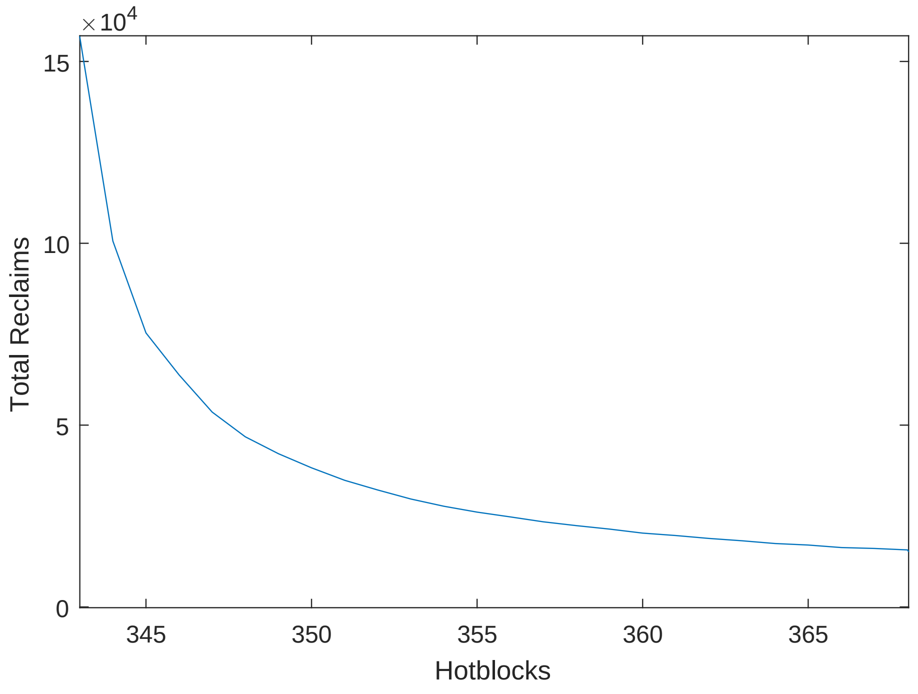

Figure 4.2: Subdivisions with GreedyReclaim for `CAMRESHMSA03-lvm1`. The x and y-axis are the same as in figure 4.1. However, Desnoyers tells us to assign 354 blocks to the hot pool, which is not the best subdivision, as deductible from this figure. More importantly: No subdivision yields a smaller amount of total reclaims.

real-life data sets exceeds a certain value, it may be a better idea to approximate it with a scaled Gaussian distribution with infinite variance, which would be Linslant.

4.2 Score Evaluation

4.2.1 Dynamically Operating Greedy

A new score has been engineered in the scope of this work. The insight relies on the fact that initially, wear-leveling should not matter as the flash device is in the early stage of its life and thus should not fail. However, as blocks approach the end of their lifetime due to reclaims, wear-leveling becomes more and more important. First a coefficient is computed in order to determine the relative age of the block

$$\delta = \frac{\mathit{lifecycles}}{\mathit{lifeexp}} \tag{4.1}$$

And then the score is computed as

$$\text{score} = \frac{(1-\delta)\cdot \mathit{invalid}}{\delta \cdot \mathit{lifecycles}} \tag{4.2}$$

which is the combination of GreedyVariance and Greedyreclaim weighted with δ. Hence the name, Dynamically Operating Greedy, or short: DOG.

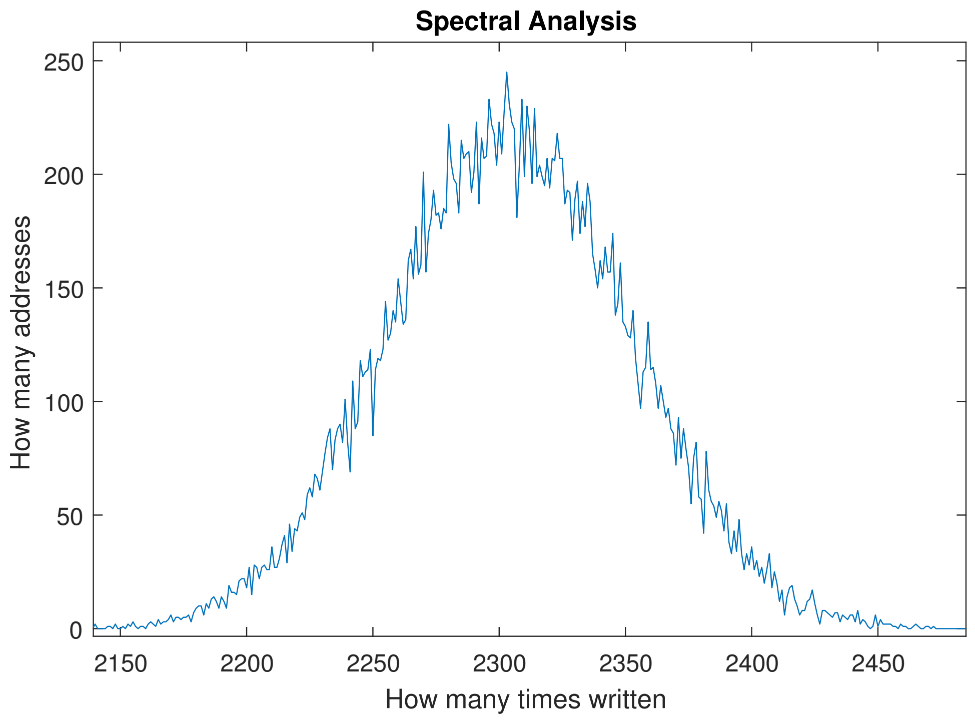

Figure 4.3: The SpectralAnalysis plot for synthetically generated uniform traffic. It indicates how many addresses (y-axis) have been written how many times (x-axis). In the Gaussian interpretation: the smaller the variance, the more ideal the uniformity.

4.3 Overview of simulation runs

Table 4.1 contains the findings for all the simulations. For each score it shows the results under various conditions.

The new DOG score and the state-of-the art CAT score perform equally

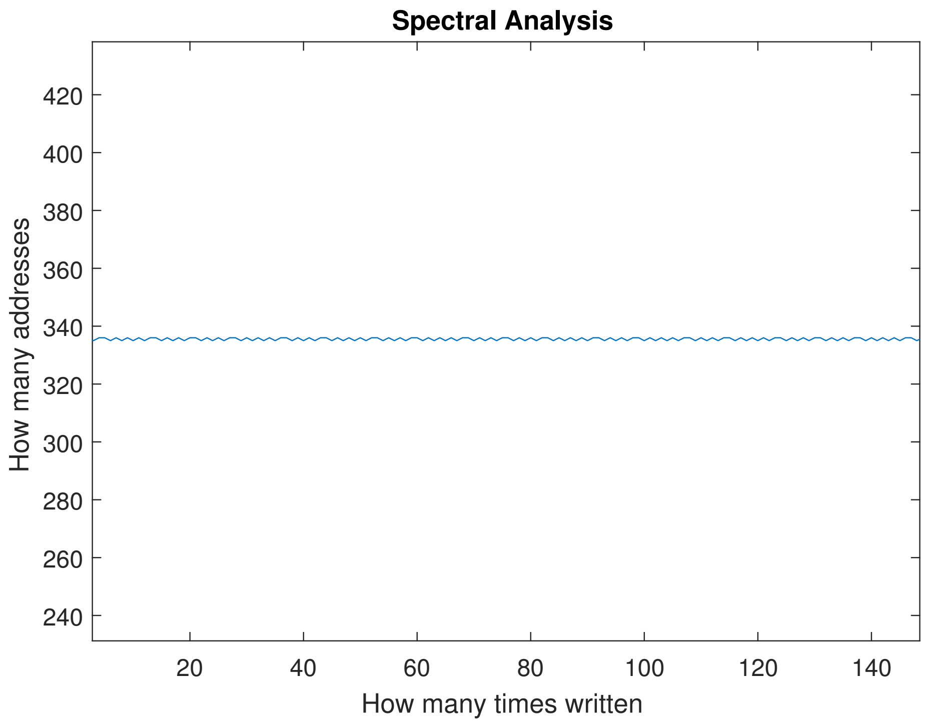

Figure 4.4: The SpectralAnalysis plot for synthetically generated Linslant traffic. Thes axis denote the same as in figure 4.3. In the Gaussian interpretation: the bigger the variance, the more ideal the Linslant.

well for uniform traffic. For Linslant, however, DOG dominates both in terms of variance and reclaims. Considering the Microsoft data sets, the weakness of CAT shows through: it performs very poorly under mixed traffic conditions. By contrast, DOG is very efficient, sometimes even dominating GreedyReclaim. CICL is a worthy opponent of DOG, usually having a lower variance. However, this could change for bigger data sets where DOG would be change to a more variance-reducing behavior as the SSD would be more worn out. GreedyVariance does a very good job for wear-leveling, however, its total amount of reclaims is far from what is achievable with DOG or GreedyReclaim.

	CAT	DOG	CICL	GreedyReclaim	GreedyVariance
Uniform	103721, 0.4929	103684, 0.7275	103691, 0.7626	103621, 2.0601	105078, 0.1102
Linslant Dynamic	50754, 5.9802	50726, 2.9854	50708, 10.557	50721, 23.939	51381, 0.0462
Linslant Standard Cold	180, 0.2857	181, 0.2679	183, 0.4107	181, 0.2679	181, 0.2679
Linslant Standard Hot	57018, 5.4066	57010, 3.0311	56988, 3.6263	56964, 24.932	57802, 0.0943
Linslant Desnoyers Cold	77, 0.2333	77, 0.2333	79, 0.1000	77, 0.2333	77, 0.2333
Linslant Desnoyers Hot	82233, 8.3218	82168, 7.4459	82158, 5.8920	82187, 70.439	83935, 0.0696
01_1 Dynamic	11099, 156.51	5495, 80.028	9515, 11.880	5680, 88.083	10316, 4.7819
01_1 Standard Cold	12360, 375.55	2544, 5.4161	4841, 2.1865	2584, 6.5261	4774, 0.6569
01_1 Standard Hot	5302, 774.98	5101, 1565.9	5225, 489.66	5206, 2090.6	6486, 5.1143
01_1 Desnoyers Cold	12964, 362.84	2927, 7.4761	5430, 3.2519	2864, 7.9138	5658, 0.8375
01_1 Desnoyers Hot	1327, 144.66	1300, 95.474	1333, 80.029	1302, 460.62	1526, 0.2211
03_1 Dynamic	17355, 6.7652	14262, 10.008	16738, 2.0535	14232, 10.197	20121, 0.2217
03_1 Standard Cold	735, 64.207	311, 5.7102	440, 7.8653	313, 5.8635	279, 0.4791
03_1 Standard Hot	20578, 23.621	16073, 6.0558	18920, 2.6623	16021, 9.0799	23315, 0.2152
03_1 Desnoyers Cold	219, 9.2034	99, 0.7926	144, 0.4626	99, 0.8296	123, 0.1838
03_1 Desnoyers Hot	35564, 42.627	29981, 13.075	34819, 5.9079	29581, 21.832	46963, 0.1745
03_0 Dynamic	116324, 2908.3	93836, 2134.7	173393, 268.39	97218, 4167.6	155972, 195.62
03_0 Standard Cold	297987, 45650	15945, 111.02	36916, 60.557	16084, 120.69	28215, 43.327
03_0 Standard Hot	493278, 48641441	91031, 105159	92379, 62633	91610, 442659	114966, 1.8571
03_0 Desnoyers Cold	301095, 48126	16451, 121.09	39060, 80.655	16794, 132.34	29731, 56.297
03_0 Desnoyers Hot	368185, 73116525	37800, 5620.1	37578, 12103	37413, 28328	49522, 0.1311

Table 4.1: The evaluation of CAT, DOG, CICL, GreedyReclaim and GreedyVariance under various conditions (input vector and subdivision). The over-provisioning factor is $\alpha = 1.07$. Dynamic stands for no subdivision, Desnoyers for the Desnoyers subdivision and Standard for two pools with the same α. For each score, the results are given in two numbers: the first number shows total amount of reclaims and the second indicates the variance between the life cycles of the blocks. In the case of Desnoyers and Standard, two rows are displayed as we have a hot and cold pool, meaning the sum of reclaims would equal the total reclaims.

In order to analyze which is the best score in all the conditions, table 4.1 has been summarized in table 4.2.The column shows how many times a specific score is dominated by each of the other scores. The rows show how many times a specific score dominates the other scores. A score dominates another score when both of its results (total number of reclaims and variance of lifecycles of the blocks) are lower than the results of the other score. We can see three important results:

1. The worst score is the CAT score as it is dominated in 51 instances by the other scores and dominates only in one instance another score.
2. CICL and GreedyReclaim dominate roughly as much as they are dominated.
3. GreedyVariance and DOG are never or very rarely dominated. However, DOG outperforms GreedyVariance as it dominates the scores in 29 cases against only 18 cases for GreedyVariance, roughly 40% more.

This illustrates again the outstanding performance of the DOG score.

	CAT	DOG	CICL	GRec	GVar	Overall dominating
CAT	-	0	1	0	0	1
DOG	16	-	3	10	0	29
CICL	15	1	-	2	0	18
GreedyReclaim	11	0	2	-	0	13
GreedyVariance	9	1	7	1	-	18
Overall dominated	51	2	13	13	0	

Table 4.2: Summary of scores

Chapter 5
Conclusion

The simple fact that writing is page-wise, but deleting block-wise, generates a very complex problem. In an attempt to minimize both variance and total number of reclaims, or in other words to wear out the SSD evenly while keeping write amplification low, several simplifying assumptions have been made in the literature (uniform hot and cold traffic).

We show that contrasting uniform traffic with Linslant traffic can yield interesting insights; we also propose DOG, a dynamic score computation.

Two essential findings have been made, both based on heuristics:

1. Dividing an SSD into a static and dynamic part can yield a better result than any possible subdivision using static, cold and hot. This is due to the fact that Linslant traffic can outweigh the hot-cold uniform traffic approach;
2. Under dynamic real-life conditions, our new DOG score outperforms both CICL and CAT scores.

Based on this paper, future research could investigate the nature of Linslant traffic in an attempt to analytically derive its characteristics like Desnoyers did with uniform traffic. Also, the process on how to incorporate Linslant data alongside hot and cold data could be explored. Investigating DOG under different models would yield better insights into its suitability for different access patterns.

Bibliography

[1] Desnoyers, P.: Analytic modeling of ssd write performance. In: Proceedings of the 5th Annual International Systems and Storage Conference. (June 2012)

[2] Hu, X.Y., Eleftheriou, E., Haas, R., Iliadis, I., Pletka, R.: Write amplification analysis in flash-based solid state drives. In: SYSTOR '09 Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference Article No. 10. (May 2009)

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

[4] Kim, H., Lee, S.: An effective flash memory manager for reliable flash memory space management. In: IEICE Trans. Inform. Syst. E85-D, 6: 950-964. (November 2002)

[5] Narayanan, D., Donnelly, A., Rowstron, A.: Write off-loading: Practical power management for enterprise storage. In: Proc. 6th USENIX Conference on File and Storage Technologies. (July 2008)