HAL
open science

Improvement of the tolerated raw bit-error rate in NAND Flash-based SSDs with the help of embedded statistics

Valentin Gherman, Emna Farjallah, Jean-Marc Armani, Marcelino Seif, Luigi Dilillo

To cite this version:

Valentin Gherman, Emna Farjallah, Jean-Marc Armani, Marcelino Seif, Luigi Dilillo. Improvement of the tolerated raw bit-error rate in NAND Flash-based SSDs with the help of embedded statistics. ITC 2017 - 48th International Test Conference, Oct 2017, Fort Worth, United States. 10.1109/TEST.2017.8242066 . lirmm-01582185

HAL Id: lirmm-01582185

https://hal-lirmm.ccsd.cnrs.fr/lirmm-01582185v1

Submitted on 5 Jan 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with the Help of Embedded Statistics

Valentin Gherman, Emna Farjallah,
Jean-Marc Armani, Marcelino Seif
CEA, LIST,
Laboratoire Fiabilité et Intégration Capteurs
PC 172, 91191 Gif sur Yvette, France

Luigi Dilillo
LIRMM
UMR 5506, 161 rue Ada
34095 Montpellier Cedex 5, France

***Abstract*—Solid-state drives (SSDs) based on NAND flash memories provide an attractive storage solution as they are faster and less power hungry than traditional hard-disc drives (HDDs). Aggressive storage density improvements in flash memories enabled reductions of the cost per gigabit but also caused reliability degradations. A recent large-scale study revealed that the uncorrectable bit error rates (UBER) in data center SSDs may fall far below the JEDEC standard recommendations. Here, a technique is proposed to improve the tolerated raw bit error rate (RBER) based on the observation that (a) a small SSD ratio may have a much higher RBER than the rest and (b) the RBER is dominated by the retention error rate. Instead of employing stronger but costly error-correcting codes a statistical approach is used to estimate the remaining retention time, i.e., the reliable data storage time, of flash memory pages. This estimation can be performed each time a memory page is read based on the number of detected retention errors and the elapsed time since data was programmed. The fact that the estimated remaining retention time is smaller than a maximum time interval before the next read operation is an indication that data needs to be refreshed. It is estimated that the tolerated RBER can be increased by more than a decade over a storage period of 3 years if the stored data are verified on a monthly basis and refreshed only if necessary. The proposed technique has the ability to adapt the average time between refresh operations to the actual RBER. This enables performance overhead reductions with factors between 8x and 12x as compared to systematic refresh schemes.**

Keywords—NAND flash; SSD; reliability; adaptability; data retention; bit error rate; embedded statistics

I. Introduction

Solid-state drives (SSDs) based on NAND flash memories offer a low power and high performance storage alternative to traditional hard-disc drives (HDDs) [13]. The continuous technology scaling and emergence of flash memories with multilevel cells (MLC) brought not only cost per gigabit reductions but also reliability degradations. For instance, the *cycling endurance* of a flash memory, i.e., the cumulative number of program/erase (P/E) cycles that can be sustained by a memory cell, is decreased by an order of magnitude each time the cell storage capacity is enhanced with an additional bit [5][13][20]. What is more, a recent large-scale study revealed that the uncorrectable bit error rate (UBER) of data center SSDs can significantly exceed the JEDEC standard recommendations. The reported UBER values are between 10^{-11} and 10^{-9} [12] while client and enterprise class SSDs are required to provide an UBER below 10^{-15} and 10^{-16}, respectively [9].

An efficient approach to improve UBER is to use stronger error-correcting codes (ECCs). Unfortunately, powerful ECCs come with important storage and latency overheads. For instance, the storage overhead of a BCH code increases almost linearly with the number of correctable errors and also with the memory page size for a given ratio of bits with correctable errors [7].

The need for strong ECCs may be reduced by containing the raw bit error rate (RBER). Besides technological fixes or solutions based on improved read and write algorithms [1][4][13], the RBER can be tempered if the stored data are periodically refreshed [2][13][14][17]. A refresh operation can be executed in-place by injecting only the missing amount of charge into the floating gates of the flash memory cells or by relocating the data to a different physical location [2][3]. Relocation operations may result in significant P/E cycle overhead especially in the case of read-intensive applications for which the data relocation frequency may become larger than the functional update rate [2]. A way to reduce this overhead is to adapt the relocation rate to the number of P/E cycles endured by each flash memory block [2][3].

One limitation of such refresh schemes is that they are based on worst-case scenarios, oblivious to intra- and inter-device variations, which may lead to unnecessary overheads with respect to response latency, dissipated power and P/E cycles. For example, the large-scale study reported in [12] unveiled that only a small number of SSDs may contribute to the overall UBER degradation.

Here, a statistical approach is proposed to avoid the utilization of strong ECCs or worst-case refresh frequencies for dealing with a whole population of NAND flash memories or

SSDs that may contain some error-prone units. The idea is to exploit the fact that the retention error rate dominates the RBER in NAND flash memories [3] and take advantage of the read operations of each flash memory page to estimate its remaining reliable data storage time, i.e., *retention time*. Such an estimation can be done based on the detected number of retention errors and the *retention age*, i.e., the elapsed time since data was programmed. A valid memory page should be refreshed when the estimated remaining retention time is smaller than the timespan to the next read operation.

Such a technique is effective when a maximum time interval is imposed between consecutive read operations of any memory page. For example, the tolerated RBER can be increased by up to 28× over a storage period of 3 years if one makes sure that the stored data are read at least once in a month. The resulting data refresh frequency is not necessarily correlated to the frequency of data read operations since it depends on the actual RBER via the estimated remaining retention time. It is shown that the refresh probability is negligible at RBERs that can be managed by the available ECC alone and starts to increase only at larger RBERs. Compared to systematic refresh schemes able to ensure the same protection level, performance overhead reductions between 8× and 12× have been simulated.

Types of storage errors that may affect NAND flash memories are analyzed in Section II. The proposed refresh scheme based on the estimation of the remaining retention time of flash memory pages is presented in Section III. Simulation results concerning the improvement of the tolerated RBER and the reduction of the refresh frequency are reported in Section IV. A parallel with a state-of-the-art scheme that also relies on the estimation of the remaining retention time is made in Section V. Concluding remarks are drawn in Section VI.

II. Typical Storage Errors in NAND Flash Memories

A flash memory cell consists of a MOS transistor with a floating gate or a charge trap layer embedded in the dielectric between channel and control gate. Data are programmed via the injection/erasure of electric charge into/from the floating gate or the charge trap layer. The threshold voltage distribution created by the injected charge into the floating gate of an MLC flash memory is illustrated in Fig. 1 [13]. In a NAND flash memory, between 32 and 64 memory cells are connected together to form a string. Thousands of strings are assembled in a storage array called block and few thousands of blocks may be contained in a flash memory chip. In a block, memory cells on the same string are accessed with the help of different word lines. The bits stored in memory cells accessed by the same word line are logically grouped into one or several pages.

A NAND flash memory can be affected by different types of storage errors like retention errors, write errors, also called program-interference or over-programming errors, read-disturb errors and erase errors. Retention errors affect the ability of a memory to keep the stored information over a required period of time. As shown in Fig. 2, retention errors appear due to a drift to the left of the threshold voltage distribution and the resulting crossing of the reference values used during read operations. For retention ages larger than one month, the retention error rate largely dominates the other error rates [3].

The remaining error types are characterised by a drift to the right of the threshold voltage distribution. Write errors are induced by parasitic capacitance-coupling affecting memory cells on a certain word line subsequent to a program operation on a neighbour word line. Once a memory block is fully programmed, the number of write errors does not increase with the retention age. In NAND flash memories, the write errors have the second largest occurrence rate [3].

Erase errors are the outcome of an erase operation that fails to reset all cells in a memory block to the erased state [3]. Upon the occurrence of an erase error an entire block may be marked as bad and discarded [13]. In the following, we will assume that programmed memory pages are not affected by erase errors.

A read-disturb error occurs when the content of a memory cell is corrupted due to repeated read operations of cells on the same string. In the following, read-disturb errors will be neglected due to their very small rate [3].

Retention errors will be considered as the only errors whose rate may increase with the retention age once a memory bloc has been programmed. It will be assumed that the retention RBER ($RBER_{RET}$), i.e., the probability that a vulnerable bit is affected by a retention error, is given by the following expression:

$$RBER_{RET}(t_{AGE}) = 1 - e^{-\lambda t_{AGE}} \quad (1)$$

where t_{AGE} is the retention age. The parameter λ may vary from one SDD or flash memory to another [12][15], between the pages of the same memory block [3] and with the number of P/E cycles endured by a memory block [2][3]. This law is in agreement with the results reported in [15] and it has the properties of a cumulative distribution function, i.e., $RBER_{RET}(0) = 0$ and $RBER_{RET}(\infty) = 1$.

Retention errors can be easily distinguished from other error types. Typically, each read operation is followed by an error correction step during which the erroneous bits are identified with the help of an ECC. The error-correcting process allows to infer the polarity of each error, i.e., the difference

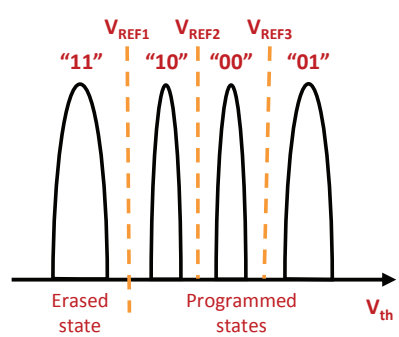

Fig. 1 Threshold voltage distribution and example of logical state encoding for a 2-bit MLC flash memory.

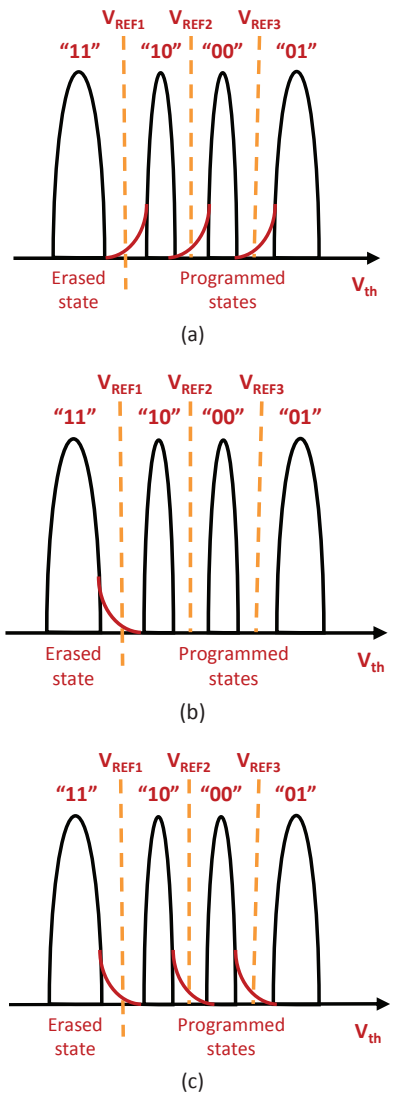

Fig. 2 Threshold voltage distribution of the logical states in a 2-bit MLC flash memory affected by (a) retention errors, (b) erase errors and (c) write and read-disturb errors. There is always at least 1 bit value which is not vulnerable to an arbitrary error type.

between the corrected and the initial values of the erroneous bit. The error polarity allows to identify the error type if one assumes that errors can only result from threshold voltage transitions between neighbouring states. Table I gives the retention error fingerprints for the MLC considered in Fig. 2. For flash memories with 1 bit per cell (SLC), the retention error fingerprint is given by the first line in Table I.

TABLE I. RETENTION ERROR FINGERPRINTS FOR THE MLC FLASH MEMORY CELL CONSIDERED IN FIG. 2.

Bits in the same cell	Read value	Corrected value	Value of the companion bit
First bit	1	0	-
Second bit	1	0	1
Second bit	0	1	0

III. EMBEDDED STATISTICS TO IMPROVE THE TOLERATED RBER IN NAND FLASH MEMORIES

Here, an approach is proposed to deal with RBER variations beyond the error protection provided by the ECC of a NAND flash memory. The main idea is to take advantage of each read operation of a flash memory page to estimate the parameter λ in (1) and the remaining retention time τ of the stored data. The remaining retention time τ refers to the storage time that is still left before the UBER target is exceeded [9]. Assuming a maximum time period T_{READ} between two consecutive read operations of any page, the read data has to be refreshed if $\tau < T_{READ}$. The necessary check operations associated to a read operation are formalized below.

Algorithm 1: Oracle based on embedded statistics

Require: An accessed flash memory page protected by an ECC with known error correction strength

Require: N_{VUL}, the initial number of bits vulnerable to retention errors

Require: ε_{RET}, the number of already existing retention errors

Require: $\varepsilon_{\neg RET}$, the number of already existing non-retention errors

Require: T_{READ}, the maximum time interval before the next read operation

1 Calculate the retention age t_{AGE} of the accessed page

2 Get the remaining retention time τ as a function of t_{AGE}, N_{VUL}, ε_{RET} and $\varepsilon_{\neg RET}$ such that the UBER target is still preserved

3 **if** $\tau(t_{AGE}, N_{VUL}, \varepsilon_{RET}, \varepsilon_{\neg RET}) < T_{READ}$ **then**

4 Refresh the accessed page

5 **end**

The parameter ε_{RET} can be calculated with the help of the ECC decoder. For example, the decoding scheme of a BCH ECC is usually concluded by the execution of a so-called *Chien algorithm* that looks for a potential error in each bit position [7]. When an error location is found, a simple check of the conditions in Table I allows to identify a retention error and increment a retention error counter. Similarly, one can count the number of non-retention errors $\varepsilon_{\neg RET}$.

A maximum time interval T_{READ} between consecutive read operations can be imposed via periodic scrubbing [16]. As it will be seen later, a relatively large T_{READ}, i.e., few months, allows to significantly improve the tolerated RBER. It has been reported that the average response time of an SSD is insignificantly degraded if the refresh operations can be interrupted with sufficiently high granularity [17]. Here, the performance overhead of the scrubbing operations is expected to be even lower since not all read/check operations have to be followed by a refresh operation. In order to avoid redundant verifications of frequently accessed pages, one bit of metadata may be reserved for each page to indicate whether the page has already been read/checked due to a functional request during the current scrubbing period.

The retention age t_{AGE} can be calculated as the difference between a timestamp associated to the page being accessed and the current state of the timer used to provide timestamps [6]. A single timestamp may be used to characterize the programing time of all pages in a flash memory block [13]. The resulting storage overhead of the timestamp table is significantly smaller than in the case of other metadata structures such as the remapping table of the flash translation layer (FTL) [19].

The remaining retention time τ used in Algorithm I can be computed off-line for all possible parameter combinations. This is achieved via a statistical estimation of the parameter λ as explained in Annex I. The estimated λ allows to predict $RBER_{RET}$ at the end of any future storage period t and τ can be selected as the maximum value of t for which $UBER(t)$ still complies with the JEDEC standard [9]. Based on the assumption that only the retention errors may accumulate in time, $UBER(t)$ can be computed with the expression below [15].

$$UBER(t) = \frac{1}{N} \sum_{i=M-\varepsilon_{RET}-\varepsilon_{\neg RET}+1}^{N_{VUL}-\varepsilon_{RET}} \binom{N_{VUL}-\varepsilon_{RET}}{i} \left(RBER_{RET}(t)\right)^{i} \times \left(1-RBER_{RET}(t)\right)^{N_{VUL}-\varepsilon_{RET}-i} \quad (2)$$

where:

- N is the total number of bits in a flash memory page,
- M is the maximum number of errors that can be corrected with the available ECC,
- N_{VUL} is the initial number of bits vulnerable to retention errors,
- ε_{RET} and $\varepsilon_{\neg RET}$ are the numbers of bits affected by retention and non-retention errors,
- $M-\varepsilon_{\neg RET}-\varepsilon_{RET}$ represents the maximum number of additional errors that can still be corrected with the available ECC,
- $N_{VUL}-\varepsilon_{RET}$ represents the number of bits which are still vulnerable to retention errors.

N and M are a priori known parameters. N_{VUL}-ε_{RET} could be calculated if one saves N_{VUL} as metadata for each flash memory page [11]. Fortunately, this is not necessary as it will be explained in the following.

Fig. 3 illustrates examples of the remaining retention time τ as a function of ε_{RET} at several retention ages t_{AGE} for a flash memory page with 8Kb bits vulnerable to retention errors. The remaining retention time is calculated with a resolution of 1 month and a confidence level (CL) of 90%. The maximum value on the Y-axis corresponds to a target retention time of 3 years [18]. It can be observed that the remaining retention time τ decreases rapidly with the number of retention errors ε_{RET} and increases with the retention age t_{AGE}.

The good news is that τ has a weak monotonic dependence on the number of vulnerable bits N_{VUL} as illustrated in Fig. 4. For example, assuming that T_{READ} is one month, the *if condition* in Algorithm 1 gives different outcomes for $N_{VUL} = 100$ and $N_{VUL} = 16K$ only if ε_{RET} is equal to 26. The monotonic dependence of τ on N_{VUL} is the result of the fact that $UBER$ increases monotonically with $RBER_{RET}$ [13][15] and also with N_{VUL} for a relatively large $RBER_{RET}$. This means that a smaller N_{VUL} provides a margin to increase $RBER_{RET}$ without compromising the upper limit of $UBER$. According to (1), a larger $RBER_{RET}$ enables a larger τ for a given λ. The situation depicted in Fig. 4 corresponds to the maximum considered retention age which gives (a) the largest remaining retention times for a certain ε_{RET} as illustrated in Fig. 3 and, implicitly, (b) the largest differences between remaining retention times for different N_{VUL} values.

Since $\tau(t_{AGE}, N_{VUL}, \varepsilon_{RET}, \varepsilon_{\neg RET})$ decreases very slowly with N_{VUL}, a conservative implementation of Algorithm 1 is to always use the value of τ that corresponds to the maximum possible value of N_{VUL} for any possible combination of the parameters t_{AGE}, ε_{RET} and $\varepsilon_{\neg RET}$. This observation allows to greatly reduce the storage overhead of keeping the values of τ and avoid the overhead of calculating and storing N_{VUL} for each flash memory page.

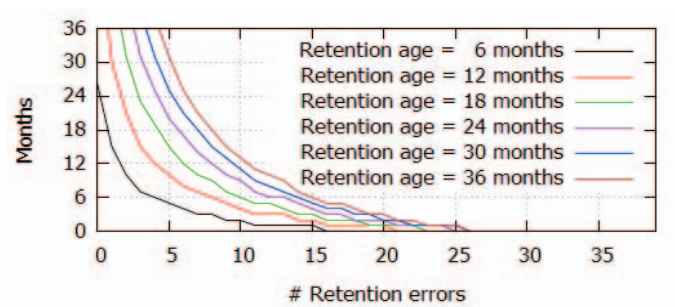

Fig. 3 Estimated remaining retention time as a function of the number of retention errors for several retention ages and a confidence level of 90%. The estimation is made with a granularity of 1 month for a 16Kb flash memory page and 8Kb bits vulnerable to retention errors. The considered ECC is able to correct up to 40 errors per page. The imposed upper UBER limit is 10^{16} [9].

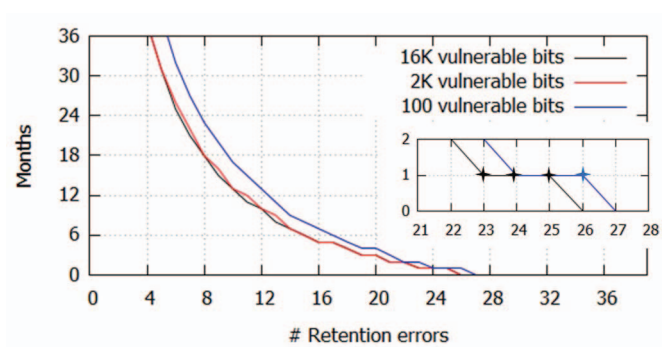

Fig. 4 Estimated remaining retention time at a maximum retention age of 36 months as a function of the number of retention errors and number of vulnerable bits. The other parameters are similar to those considered in Fig. 3. The inset presents a zoom of the region where the remaining retention time becomes 1 month.

The storage overhead can be further reduced by taking advantage of the fact that $\tau(t_{AGE}, \varepsilon_{RET}, \varepsilon_{\neg RET})$ decreases with ε_{RET}. Consequently, one only needs to store the largest ε_{RET} value for which τ is still larger than a preselected T_{READ} value. In such a case, the line 3 in Algorithm 1 may be implemented as the comparison of the maximum tolerated ε_{RET} to the measured ε_{RET}. The resulting storage cost measured in bits can be expressed as follows:

$$storage\ cost = (\varepsilon_{\neg RET_MAX} + 1)\frac{T_{MAX}}{T_{READ}}\lceil log_2\,(M+1)\rceil \quad (3)$$

where:

- T_{MAX} is the maximum required retention time,
- T_{READ} is the maximum time interval between two consecutive read/check operations of any flash memory page,
- M is the maximum number of errors that can be corrected with the available ECC,
- $\varepsilon_{\neg RET_MAX}$ is the maximum allowed number of bits affected by non-retention errors in a flash memory page,
- $\lceil\rceil$ represents the ceiling function.

The value of $\varepsilon_{\neg RET_MAX}$ can be selected to be much smaller than M due to the fact that the number of retention errors at large retention ages is expected to be orders of magnitude greater than the number of non-retention errors [3]. Pages with a number of non-retention errors higher than $\varepsilon_{\neg RET_MAX}$ can be discarded by declaring their block as *bad*. Another possibility is to use these pages to store hot data, i.e., data with a high update frequency and a small storage time in order to reduce the number of potential retention errors.

According to (3), the resulting storage cost amounts to a few hundreds of bits for typical values of the involved parameters. For example, if one considers M=10, $\varepsilon_{\neg RET_MAX}$=1, T_{MAX} equal to 3 years and T_{READ} equal to 1 month, the resulting storage overhead is 288 bits. This is negligible compared to the overhead of the FTL remapping table whose size is measured in megabytes [19].

Observation

Since the statistical method proposed in Annex I is rather pessimistic, the retention RBER values estimated for storage periods of at least 1 month are rather important. Consequently, over such periods, ECCs with a relatively low error-correcting strength are unable to ensure JEDEC compliant UBER values [9]. In order to prevent systematical refreshed operations after each T_{READ} period, we imposed that, in the absence of retention errors, the estimated remaining retention time τ is at least equal to the retention age. The same constraint was imposed in the case when the number of retention errors is equal to 1 and the available ECC is able to correct at least 10 errors per page.

IV. Simulation Results

In order to assess the effectiveness of the proposed statistical approach we first evaluated the extent to which the retention error rate $RBER_{RET}$ may be increased with respect to data that is not refreshed without compromising the recommended upper limit of UBER. The obtained results are reported in Table II for ECCs able to correct up to 40 random single-bit errors per flash memory page. It was considered that each page could be affected by at most one non-retention error ($\varepsilon_{\neg RET} = 1$). The tolerated $RBER_{RET}$ values are compliant with the requirement to keep UBER below 10^{-16} [9]. UBER calculation details are given in Annex II.

As illustrated in Table II, the tolerated $RBER_{RET}$ is increased by (a) the reduction of T_{READ}, i.e., the maximum time interval between two successive read/check operations of any flash memory page, and (b) the reduction of the number of bits vulnerable to retention errors N_{VUL}. The latter impact can be explained by the fact that $UBER$ increases with both $RBER_{RET}$ and N_{VUL} which means that a smaller N_{VUL} value makes room to increase the maximum tolerated $RBER_{RET}$.

When T_{READ} is equal to 1 month, the obtained improvement factors are between 19× and 28×. Better results should be expected for lower T_{READ} values. Moreover, the maximum tolerated $RBER_{RET}$ values and resulting improvement factors are pessimistic as check operations triggered by functional read operations are not taken into account.

With an ECC able to correct up to 40 errors per page, the improvement factor grows with the number of vulnerable bits N_{VUL}. For the other considered ECC strengths, the number of vulnerable bits does not have a sensible influence on the improvement factor. This may be explained by the fact that the maximum tolerated $RBER_{RET}$ is increased by the reduction of N_{VUL} also in the absence of data refresh operations.

A T_{READ} equal to 1 month may have a larger impact on the tolerated $RBER_{RET}$ than the triplication of the number of correctable errors. This can be observed by comparing the *1 month* line for 10 correctable errors with the *no check* line for 30 correctable errors. A T_{READ} equal to 6 months allows to tolerate a larger $RBER_{RET}$ than a duplication of the number of correctable errors. This can be noticed by comparing the *6 months* line for 20 correctable errors with the *no check* line for 40 correctable errors. For BCH codes, the extension of the number of correctable errors from 20 to 40 may augment the storage overhead by 75%. This impact becomes 180% when the number of correctable errors is changed from 10 to 30.

The impact of the page size is not considered since, according to (2), it affects UBER only as a scaling factor. The page size may influence the maximum number of non-retention errors that has to be considered.

The results reported here are calculated by neglecting system downtimes and they can only be expected for systems with power outages much shorter than the imposed check period T_{READ}. This is true for enterprise class SSDs with limited outage periods, i.e., maximum few hours per year [20]. When

longer outage periods are expected, one has to consider the maximum downtime $T_{POWER-OFF}$ and modify the *if condition* in Algorithm 1 such that a data refresh operation is initiated if $\tau < T_{READ} + T_{POWER-OFF}$. The worst-case scenario happens when the time period between the scrubbing campaigns becomes $T_{READ} + T_{POWER-OFF}$ instead of T_{READ}. In such a case, the benefit brought by the proposed approach can be calculated by considering an imposed check period equal to $T_{READ} + T_{POWER-OFF}$. For instance, if both T_{READ} and $T_{POWER-OFF}$ are 1 month, the improvements obtained with our approach are given by the lines with the label *2 months* in Table II. The case when $T_{POWER-OFF}$ is equal to 3 months corresponds to a JEDEC requirement for enterprise class SSDs [9]. In Table II, the *4 months* lines give worst-case improvement factors between 4.9× and 7.5× for a JEDEC compliant enterprise class SSDs in which data are checked every month.

The reported improvements of the tolerated $RBER_{RET}$ require a number of refresh operations that may be greatly reduced as compared to a conventional scheme with systematic refresh operations [2][3]. Fig. 5 shows the probability of refresh operations induced by the execution of Algorithm 1 over a storage period of 3 years. The refresh probability starts to rise near the largest $RBER_{RET}$ value that can be managed by the available ECC without refresh operations and continues to progressively increase with $RBER_{RET}$. The smallest check period provides the lowest refresh probability as the stored data should survive a shorter period of time before the next check operation. This proves the inherent ability of the proposed approach to adapt its refresh frequency to the actual $RBER_{RET}$. Due to this adaptability, the impact of the P/E cycles on $RBER_{RET}$ does not have to be explicitly taken into account as opposed to a systematic refresh scheme [2][3].

TABLE II. IMPROVEMENT OF THE MAXIMUM RETENTION RBER THAT CAN BE TOLERATED IN A 16KB FLASH MEMORY PAGE. THE REPORTED RBER FIGURES CORRESPOND TO A TARGET RETENTION TIME OF 36 MONTHS. THE REMAINIG RETENTION TIME τ USED IN ALGORITHM 1 WAS CALCULATED WITH A CONFIDENCE LEVEL OF 90%.

Number of correctable errors	Check period (T_{READ})	Maximum tolerated $RBER_{RET}$ with UBER ≤ 10^{-16}					Improvement factor with respect to *no refresh*				
		N_{VUL}=16Kb	N_{VUL}=8Kb	N_{VUL}=4Kb	N_{VUL}=2Kb	N_{VUL}=1Kb	N_{VUL}=16Kb	N_{VUL}=8Kb	N_{VUL}=4Kb	N_{VUL}=2Kb	N_{VUL}=1Kb
40	1 month	1.53×10^{-2}	2.56×10^{-2}	5.10×10^{-2}	9.83×10^{-2}	1.87×10^{-1}	24.4	20.3	20.2	19.5	18.3
	2 months	7.70×10^{-3}	1.29×10^{-2}	2.58×10^{-2}	5.04×10^{-2}	9.81×10^{-2}	12.3	10.2	10.2	10.0	9.6
	3 months	5.14×10^{-3}	8.61×10^{-3}	1.73×10^{-2}	3.39×10^{-2}	6.65×10^{-2}	8.2	6.8	6.9	6.7	6.5
	4 months	3.86×10^{-3}	6.47×10^{-3}	1.30×10^{-2}	2.56×10^{-2}	5.04×10^{-2}	6.1	5.1	5.2	5.1	4.9
	6 months	2.61×10^{-3}	4.42×10^{-3}	8.87×10^{-3}	1.74×10^{-2}	3.54×10^{-2}	4.2	3.5	3.5	3.4	3.5
	no check	6.28×10^{-4}	1.26×10^{-3}	2.52×10^{-3}	5.05×10^{-3}	1.02×10^{-2}	-	-	-	-	-
30	1 month	9.69×10^{-3}	1.94×10^{-2}	3.87×10^{-2}	7.71×10^{-2}	1.52×10^{-1}	26.9	26.9	26.9	26.7	26.1
	2 months	4.86×10^{-3}	9.73×10^{-3}	1.95×10^{-2}	3.93×10^{-2}	7.93×10^{-2}	13.5	13.5	13.5	13.6	13.6
	3 months	3.24×10^{-3}	6.50×10^{-3}	1.31×10^{-2}	2.64×10^{-2}	5.36×10^{-2}	9.0	9.0	9.1	9.1	9.1
	4 months	2.44×10^{-3}	4.89×10^{-3}	9.83×10^{-3}	1.99×10^{-2}	4.05×10^{-2}	6.8	6.8	6.8	6.9	7.0
	6 months	1.71×10^{-3}	3.43×10^{-3}	6.87×10^{-3}	1.38×10^{-2}	2.80×10^{-2}	4.8	4.8	4.8	4.8	4.8
	no check	3.60×10^{-4}	7.20×10^{-4}	1.44×10^{-3}	2.89×10^{-3}	5.82×10^{-3}	-	-	-	-	-
20	1 month	4.10×10^{-3}	8.20×10^{-3}	1.64×10^{-2}	3.28×10^{-2}	6.54×10^{-2}	28.1	28.0	27.8	27.8	27.7
	2 months	2.05×10^{-3}	4.11×10^{-3}	8.23×10^{-3}	1.65×10^{-2}	3.32×10^{-2}	14.0	14.0	14.0	14.0	14.1
	3 months	1.40×10^{-3}	2.80×10^{-3}	5.60×10^{-3}	1.12×10^{-2}	2.26×10^{-2}	9.6	9.6	9.6	9.5	9.6
	4 months	1.09×10^{-3}	2.17×10^{-3}	4.35×10^{-3}	8.72×10^{-3}	1.75×10^{-2}	7.5	7.4	7.4	7.4	7.4
	6 months	7.83×10^{-4}	1.57×10^{-3}	3.14×10^{-3}	6.29×10^{-3}	1.26×10^{-2}	5.4	5.4	5.4	5.3	5.3
	no check	1.46×10^{-4}	2.93×10^{-4}	5.86×10^{-4}	1.18×10^{-3}	2.36×10^{-3}	-	-	-	-	-
10	1 month	3.73×10^{-4}	7.47×10^{-4}	1.49×10^{-3}	2.99×10^{-3}	5.99×10^{-3}	19.7	19.8	19.7	19.8	19.8
	2 months	1.91×10^{-4}	3.83×10^{-4}	7.66×10^{-4}	1.53×10^{-3}	3.07×10^{-3}	10.1	10.2	10.1	10.1	10.1
	3 months	1.32×10^{-4}	2.64×10^{-4}	5.29×10^{-4}	1.06×10^{-3}	2.12×10^{-3}	7.0	7.0	7.0	7.0	7.0
	4 months	1.02×10^{-4}	2.05×10^{-4}	4.10×10^{-4}	8.21×10^{-4}	1.64×10^{-3}	5.4	5.4	5.4	5.4	5.4
	6 months	7.23×10^{-5}	1.45×10^{-4}	2.90×10^{-4}	5.80×10^{-4}	1.16×10^{-3}	3.8	3.8	3.8	3.8	3.8
	no check	1.89×10^{-5}	3.77×10^{-5}	7.55×10^{-5}	1.51×10^{-4}	3.03×10^{-4}	-	-	-	-	-

As illustrated in Fig. 6, with the proposed scheme the average time between refresh operations may become significantly larger than the ideal refresh period of a systematic refresh scheme. When the read/check period is 1 month, the number of refresh operations may be reduced up to 3× as compared to the case when data are systematically refreshed, not to speak of the difficulty to infer the ideal refresh frequency at run-time.

The augmentation of the average time between refresh operations enables a reduction of the performance overhead as compared to a conventional scheme with fixed refresh frequency. Assuming a low variation rate of the amount of data that may need refresh operations, i.e. static data characterized by a low functional update frequency, the performance overhead reduction can be calculated as follows:

$$\frac{(\tau_{page_write}+\tau_{page_read})f_{fixed_refresh}}{\tau_{page_write}f_{ES_refresh}+\tau_{page_read}f_{ES_check}} \quad (4)$$

where τ_{page_write} and τ_{page_read} stand for the latencies of page write and page read operations, $f_{fixed_refresh}$ represents the fixed refresh frequency of a systematic refresh scheme and f_{ES_check} and $f_{ES_refresh}$ are the frequencies of the check and refresh operations with the proposed method based on embedded statistics (ES). The execution latency of the check operations in Algorithm 1 is considered negligible with respect toτ_{page_write} and τ_{page_read}.

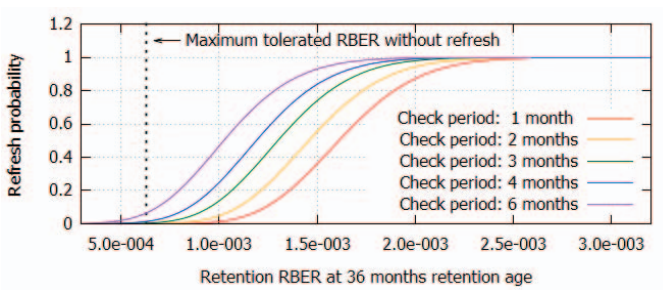

Fig. 5 Refresh probability with the proposed scheme over a target storage period of 3 years. We considered flash memory pages with 16Kb bits vulnerable to retention errors and up to 40 correctable errors.

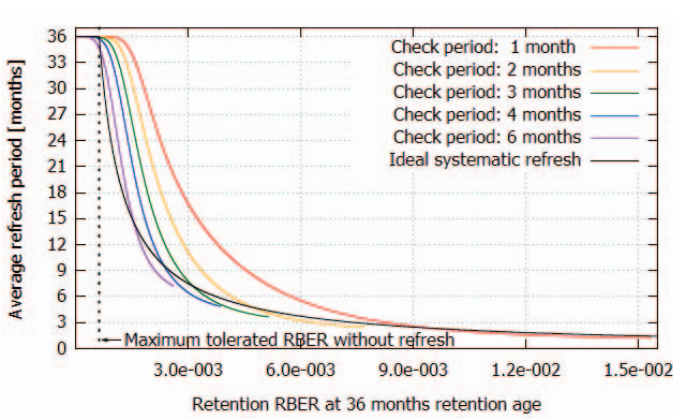

Fig. 6 Average time between refresh operations with the proposed scheme and a systematic refresh scheme with the refresh frequency ideally adapted to the actual RBER. The considered parameters are similar to those considered in Fig. 5. The colored curves stop at the maximum RBER that can be tolerated.

Fig. 7 to Fig. 10 illustrate the reduction of the performance overhead for an MLC NAND flash memory with (i) $\tau_{page_write} = 1500\mu s$ and $\tau_{page_read} = 60\mu s$ [10] and (ii) a $f_{fixed_refresh}$ that allows to tolerate the maximum retention RBER that can be guaranteed with the proposed scheme. Maximum overhead reductions between 8.5× and 12× are obtained for retention RBER values that can be handled by the available ECC alone. Even better results can be obtained for larger $\tau_{page_write}/\tau_{page_read}$ ratios. On the other hand, the performance overhead may become larger than in systematic refresh schemes near the maximum tolerated retention RBER as $f_{ES_refresh}$ approaches $f_{fixed_refresh}$ and fails to compensate for the fact that f_{ES_check} is larger than $f_{fixed_refresh}$.

When up to 10 errors per page can be corrected, the performance overhead reductions are smaller and the curve shapes are different as illustrated in Fig. 10. At this error correction strength, refresh operations are almost systematically required due to the pessimistic tendency of the statistical approach in Annex I. Only some refresh operations are avoided due to the heuristics explained in the observation at the end of Section III. This is also the reason for the smaller improvement factors reported in Table II when the number of correctable errors per flash memory page is equal to 10.

For flash memories and SSDs populations where only a small number of units are error-prone [12], the average performance overhead reduction is expected to approach the values near the vertical dashed lines in Fig. 7 to Fig. 10.

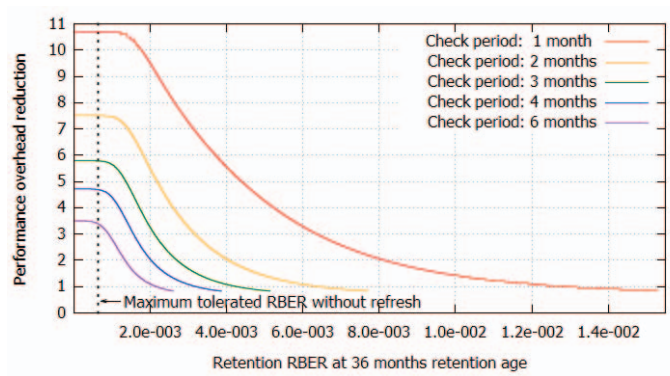

Fig. 7 Performance overhead reduction with respect to a systematic refresh scheme with fixed refresh frequency that can tolerate the same maximum RBER. Similar parameters are considered as in Fig. 5. Each curve stops at the maximum tolerated RBER.

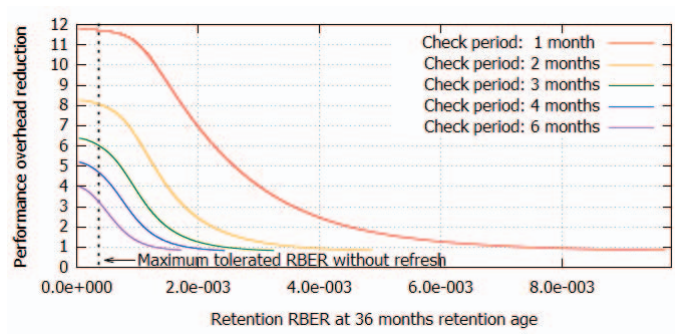

Fig. 8 Similar to Fig. 7 but for an ECC able to correct up to 30 errors per flash memory page.

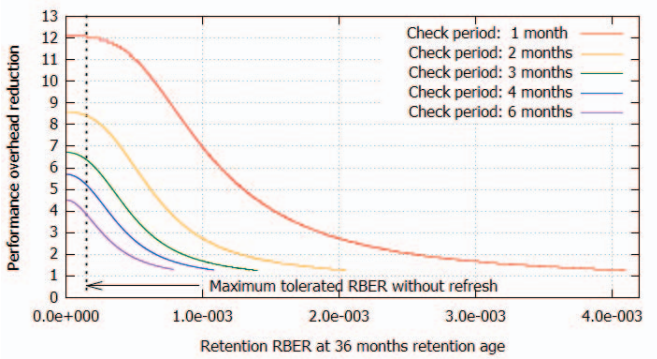

Fig. 9 Similar to Fig. 7 but for an ECC able to correct up to 20 errors per flash memory page.

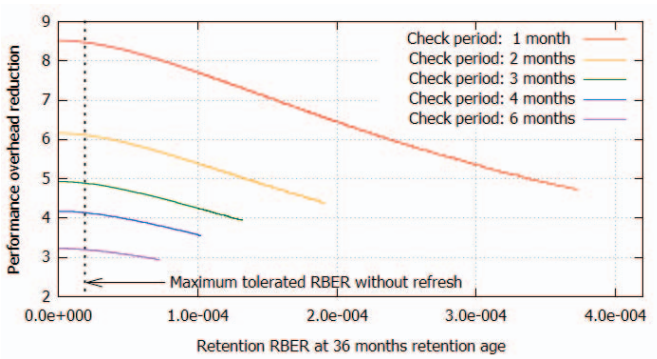

Fig. 10 Similar to Fig. 7 but for an ECC able to correct up to 10 errors per flash memory page.

V. Comparison to State-of-the-Art

A similar approach was presented in [6]. The purpose was to improve the response latency of NAND flash memories by adapting the ECC strength to the actual RBER experienced by each memory page. The ECC strength is selected based on the number of errors occurred during a certain time window. The number of retention errors is estimated based on a precomputed RBER and subtracted from the actual number of errors to infer the number of write errors. The parameters of the RBER model are not adapted to the actual error rates measured in field. During the page read operations no distinction is made between the different error types. Consequently, the precomputed RBER model needs to take into account the number of endured P/E cycles. This is not the case with our solution which only assumes that the number of endured P/E cycles affects the parameter λ of the $RBER_{RET}$ distribution but not the distribution law. Our method is able to adapt to the variations of the parameter λ.

The solution proposed here is orthogonal to other techniques used to reduce the number of retention errors based on the utilization of read reference voltages which are aware of (a) the data retention age [4] or (b) the number of bits vulnerable to retention errors [11].

VI. Conclusions

A statistical approach was proposed to improve the tolerated raw bit error rate (RBER) in NAND flash-based SSDs via an estimation of the remaining retention time. This estimation can be performed each time a flash memory page is read and relies on the number of detected retention errors and the calculated retention age, i.e., the elapsed time since data was programmed. The checked data should be refreshed if the estimated remaining retention time is smaller than a maximum timespan to the next read operation. It was calculated that the tolerated retention error rate can be increased by up to 28× if data are checked on a monthly basis during a storage period of 3 years. Such an improvement may be larger than what can be obtained by triplicating the strength of the available ECC. Even for a check period of 6 months the tolerated RBER improvement can be larger than when the ECC strength is doubled. The proposed method has the ability to adapt the average time between refresh operations to the actual retention RBER. This enabled performance overhead reductions of up to 12× as compared to systematic refresh schemes.

Annex I

This Annex presents a possible way to estimate the parameter λ of the $RBER_{RET}$ model expressed in (1). Based on (1), the probability to have ε_{RET} retention errors in a flash memory page with N_{VUL} vulnerable bits after a storage period t_{AGE} is given by the following binomial law:

$$P(t_{AGE}, N_{VUL}, \varepsilon_{RET}, \lambda) = \binom{N_{VUL}}{\varepsilon_{RET}} e^{-\lambda t_{AGE}(N_{VUL}-\varepsilon_{RET})} \times \left(1 - e^{-\lambda t_{AGE}}\right)^{\varepsilon_{RET}} \quad (5)$$

The relation above can be seen as a discrete probability distribution of ε_{RET} and also as a continuous probability distribution of λ if the following normalization factor is used:

$$\int_0^\infty P(t_{AGE}, N_{VUL}, \varepsilon_{RET}, \lambda) d\lambda = \frac{1}{t_{AGE}(N_{VUL} - \varepsilon_{RET})}$$

The upper bound of λ, i.e. λ_{CL}, can be determined with a certain confidence level (CL) from the following equation:

$$CL = t_{AGE}(N_{VUL} - \varepsilon_{RET}) \int_0^{\lambda_{CL}} P(t_{AGE}, \lambda, N_{VUL}, \varepsilon_{RET}) d\lambda$$

$$CL = (N_{VUL} - \varepsilon_{RET}) \binom{N_{VUL}}{\varepsilon_{RET}} \int_{\zeta_{CL}}^{1} \zeta^{(N_{VUL}-\varepsilon_{RET}-1)} (1-\zeta)^{\varepsilon_{RET}} d\zeta,$$

with $\zeta = e^{-\lambda t_{AGE}}$.

The values of ζ_{CL} and λ_{CL} can be calculated off-line for different combinations of N_{VUL} and ε_{RET}. For ε_{RET} values which are relatively small compared to N_{VUL}, one may use a formalism based on the exponential chi-squared (χ^2) and Poisson distributions. Such a formalism should only be used as an approximation since the error occurrence events do not follow a Poisson process. The approach presented here can be applied to any $RBER_{RET}$ model especially when it depends on a single parameter besides t_{AGE} as is the case with (1).

ANNEX II

This Annex presents a method to compute UBER for flash memories with pages that are periodically checked and may be refreshed according to Algorithm 1. UBER can be obtained by adding the probabilities of the uncorrectable errors that may occur in a flash memory page between consecutive check operations as follows:

$$UBER = \frac{1}{N} \sum_{i=1}^{\left\lceil \frac{T_{MAX}}{T_{READ}} \right\rceil} UBER(i)$$

where:

- N is the total number of bits in a flash memory page,
- T_{MAX} is the maximum required retention time,
- T_{READ} is the time interval between consecutive read/ check operations,
- $UBER(i)$ represents the contribution to UBER of the uncorrectable errors that may occur during the time interval T_{READ} between the *(i-1)*th and *i*th check operations.

UBER(i) can be calculated with the relation below by multiplying (i) the occurrence probability of all retention error numbers ε_{RET} which can be handled by the available ECC and do not impose a page refresh according to Algorithm 1 and (ii) the probability that during the subsequent check operation the errors cannot be corrected anymore:

$$UBER(i) = \sum_{\varepsilon_{RET}=0}^{n_{i-1}} P((i-1) * T_{READ}, N_{VUL}, \varepsilon_{RET}) \times \left[1 - \sum_{\varepsilon'_{RET}=0}^{M-\varepsilon_{\neg RET}-\varepsilon_{RET}} P(T_{READ}, N_{VUL} - \varepsilon_{RET}, \varepsilon'_{RET}) \right] \quad (6)$$

where:

- $P(T_{READ}, N_{VUL}, \varepsilon_{RET})$ is calculated with (5) for a given parameter λ such that $P(0, N_{VUL}, 0) = 1$ and $P(0, N_{VUL}, \varepsilon_{RET}) = 0$ if $\varepsilon_{RET} \neq 0$,
- for *i>1*, $P(i * T_{READ}, N_{VUL}, \varepsilon_{RET})$ can be inferred recursively with (7) as shown below,
- n_{i-1} is the maximum number of retention errors for which the *if condition* in Algorithm 1 is false and no refresh operation needs to executed during the *(i-1)*th check operation,
- M is the maximum number of errors that can be corrected by the available ECC,
- $\varepsilon_{\neg RET}$ is the number of non-retention errors occurred in a flash memory page,
- ε_{RET} and ε'_{RET} are numbers of retention errors that can be tolerated in a flash memory page.

$$P(i * T_{READ}, N_{VUL}, \varepsilon_{RET}) = \sum_{\varepsilon'_{RET}=0}^{\min(\varepsilon_{RET}, n_{i-1})} P((i-1) * T_{READ}, N_{VUL}, \varepsilon'_{RET}) \times P(T_{READ}, N_{VUL} - \varepsilon'_{RET}, \varepsilon_{RET} - \varepsilon'_{RET}) \quad (7)$$

The parameters in (7) have the same meaning as in (5) and (6). Each term in (7) represents the probability of a possible repartition of ε_{RET} retention errors over the time period before the *(i-1)*th check operation and the time interval between the *(i-1)*th and *i*th check operations. Similarly to (5), the presence of the n_{i-1} parameter in the upper summation limit indicates that not all error occurrence scenarios are possible due to a refresh operation that may be triggered during the execution of the *(i-1)*th check operation.

REFRENCES

[1] D. Bertozzi et al., "Performance and reliability analysis of cross-layer optimizations of NAND flash controllers," ACM Transactions on Embedded Computing Systems, vol. 14, no. 1, Article 7, 2015.

[2] Y. Cai et al., "Flash correct-and-refresh: retention-aware error management for increased flash memory lifetime," IEEE International Conference on Computer Design, pp. 94–101, 2012.

[3] Y. Cai et al., "Error analysis and retention-aware error management for nand flash memory," Intel Technology Journal, Volume 17, Issue 1, pp. 140–164, 2013.

[4] Y. Cai et al., "Data retention in MLC NAND flash memory: characterization, optimization, and recovery," International Symposium on High Performance Computer Architecture, pp. 551-563, 2015.

[5] L.-P. Chang, "Hybrid solid-state disks: combining heterogeneous NAND flash in large SSDs," IEEE Asia and South Pacific Design Automation Conference, pp. 428–433, 2008.

[6] S. Di Carlo et al. "FLARES: an aging aware algorithm to autonomously adapt the error correction capability in NAND flash memories," ACM Transactions on Architecture and Code Optimization, vol. 11, issue 3, article no. 26, Oct. 2014.

[7] E. Fujiwara, "Code design for dependable systems: theory and practical applications," Wiley-Interscience, pp. 60, 2006, ISBN: 0471756180.

[8] L.M. Grupp, J.D. Davis, and S. Swanson, "The bleak future of NAND flash memory," File and Storage Technologies, 2012.

[9] JEDEC Standard, "Solid-state drive (SSD) requirements and endurance test method," JESD218A, February 2011.

[10] X. Jimenez, D. Novo, and P. Ienne, "Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance," USENIX Conference on File and Storage Technologies, pp. 47–59, 2014.

[11] C. Lee et al., "A 32Gb MLC NAND-flash memory with V_{th}-endurance-enhancing schemes in 32nm CMOS," ISSCC, pp. 446–448, 2010.

[12] J. Meza et al., "A large-scale study of flash memory failures in the field," ACM SIGMETRICS International Conference on Measurement and Modeling of Computer System, pp. 177–190, 2015.

[13] R. Micheloni, L. Cripa, and A. Marelli, "Inside NAND flash memories," Springer-Verlag, Berlin, 2010.

[14] Micron Technology, "TN-12-30: NOR flash cycling endurance and data retention introduction," Technical Note, 2013.

[15] N. Mielke et al., "Bit error rate in NAND flash memories," IEEE Annual International Reliability Physics Symposium, pp. 9–19, 2008.

[16] S. Mukherjee et al., "Cache scrubbing in microprocessors: myth or necessity?," IEEE Dependable Computing, 2004.

[17] Y. Pan et al., "Quasi-nonvolatile SSD: trading flash memory nonvolatility to improve storage system performance for enterprise applications," High Performance Computer Architecture, pp. 1–10, 2012.

[18] Sandisk, "WP001 - Flash management/A detailed overview of flash management techniques," White Paper, November 2013.

[19] M. Seif et al., "Refresh frequency reduction of data stored in SSDs based on A-timer and timestamps," IEEE European Test Symposium, pp. 1–6, 2017.

[20] N. Sundby and D. Taylor, "Beyond capacity: storage architecture choices for the modern datacenter," White Paper, IDC Analyze the Future, Sponsored by: Toshiba, February 2014.