Saving Bit-flips through Smart Overwrites in NVRAM

Arockia David Roy Kulandai, *Student Member, IEEE*, Thomas Schwarz, *Senior Member, IEEE*

***Abstract*—New generations of non-volatile random access memories will combine the best features of memory (access times, byte addressability) with the best features of storage (non-volatility, low costs per byte). Some, like PCM, have a limited endurance. All will only consume energy when accessed, but writes will use much more energy than reads. These characteristics put a cost on flipping bits in memory. Bit-flip aware data structures lower the number of bits flipped by not resetting fields to zero to indicate a deleted record but by using bit-maps. If given a choice of where to over-write data, they will select the location which results in a lower number of bit-flips. We calculate the expected bit-flip savings of this strategy and derive a rule to determine the number of the possible candidate locations.**

***Index Terms*—PCM endurance, NVRAM, Smart writes for PCM.**

I. INTRODUCTION

Processed information continues to grow exponentially [16]. The emergence of Non-Volatile RAM (NVRAM) technologies that combine the advantages of storage (non-volatility, low-costs, large size) and of memory (fast access times, byte-addressability) allow systems to combine the functions of memory and storage in a single layer. These types of NVRAMs do not use energy when their data is at rest. Writes typically use much more energy than reads. Some, like Phase Change Memory (PCM) have limited endurance for over-writes. (Their endurance is more than sufficient for use as main memory as long as the over-write load is decently distributed over a large memory. As memories in the Terabyte range are affordable, this is not a problem.) These behaviors put a premium on bit-flip avoiding behavior.

A large number of schemes to save bit-flips in hardware exists. Fundamental is *Data Comparison Write* (DCW) that eliminates redundant bit writes by first reading the word before writing it and only setting and resetting bits that need to be changed [18], [19]. On the software side, Bittman and colleagues [3], [4] observe that data structures can save considerably on the number of bit-flip operations. One ingredient of these bit-flip aware data structures is to try to overwrite new data with stale data of roughly the same type. Slight encoding can increase the effect for web-content [10] and for pointers [11]. Besides observing that storing pointers as the result of an exclusive-or with another pointer, Bittman *et al.* found that bit-flips can be saved if a data structure does not invalidate keys by zeroing them out but by using a bit to indicate whether a key entry exists or not. Unfortunately, they did not elaborate on this observation.

In this article, we investigate the amount of savings to be had by using an "is-valid" bit-array instead of using overwrites. We also follow up on another suggestion by Bittman, namely that selecting a candidate stale key among a set of keys can lead to additional savings. We therefore determine experimentally the number of bit-flips if we choose the best key among k to overwrite, with k varying from 2 to 10. We can then use these numbers to determine the best strategy for saving bit-flips. Because processors communicate with memory through several levels of cache, we investigate whether loading additional cache lines in order to find better candidates for overwrites is advantageous.

Our goal is to understand how the internal character of data interacts with bit-flip pressure, not to build a new data structure. The latter is the ultimate goal. We contribute to it by trying to understand the fundamental building blocks. As a consequence, our setup is not a complete data structure, but a test bed to answer the question how much better it is to keep stale data (marked as such by a valid-bit) as opposed to zeroing it out. For starters, zeroing out deleted keys has the advantage of preventing reading deleted keys so that we are protected against software faults.

In the following, we first describe our data structure. We then discuss the case of uniformly distributed random bit-strings. To start our experimental work, we first discuss the impact of encoding of non-Latin alphabets. We then present our results using a number of data sets, using different natural languages, and also a floating point key. We did not try out integer keys such as social security numbers or telephone numbers, as they were not available for privacy reasons. We then verify our data by a closer simulation using data from Amazon product reviews. We then calculate the optimal energy saving strategies.

II. SETUP

There are many data structures that implement a key-value store, with key-based operations of insert, delete, look-up, and update. The various types of B-trees also implement a range query (for a range of keys). The importance of B-tree can hardly be exaggerated.

In contrast to the B-tree in general, B-tree node implementation has received less interest [8]. Early on, the prefix B-tree used prefix and suffix compression to place more keys in a node and therefore achieve better performance [2], [14]. B-tree nodes that do not use key compression often

D. Roy and T. Schwarz are with the computer science department at Marquette University, Milwaukee, Wisconsin, USA.

E-mails: david.roy@sxca.edu.in and thomas.schwarz@marquette.edu.

DOI: 10.36244/ICJ.2022.4.5

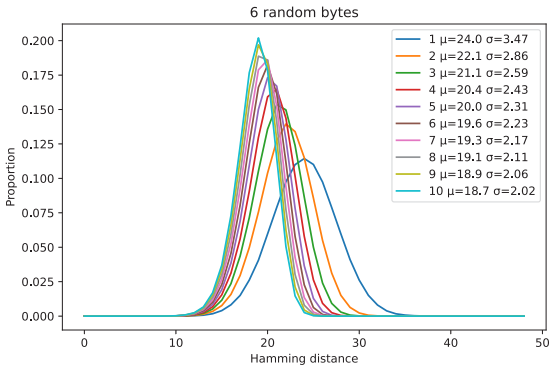

Fig. 1: Average number of bit-flips when overwriting a uniformly distributed, random 6B key. We select the key among k keys, $k = 1,2,...,10$.

place fixed-length keys in a contiguous array. Our results in what follows show that marking a key as invalid, and then overwriting it with a new key, results in bit-flip savings. Some node implementations will keep keys in order, which almost eliminates the chance to be able to select between two stale keys for overwrite. Other implementations already facilitate key insertions by using an auxiliary data structure to preserve the order. In this case, all of our experimental results tell us what bit-flip savings can be achieved. A thorough investigation of B-tree node structures, their interaction with caches, and their relations to bit-flip savings is left to the future. The clfB-tree [9] for example packs B-tree nodes into a cache line but does not consider bit-flips.

Key-value stores are of course not limited to B-trees and their derivatives. They can be based on hashing or other types of trees. In the context of NVRAM, we might store many sets of pairs of keys and pointers to records. Incidentally, the bit-flip aware manipulation of pointers is a different issue [11]. We now study in more detail the bit-flip behavior of the key portion of such a data structure. First, we consider the case of random keys, not because this is a frequent use case, but to set a base line.

III. The Random Case

We now study the expected number of bit-flips overwriting a fixed length key or overwriting the best of k candidate keys. This is a value that depends on the population of possible keys.

The simplest model for the keys is a string of random bits, where each bit is set with probability 50%. The Hamming distance between two such keys is binomially distributed with parameters n, the length of the key, and probability $p = 0.5$. The minimum Hamming distance between one such key and k other keys is the first order statistics of binomial distributions. If $\mathbb{B}(k;n,p)$ is the Cumulative Distribution Function (CDF) of the Binomial distribution with n and p, i.e.

$$\mathbb{B}(\nu;n,p) = \text{Prob}(X \leq \nu) = \sum_{i=0}^{\nu} \binom{n}{i} p^i (1-p)^{n-i}$$

TABLE I
Expected minimum of k normally distributed random values with mean $\mu = 24$ and standard deviation $\sigma = \sqrt{48/4}$ and exact numbers for a Binomial distribution with parameters $n = 48$ and $p = 0.5$.

k	Expected Value Normal Appr.	Expected Value Exact
1	24.0000	24.0000
2	22.0456	22.0507
3	21.0684	21.0760
4	20.4341	20.4442
5	19.9714	19.9838
6	19.6103	19.6250
7	19.3159	19.3328
8	19.0685	19.0875
9	18.8558	18.8767
10	18.6696	18.6925

then the CDF $\Phi(\nu)$ of the minimum out of k is given by

$$\begin{aligned} 1 - \Phi(\nu) &= \text{Prob}(\min(X_1, X_2, \ldots X_k) > \nu) \\ &= \prod_{i=1}^{k} \text{Prob}(X_i > \nu) \\ &= (1 - \mathbb{B}(\nu;n,p))^k. \end{aligned}$$

For reasonably small values, the exact formula can be evaluated. The binomial distribution can be approximated well with a normal distribution, but the order statistics for independently and identically distributed normal distribution only has a closed form even for the expectation for very small values of k [1].

We give the values of the expectation of the minimum of k normally distributed independent random variables with parameters $\mu = 0.5 \times 48$ and $\sigma = \sqrt{48 \times 0.5^2}$, i.e. where $n = 48$, which is the approximation for our experimental data in Table I.

Even in the random case, we save bit-flips by not zeroing out deleted keys. If we delete a key and then insert another one and if we use the valid-bit array, the valid-bit array itself has one bit set and reset (2 flips) and the expected costs of overwriting the key (of length 6B) is 24 flips. Zeroing out costs 24 bit-flips and overwriting costs also 24 bit-flips for a total of 48 bit-flips.

IV. Experimental Data

The efficiency of overwriting stale keys instead of zeroing out depends on the nature of the keys. We now gather experimental data on various data-sets. The most important class of keys that do not behave like random numbers are strings of characters. To avoid an anglo-centric view, we first discuss non-Latin alphabets. Unfortunately, our lack of knowledge of Chinese does not allow us to test for keys taken from this important language. We then use several data-sets with different types of keys in different languages to determine their bit-flip propensity. Finally, we use a different dataset to confirm the predictions based on our measurements for a closer simulation of a hypothetical data structure made up of key – pointer to record entries.

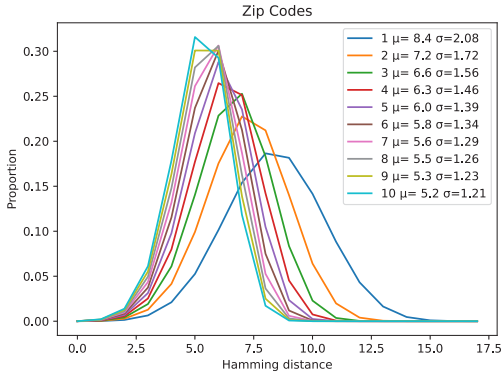

Fig. 2: Average number of bit-flips when overwriting a 4B zip code from the credit industry complaint data set.

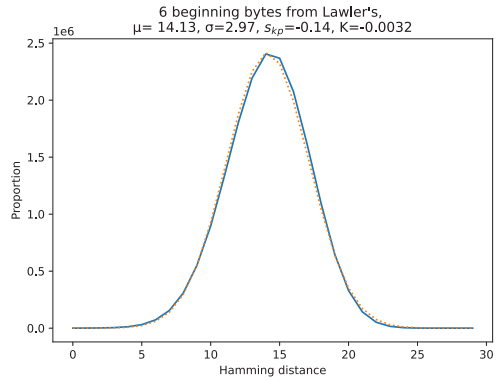

Fig. 3: Average number of bit-flips when overwriting a key in the Lawler corpus with another one and comparisonwith the normal distribution with the same mean and standard deviation.

A. Non-Latin Alphabets

At the byte level, encodings matter. Given its importance, we concentrate on keys encoded with utf-8, a version of Unicode very popular for web-documents. The utf-8 encoding is very efficient for English text, as the English character set is encoded just as the lower half of ASCII. For German, French, Spanish, or other languages using a Latin character set, the relatively infrequent letters with accents and Umlauts are stored in two bytes.

For non-Latin alphabets such as Tamil's Dravidian and Hindi's Devanagari script, utf-8 is not space efficient. Both scripts vary 7 bits encoded within three bytes for each character. In contrast, the less common utf-16 only uses two bytes for each Dravidian or Devanagari character. The *Standard Compression Scheme for Unicode* (SCSU) defined in the Unicode Technical Standard Nr. 6 uses *dynamically positioned windows* so that characters belonging to small scripts such as Devanagari can be encoded in a single byte [6], [7]. As an alternative to SCSU, Vijayalakshmi and Sasirekha propose to map the Tamil characters to the upper half of the ASCII encoding, i.e. between 0x80 and 0xff, characterized by the first bit being set [17]. While such a compression scheme uses space more efficiently, the costs of compression and decompression might mitigate against their use. If such a compression scheme is used, keys in Tamil or Hindi behave like keys in English or German. If instead utf-8 is used, the larger number of bytes to be read increases the read energy consumption, but in general, overwrites are close in efficiency to that of English text. Only the interference of punctuation marks, white spaces, or ASCII numerals cause alignment issues and generate more bit-flips and hence higher write energy use.

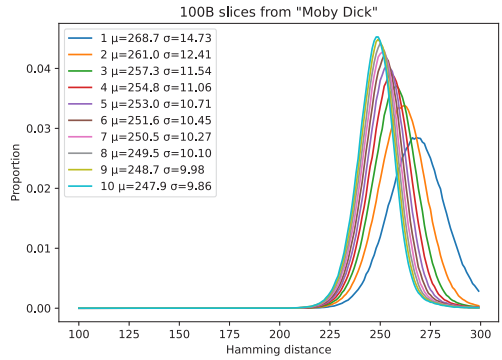

Fig. 4: Average number of bit-flips when overwriting a 100B slice with the best of a set of k, k = 1,2,...,10 randomly selected slices not containing the original slice.

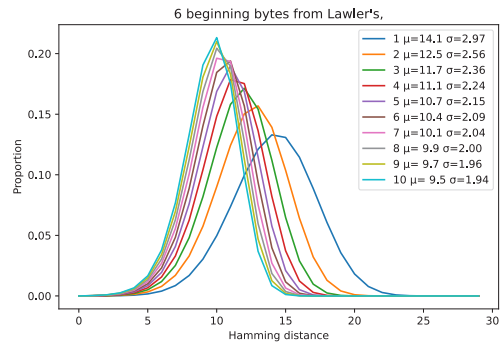

Fig. 5: Average number of bit-flips when overwriting a 6B key taken from the first 6 bytes of the words in Lawler's vocabulary list with the closest of k keys from the same source, where k varies k = 1,2,...,10.

B. Results

We used the following corpora for our experiments:
(1) The zip codes from a Kaggle dataset collected by A. Kumar on consumer complaints of financial products from the Consumer Financial Protection Bureau (CFPB) Open Tech site [12]. There are 26800 unique zip-codes, stored as integers in four bytes. Since the largest zip-code is 99999, which is in hexadecimal 0x1869f, only 17 bits are ever set.
(2) The novel "Moby Dick" from Project Gutenberg, downloaded as utf-8. We divided the novel into slices of 100 bytes each.

TABLE II
FREQUENCY OF BITS SET IN A BYTE

Bits	0	1	2	3	4	5	6	7	Total Bits Set	δ
Zip codes	0.44	0.33	0.33	0.34	0.34	0.34	0.32	0.28	2.12	2.12
Zip codes (last 16 b)	0.50	0.50	0.50	0.50	0.51	0.51	0.49	0.43	3.93	3.90
German	0.56	0.49	0.53	0.33	0.33	0.85	0.98	0.04	4.08	2.24
German (miniscules)	0.56	0.46	0.53	0.33	0.33	0.98	0.98	0.04	4.20	2.11
English (Moby Dick)	0.45	0.34	0.48	0.34	0.25	0.96	0.78	0.00	3.60	2.34
English (Lawler)	0.59	0.40	0.54	0.40	0.31	1.00	0.99	0.00	4.22	2.00
Tamil (Tirukkural)	0.28	0.48	0.52	0.51	0.14	0.85	0.33	1.00	4.12	2.37
Tamil (Agananuru)	0.28	0.47	0.52	0.51	0.14	0.85	0.33	1.00	4.10	2.45
Hindi (Bible)	0.22	0.14	0.48	0.18	0.12	0.91	0.33	0.90	3.28	1.65
Hindi (Ambedkar)	0.21	0.16	0.53	0.18	0.15	0.87	0.33	0.99	3.49	1.76
Earthquakes	0.41	0.42	0.41	0.38	0.40	0.42	0.60	0.39	3.44	3.25

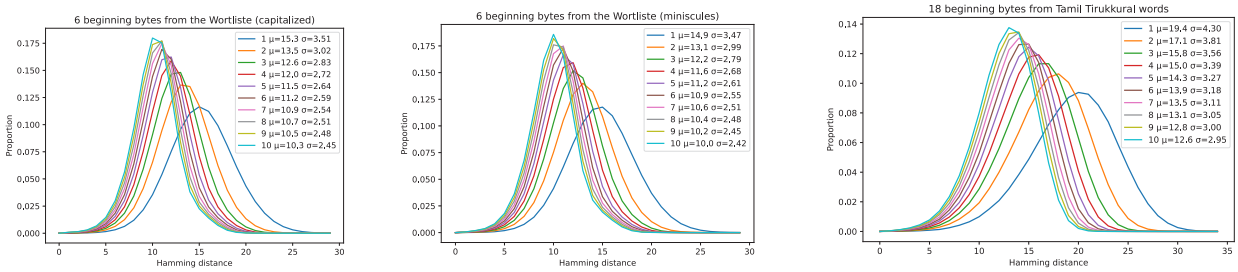

Fig. 6: Average number of bit-flips when overwriting a 6B key taken from the first 6 bytes of the words in Davidak's list with the closest of k keys from the same source, where the number k of candidates varies between 1 and 10. The graph on the left shows the results with upper case letters, the one in the middle with upper case letters converted to minuscules, and the one on the right from the Tirukkural.

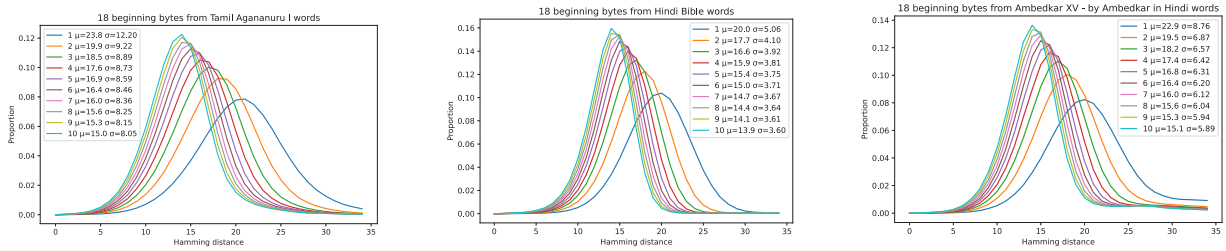

Fig. 7: Average number of bit-flips when overwriting a 6B key taken from the first 6 symbols (or 18B) from long words in Agananuru (left), a Hindi translation of the Old Testament (middle) and a volume of Dalit-leader B.R. Ambedkar (right).

(3) A list of English words by Lawler [13]. We used the first 6 bytes of each word at least this long.
(4) A word list *Wortliste* of German words collected by the pseudonymous Davidak [5]. We used the first 6 bytes of each word at least this long.
(5) The same list moving all capital letters to miniscules.
(6) A list of beginnings of words in Tamil. We used Tirukkural, the classic collection of poems from the 1st century by Thiruvalluvar in utf-8 format. We extracted the first six characters, i.e. first 18B, but ensured uniqueness.
(7) A similar list of beginning of words in Tamil from Agananuru, another poetry collection.
(8) A list of beginnings of words in Hindi. We used a translation of part of the Old Testament in the Bible.
(9) A similar list in Hindi taken from Volume 15 of Ambedkar's writings.
(10) Longitude and latitudes stored as floating point numbers from a list of major earth-quakes from 1974 to 2001. We mixed longitudes and latitudes and removed duplicate values from the list.

We use the zip codes as keys as an example where a "real-life" data-set has keys that are essentially random, where each bit is distributed with a Bernoulli distribution with parameter $p = 0.5$. A zip code stored as an integer uses two bytes and additionally the least significant bit of a third byte. As we can see from Figure 2, the frequency distributions closely resemble the one from Figure 1. Indeed, the means of the minima of k Binomial distributions with parameters $n = 17$ (corresponding to the 17 bits used for a zip code) and $p = 0.5$ are 8.5, 7.34541, 6.76812, 6.39478, 6.12341, 5.91242, 5.74099, 5.59733, 5.47414, and 5.36662 for $k = 1, \ldots, 10$.

Another example for this behavior would be keys derived from a good hash function such as MD5 or the SHA series.

We first calculated the bits set in each text corpus, Table II. The number for Latin alphabets is quite close, but still reflects differences. The numbers for the non-Latin alphabets are quite different. From these numbers alone, we can understand why overwriting text from one corpus with text from another corpus does not flip on average four bits per byte. Depending on the corpus, we can also define a most likely byte. Instead of replacing a deleted key with zero bytes, we could replace it with this most-likely byte. Unfortunately, this is somewhat dangerous, as the most likely byte is a normal letter, the character 'a' for German and English, and not special, like the zero byte. The last column of Table II gives the average distance δ of a byte in the corpus to the most-likely byte. Overwriting stale keys with a sequence of this most-likely byte costs 2δ bit-flips, resulting in strong savings compared to a policy of zeroing out stale keys. In the case of keys in a non-Latin alphabet, stored as a combination of bytes, we still advocate the use of a single most likely byte in order to avoid alignment problems such as those resulting from using (ASCII) digits or punctuations within the key, as the latter are encoded as a single byte.

Next, we calculate the expected number of bit-flips of overwriting with a new key, either a single, stale key or the closest (according to the Hamming distance) key of k candidate stale keys. We determined the Hamming distance between each key and the best of a sample of k candidate keys for a total of 300 samples per key. The results are given in Figures 5, 4, 6, and 7. The results look remarkably similar. Especially for $k = 1$, where we just overwrite a single key, the distribution looks remarkably similar. Figure 3 compares the value for the Lawler corpus with a normal approximation. (The Lawler distribution is discrete, and the normal approximation uses the difference between the CDF of the normal distribution at $i + 0.5$ and $i - 0.5$ as the discrete PDF, as is usual.) While optically, the approximation is very good, but skew and kurtosis are significantly different and as the χ^2 value is over $20,000$, the frequency distribution is definitely not normally distributed. When we look at the mean of order statistics (e.g. the minimum of k independently and identically distributed random variables,) then the difference also becomes obvious.

The numbers for the earthquake data set have peculiarities that we can ascribe to the nature of representations of floating point number. The curve for $k = 1$ is tri-modal and all other ones are bi-modal. Also, the decrease in the expected value μ_k of the bit-flips when selecting from k candidates is less pronounced. Despite these differences, the overall picture remains roughly the same.

The savings obtained by using more candidate keys for overwriting are remarkably similar. If the number of bit-flips has mean μ and a standard deviation of σ, then selecting the best of four candidate key fields (with previously deleted keys) lowers the expected number of bit-flips to $\mu - \sigma$. Moving to the best of ten candidate fields does not lower the expected number of bit-flips to $\mu - 2\sigma$.

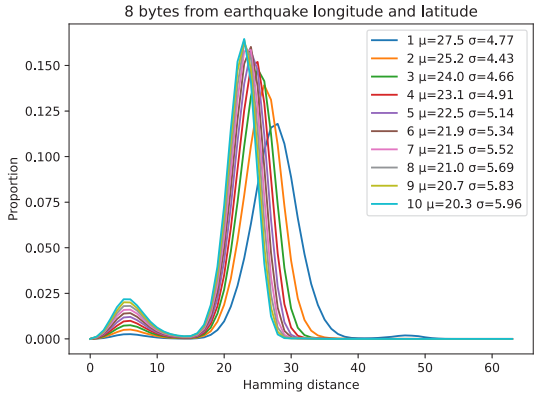

Fig. 8: Average number of bit-flips when overwriting a 8B key formed by a floating point number representing the longitude and latitude of epicenters of major earthquakes from 1970 to 2014. We select the best of k candidate keys, where k varies from 1 to 10.

C. Confirmation

In our fixed sized bucket of key – pointer to record data structure keys marked as deleted stay until they are overwritten. Therefore, an outlier from the population might stay much longer than expected. We call this the *persistent outlier* phenomenon. Basically, the set of candidate deleted keys is no longer random in such a bucket.

To test this, we use a data set downloaded from Kaggle [15] that contains product reviews. We used the 14B long Amazon user identifiers as our key stand-ins. When we simulated a single slot by overwriting the current user ID with the following user ID, we notice that there is a visible deviation from the normal bell-shaped frequency curve.

We then simulated the behavior of a small bucket with k keys. After filling up the bucket, all keys are deleted. A record with a certain key is inserted and in a short time deleted. When the key is inserted, we look for the nearest deleted key according to the Hamming distance. We calculated the number of bit-flips for this scenario going through all the user IDs. As we can see behavior does not quite match the previous data sets. The decrease in the mean of number of bit-flips is less pronounced.

A comparison with the minimum number of bit-flips when overwriting one of the candidates in a k-element set, Figure 9, bottom, does show some deviations. The calculated standard deviation of the bit-flip numbers σ is considerably and consistently smaller. On the other hand, the means are almost equal. Thus, the persistent outlier is not a problem, at least not for this data set.

V. Recommendations

We now combine our results to evaluate performance. We measure the expected costs for each strategy in multiples of the energy of one bit being written. For example, if the read energy is one tenth of the energy of a write, then writing 15 bits and reading 20 bits costs the energy spent to write to 17 bits.

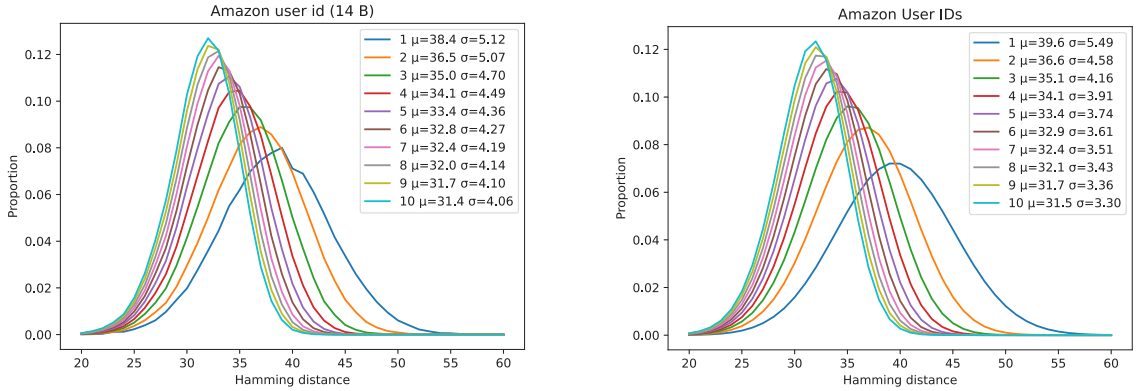

Fig. 9: Observed number of bit-flips for Amazon User IDs. The left shows the numbers when actually using the scheme, the right shows the numbers when using random elements.

We first consider the case where data on the NVRAM is accessed directly, without a cache. This might be the case for an embedded device. The costs for the zeroing strategy are simply twice the average number of bits set per byte times the size of the key. The costs for checking k candidate state keys to find the write victim to overwrite are 2 bits in the valid-bit field. The number of bits written is determined by the corresponding experimental value. We need to add to this the costs of reading $k-1$ candidates. The cost of reading the first is already included in the write costs as we are using read before write. If ρ is the ratio of read over write energies, then this adds $\rho \times (k-1)$ times the size of the key to our energy bill.

Figure 10 shows the energy costs for uniformly distributed random keys of length 4B. The zeroing strategy loses against the simple overwrite strategy. If reads take much less energy than writes, comparing a key to be inserted with up to five stale keys makes sense. As the read energy increases slightly, the number of comparisons goes down. Figure 11 gives the numbers for 6B keys derived from Lawler's word list, representing keys that are English strings. Here, zeroing is even less attractive. Depending on the costs of reads over writes, it makes sense to read up to four stale keys. Note in both cases that a more typical value for ρ is 0.15, so that the recommendation in both cases is to only select one stale key and over-write it.

Our argumentation breaks down, of course, if caching is used. Whenever we access a byte, we will cause a cache line's worth of data to be read. When the cache line is written back to NVRAM, only the bits that have changed are actually written, leading to the use of energy. To exploit the use of caching, our data structure needs to be cache-line aware. Presumably, the valid-bit array is part of the header of the bucket data structure, which will need to be read for every access. We can also assume that bucket data structures are cache-line aligned. Thus, each bucket will consist of a number of sub-buckets, each contained in a single cache-line.

If we want to insert a key, and there is at least one slot in the first sub-bucket, we have to decide whether we want to read more sub-buckets with open slots. When we do so, each sub-bucket accessed costs us read energy. As before, let us denote by rho the ratio of the energy costs of reading a bit over the energy costs of writing a bit. A typical cache-line has 64B or 512b. If we decide to load a sub-bucket of this size into cache, we have 512ρ costs. Denote the expected number of bits written after finding the best of k candidates, $1 \leq k$, with μ_k. Then on the other side of the ledger, if we have already located i stale data items and now decide on whether to read a sub-bucket with j stale data items, then reading the sub-bucket will save $\mu_i - \mu_{i+j}$ writes, but costs the equivalent of 512ρ bits written. Thus, we should read a sub-bucket if

$$\mu_i - \mu_{i+j} > 512\rho.$$

Since a typical value for ρ is $1/5$ or more for PCM, the costs savings have to be at least 100 bits written. A review of our experimental results suggests that savings of more than 1b per byte key length is unrealistic. In Figure 12, we illustrate this decision procedure. If i candidate keys are already read, we select the unread sub-bucket with the most number of keys. Assume that there are j keys in it. We then calculate the minimum number of the ratio of write energy per bit over read energy per bit (that is $1/\rho$) to save bitflips by reading the yet unread sub-bucket. Of course, if we read the beginning of the bucket and do not find a candidate for overwriting there, we will have to read a sub-bucket with a candidate. If there is no such sub-bucket, we will have to create an overflow page. The values for $1/\rho$ are almost exclusively quite high and all are higher than for currently proposed NVRAM technologies.

We give six examples in Figure 12, namely keys from Lawler's corpus, keys of 18B that behave like the keys in Lawler's corpus, keys of 18B that behave like the German word list, keys of 6B that taken from the capitalized version of the German word list, keys of 18B from the Hindi Ambedkar collection, and 18B keys from our first Tamil corpus. The

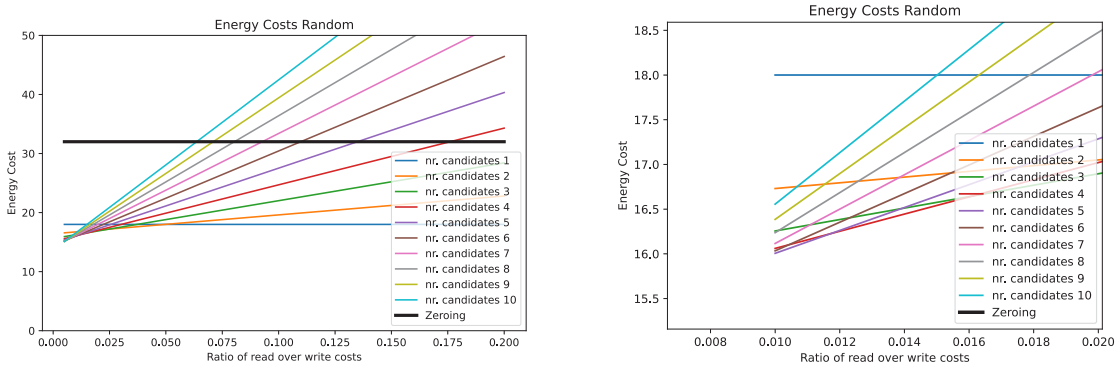

Fig. 10: Energy costs (in multiples of the energy of one bit written) using the Zeroing strategy and the best out of k overwrite strategy for random keys of length 4 bytes. The right is a blowup of the left figure.

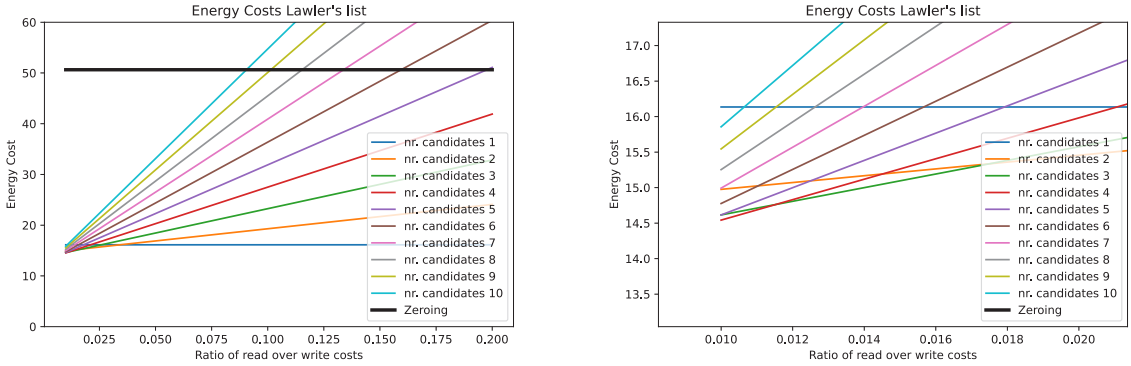

Fig. 11: Energy costs (in multiples of the energy of one bit written) using the Zeroing strategy and the best out of k overwrite strategy for 6B keys derived from Lawler's word list. The right is a blowup of the left figure.

minimum write energy needed to justify looking at another sub-bucket when one has already been located is at least 46. This happens for the German-like data set when we have one candidate and find a sub-bucket containing 4 stale keys. Even this amount is larger than for current and projected PCM memories. In all other cases, the discrepancy is much larger. We conclude from our experimental data that a cache aware algorithm trying to overwrite a stale key with a fresh, valid one should not access keys currently not in cache. Of course, this limits the bit-flip savings of such an algorithm.

VI. Conclusions

In this study, we investigated the bit-flip behavior of overwrites for typical keys. Our first take-away is instead of zeroing stale keys (or data in general), stale data should be marked and overwritten by new data. This confirms similar observations for pointers [11]. Alternatively, any key populations (such as words or names) have sufficient internal structure that zeroing out stale keys can be replaced by overwriting with the "most likely byte". Keys derived from texts in non-latin languages should be compressed. A cache length aware algorithm that is trying to overwrite a stale key should *not* load additional data into cache in order to find a better replacement. Overall, Bittman's observation and proposal have shown to be sound for a large variety of experimental data.

In general, the behavior of keys in our context is remarkably similar across different data sets. The observed distributions are very similar to a normal distribution and so are the order statistics. This gives us confidence in believing that the design of a bit-flip aware data structure will be valid across a wide range of key (and presumably data) populations.

We have not integrated these observations into the design of a bit-flip aware dictionary data structure. Neither did we investigate a bit-flip aware node structure for B-trees. This is left to future work.

References

[1] M. Ahsanullah, V. B. Nevzorov, and M. Shakil, *An introduction to order statistics*. Atlantis Press, 2013, DOI: 10.2991/978-94-91216-83-1.

[2] R. Bayer and K. Unterauer, "Prefix B-trees," *ACM Transactions on Database Systems (TODS)*, vol. 2, no. 1, pp. 11–26, 1977, DOI: 10.1145/320521.320530.

[3] D. Bittman, M. Gray, J. Raizes, S. Mukhopadhyay, M. Bryson, P. Alvaro, D. D. Long, and E. L. Miller, "Designing data structures to minimize bit flips on NVM," in *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2018, pp. 85–90, DOI: 10.1109/NVMSA.2018.00022.

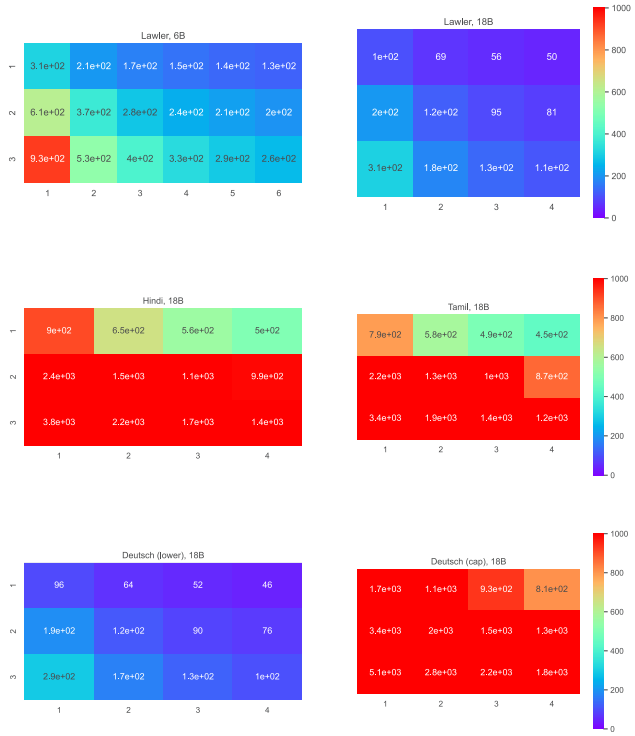

Fig. 12: Recommendations for reading a sub-bucket. The heatmap gives the minimum energy costs of writes over reads ($1/\rho$) making one read a sub-bucket. The x-axis gives the number of candidates in the new sub-bucket and the y-axis gives the number of candidates already available.

[4] D. Bittman, D. D. Long, P. Alvaro, and E. L. Miller, "Optimizing systems for byte-addressable NVRAM by reducing bit flipping," in *17th USENIX Conference on File and Storage Technologies*, 2019, pp. 17–30, https://dblp.org/db/conf/fast/fast2019.html#BittmanLAM19.

[5] Davidak, "Wortliste," 2016, accessed May 2022. [Online]. Available: github.com/davidak/wortliste

[6] D. Ewell, "A survey of unicode compression," 2004, www.unicode.org/notes/tn14/UnicodeCompression.pdf. [Online]. Available: https://www.unicode.org/notes/tn14/UnicodeCompression.pdf

[7] A. Gleave and C. Steinruecken, "Making compression algorithms for unicode text," 2017, arXiv:1701.04047.

[8] G. Graefe and H. Kuno, "Modern B-tree techniques," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 1370–1373, DOI: 10.1561/1900000028.

[9] W.-H. Kim, J. Seo, J. Kim, and B. Nam, "clfB-tree: Cache line friendly persistent B-tree for NVRAM," *ACM Transactions on Storage (TOS)*, vol. 14, no. 1, pp. 1–17, 2018, DOI: 10.1145/3129263.

[10] A. D. R. Kulandai and T. Schwarz, "Content-aware reduction of bitflips in phase change memory," *IEEE Letters of the Computer Society*, vol. 3, no. 2, pp. 58–61, 2020, DOI: 10.1109/LOCS.2020.3018401.

[11] ——, "Does XORing pointers save bitflips for NVRAM?" in *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2021, pp. 1–6, DOI: 10.1109/MASCOTS53633.2021.9614290.

[12] A. Kumar, "Dataset: Consumer complaints: financial products, from the consumer complaint database maintained," 2020, accessed May 2022. [Online]. Available: www.kaggle.com/datasets/ashwinik/consumer-complaints-financial-products

[13] J. Lawler, "An English Word List," 1999, accessed May 2022. [Online]. Available: http://www-personal.umich.edu/~jlawler/wordlist.html

[14] D. Lomet, "The evolution of effective B-tree: Page organization and techniques: A personal account," *ACM SIGMOD Record*, vol. 30, no. 3, pp. 64–69, 2001, DOI: 10.1145/603867.603878.

[15] M. G. Jillani Soft Tech, "Amazon product reviews," 2022, Kaggle dataset, accessed May 2022. [Online]. Available: www.kaggle.com/datasets/jillanisofttech/amazon-product-reviews

[16] M. Nagy, J. Tapolcai, and G. Rétvári, "R3d3: A doubly opportunistic data structure for compressing and indexing massive data," *Infocommunications Journal*, vol. 11, no. 2, June 2019, DOI: 10.36244/ICJ.2019.2.7.

[17] B. Vijayalakshmi and N. Sasirekha, "Lossless text compression for Unicode Tamil documents," *ICTACT Journal on Soft Computing*, vol. 8, no. 2, pp. 1635–1640, 2018, DOI: 10.21917/ijsc.2017.0227.

[18] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 2007, pp. 3014–3017, DOI: 10.1109/ISCAS.2007.377981.

[19] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Adurable and energy efficient main memory using phase change memory technology," in *Proceedings of the 36th International Symposium on Computer Architecture (ISCA'09)*, 2009, DOI: 10.1145/1555815.1555759.

Arockia David Roy Kulandai SJ David Roy is a Doctoral student of Computer Science at Marquette University, Milwaukee, WI, USA. Dr. Thomas Schwarz is his advisor. He has graduate degrees in Computer Applications and Philosophy. He was the Director of Xavier Institution of Computer Appli- cations (XICA) and Vice Principal of Department of Computer Sciences at St. Xavier's College (Autonomous), Ahmedabad, Gujarat, India from 2013 - 2018.

Thomas Schwarz SJ Dr. Thomas Schwarz has PhDs in Mathematics from Fern-Universität Hagen, Germany and Computer Science from University of California, San Diego. He is an associate professor in the department of computer science at Marquette University, Milwaukee, WI, USA and at Xavier Institute of Engineering, Mumbai, India. He is an adjunct professor at Santa Clara university, CA. He has taught at the Universidad Católica del Uruguay and at the Universidad Centro-Americana in El Salvador. He has over 125 publications to his credit and his research interest include Scalable Distributed Data Structures, Large Scale Storage Systems, High Availability, Erasure Correcting codes, security and Non-Volatile Memories.