Approximate Counters for Flash Memory

Jacek Cichoń and Wojciech Macyna
Institute of Mathematics and Computer Science
Wrocław University of Technology, Poland

***Abstract*—**

Flash memory becomes the a very popular storage device. Due to its shock - resistance and power economy is adopted in sensor networks and embedded systems. Recently more attention is paid to the data storage in flash memory. This is not a simple issue due to the limitations of flash memory. Data in flash memory should be distributed evenly among data blocks. If the number of writes in a data block is too high, it may cause damage of the block.

Requirements for highly reliable storage systems include efficient algorithms to maximize its lifetime and tools to predict it or monitor system status. One way to achieve this goal is to embed a system of counters which could control block usage (especially erasing operations). Some solutions of this kind including necessary algorithms are patented.

In this paper we propose an innovative solution involving the use of probabilistic counting of the number of block modifications. Our solution drastically reduces the number of bits needed to memorize counters. We precisely estimate the space requirements for the counter, discuss a series of experiments conforming the correctness of our approach and investigate the evolution system of counters.

***Index Terms*—flash memory, approximate counter**

I. Introduction

Flash memory is a very popular storage device. This is because of its shock - resistance, power economy and non-volatile nature. In recent years flash memory technology takes the lead as a main memory component in sensor networks and embedded systems. There are significant breakthroughs in the capacity and reliability of that technology. The capacity of flash memory increases dramatically with the decreasing cost of such devices. There are flash storage devices of 1 GB on the market and their size will increase. It seems that storage memory will be a good alternative for hard disks in many applications. Nowadays flash memory is used in mobile applications such as PDA devices, cell phones, music players and so on. There are many promising fields, where flash memory may be applied. One of the research directions considers application of flash memory in databases [1]. A lot of papers consider implementation of the B-tree or R-tree indexes over flash memory [2], [3]. There are two types of flash devices: NOR and NAND. The NOR devices have lower storage capacity, but faster and simpler access to stored data, so they can be used for program storage. On the contrary, the NAND devices are able to accumulate more data, because of their storage capacity. They seem to be more suitable for data storage and sensor networks. Flash memory has a completely different architecture from hard disks. The properties on NAND flash are described in [1]. In summary, the read and write operations are performed at the page granularity. One flash memory page has typically 512-2048 bytes. Pages are organized into blocks; one block consists of 32 or 64 pages. The first key feature of such memory type is the way of data erasure. The page can be overwritten only after the erasing of the entire data block, in which the page resides. As a consequence, to perform a delete operation all other pages in the block must be moved to another one. In different scenario, instead of removing the page, it may be checked as ”deleted”. Another important issue is the energy consumption. To write a page the flash memory needs more energy then to perform the read operation. This is important for the system with limited energy supply. In such a case using the energy must be planned optimally. The second feature with the great impact on the data storage and data manipulation in flash memory is the endurance issue. A block may be erased about 100000 to 1000000 times. Thereafter it may be worn out and no more usable. So the write load should be spread over the memory evenly. To do that, each block should be associated with a counter, which can record the number of erase events for this block. Due to these fundamental constraints efficient storage management is a challenging task. The counter of the block erase operation number must be maintained space-efficiently. The naive approach is to store the exact number of the erase events for each block. It's not efficient, because it requires about 21 bits memory per block.

There are several solutions to this problem in the literature. Most of them appeared as US patents. Some approaches are based on the maintaining of the exact counters associated with the erase blocks [4], [5], [6], [7]. Han patented the solution, which relies on wear estimation using erase latencies. It is based on the observation that the erase time increases with wear [8].

In this paper we propose approximate counters for the measurement of the number of the erase operations performed on the particular block in flash memory. Approximate counters may be used in situations when knowledge of a precise number of occurrences of observed phenomenon is not necessary. The main advantage of this solution for our purposes is a significant reduction of the number of bits required for storing the system of counters.

This paper is organized as follows: in the next section we describe the problem and propose our solution. In the third section we show, how to store the data connected with the approximate counters. In the next section we discuss the

evolution of the system of counters and next we describe the experiments, which were done to confirm our method empirically. In the last section we summarized the paper and described conclusions and possible extensions of our work.

II. Problem Formulation

A. Motivation

As we described in the previous section, each block in flash memory should be written with the similar frequency. So every single block must be associated with the counter, which measures the number of erase events. Suppose that the block wears out after 1 million repeated writes. In that case the number of bits used for the maintenance of the single counter is about 21 bits. So if the capacity of the flash is 4 GB, one block consists of 32 pages and every page has 512 bytes, the number of blocks is about 250000. In that case 660000 bytes of the flash memory is needed for the counters. It makes 0.016 percent of the whole memory. Additionally, the frequent increment of the counter is also not desirable, because it needs the modification of the memory block. In this article we propose the probabilistic counter instead of the real counter described above. Our counter needs only 5 bits per block in flash memory. Considering the number of blocks mentioned above, instead of 1000000 bytes (8000000 bits) we need about 1250000 bits. The other advantage of our approach is the update frequency. In the case of the real counter, every single erase of the block causes the increment of the counter. Using our probabilistic counter we don't need to do that so often.

B. Description of the approximate counter

In our work we use the *approximate counting* proposed by Morris [9]. The algorithm makes it possible to keep approximate counts of large numbers in small counters. As we explained before for the real counter, which range from 1 to M we need $\lfloor \log_2 M \rfloor + 1$ bits. So for the counter with the value up to 1 million we need 21 bits. On the contrary the approximate counter needs only approximately $\log_2 \log_2 M$ bits. In that case we need only 5 bits of flash memory for the storage for the same range. So it seems to be much more space efficient than using the real counter.

Now we shall describe the procedure shown in detail in [10]: The approximate counter, called also sometimes called as a probabilistic counter, starts with the counter value C initialized to 0. The process of incrementation of the counter C is described by the following pseudo-code:

d:= Random([0,1]);
if $d < 2^{-C}$ **then**
  C:= C+1
end if

Let C_n denote the value of the counter C after n consecutive increments. Then C_n is a random variable and in [9] it was shown that $\mathrm{E}[2^{C_n}] = n+2$ and $\mathrm{var}[2^{C_n}] = \frac{1}{2}n(n+1)$ (where $\mathrm{E}[X]$ denotes the expected value of a random variable X and $\mathrm{var}[X]$ denotes its variance). Therefore the number $2^C - 2$ is an unbiased estimator of the number of increments n. In a detailed analysis of the approximate counter done in [10] it was shown that

$$\mathrm{E}[C_n] = \lg_2 n + c + \omega(n) + O\left(\frac{1}{n^{0.98}}\right), \qquad (1)$$

where $c \approx -0.273954$ and ω is a periodic function with period 1, of mean value 0 and amplitude less than 10^{-5}. Moreover the variance of the variable C_n is small, namely we have $\mathrm{var}[C_n] \approx 0.8736$ (see also [10]). Hence we see that:

1) the random variable C_n is strongly concentrated,
2) we need approximately $\lg_2 \log_2 n$ bits to store the value of the counter C after n increments,
3) the value 2^{C_n} is a quite precise estimator of the number n of increments.

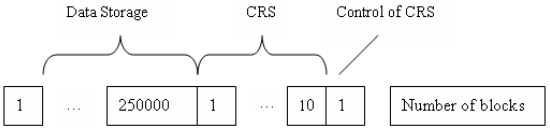

Fig. 1. Storage architecture

III. Storage issues

Lets assume, that one block contains 32 pages and each page 512 bytes. So the number of blocks of the 4GB flash memory is about 250000. As we estimated above, we need 5 bits to store one probabilistic counter. For the flash memory of 4GB it makes about 1250000 bits. The problem appears, how to maintain the counters in the flash memory efficiently. We propose to reserve the adequate number of blocks to the size of flash memory, where the counters will be stored. Due to our previous calculation 1250000 bits (156260 bytes) need about 10 blocks to store. We call these blocks the *counter reservation storage* (CRS). Of course the CRS must also be controlled for the number of erase events. For that purpose about 50 bits (7 bytes) are needed. So for the control of the CRS 1 block of memory is enough. Figure 1 shows the concept of the storage.

It's not difficult to imagine that frequent using of CRS may lead to wearing out of the memory where CRS reside. The number of write and erase events of CRS is very high, because every change in flash memory will cause the change of the associated CRS block. Using our probabilistic counter we scale down the frequency of such operations on CRS. Our counters don't increment as often as it may take place in real counters, so its value does not need to be updated so often (we will come back to this property of CRS in section IV). Of course to protect from wearing out, the CRS may be taken to the main memory and then swap periodically. In one of our experiments below we simulate this process.

IV. Evolution of System of Counters

In this section we discuss the process of evolution of a system of approximate counters. Let us fix a number m of blocks and a system $\vec{C} = (C_i)_{i=1\ldots m}$ of approximate counters describing the number of changes of corresponding blocks. Let us recall that initially we have $C_i = 0$ for each i.

A. Random model

Suppose that we randomly, with the uniform distribution, choose a block to be changed. Then we use the probabilistic increment procedure to the corresponding approximate counter.

Lemma 4.1: Let Change denotes the event that after a modification of a random block the system $\vec{C}$ is changed. Then

$$\Pr[\mathrm{Change}] = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\right)^{C_i} .$$

Proof: Let B_i denotes the event „the ith block is changed". Then

$$\Pr[\mathrm{Change}] = \sum_{i=1}^{m}\Pr[\mathrm{Change}|\mathrm{B}_i]\cdot\Pr[\mathrm{B}_i] = \sum_{i=1}^{m}\Pr[\mathrm{Change}|\mathrm{B}_i]\frac{1}{m} = \frac{1}{m}\sum_{i=1}^{m}\Pr[\mathrm{Change}|\mathrm{B}_i] = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\right)^{C_i} .$$

■

Suppose for a while that we observe a vector $\vec{C}$ after a large number $n \gg m$ of increments. Then each block has been changed approximately $\frac{n}{m}$ times, so for each i we have $C_i \approx \log_2 \frac{n}{m}$, so $2^{-C_i} \approx \frac{m}{n}$, hence $\Pr[\mathrm{Change}] \approx \frac{1}{n}$. So, we see that if n is large then the probability of a change of $\vec{C}$ after an increment operation applied to one counter is very small. Notice that in the case of a vector of real counters increment of each counter changes the counter, so the probability of change is precisely 1. Thus the evolution of a system of approximate counters radically differs from the evolution of real counters: the intensity of changes in the first one decreases quite rapidly, while it is constant in the real case.

In order to transform these intuitions into a precise result we need to prove one auxiliary results. Let us recall that a random variable Y has a Poisson distribution with a parameter λ if $\Pr[Y = k] = e^{-\lambda}\lambda^k/k!$.

Lemma 4.2: Suppose that n balls are drawn uniformly and independently into m urns. Let $X_1, \ldots X_m$ denote the numbers of balls in corresponding urns. Then

$$\Pr[X_1 \le \frac{n}{2m} \vee \ldots \vee X_m \le \frac{n}{2m}] \le 2m\left(\frac{2}{e}\right)^{\frac{n}{2m}} .$$

Proof: Let us consider a sequence $Y_1, \ldots Y_m$ of independent random variables with Poisson distribution with parameter $\lambda = \frac{n}{m}$. Using the classical Chernoff bound for Poisson distribution (see e.g. [11]) we deduce that $\Pr[Y_1 \le \frac{\lambda}{2}] \le \left(\frac{2}{e}\right)^{\frac{\lambda}{2}}$, hence

$$\Pr[Y_1 \le \frac{\lambda}{2} \vee \ldots \vee Y_m \le \frac{\lambda}{2}] \le \sum_{i=1}^{m}\Pr[Y_i \le \frac{\lambda}{2}] \le m\left(\frac{2}{e}\right)^{\frac{\lambda}{2}} .$$

We use now a standard technology (see [11], sec. 5.4 and corollary 5.11) of comparing urn and balls model with the Poisson model we get the required result. ■

Using last lemma we deduce that when a large number n of balls are drawn uniformly and independently into a fixed number m of urns then with high probability all urns are occupied by at least $\frac{1}{2}\frac{n}{m}$ balls. Therefore if we increase n times randomly chosen probabilistic counter then we may expect that with a high probability each counter will have a value at least $\log_2(\frac{n}{2m})$. Indeed, we have the following result:

Theorem 4.3: Let $\mathrm{Change_n}$ denote the event that after n modifications of blocks the next modification will change a counter. If $n \to \infty$ then

$$\Pr[\mathrm{Change_n}] = o(1) .$$

Proof: Let A denotes the event "$X_1 > \frac{n}{2m} \wedge \ldots \wedge X_m > \frac{n}{2m}$". From Lemma 4.2 we get $\Pr[A^c] \le 2m\left(\frac{2}{e}\right)^{\frac{n}{2m}}$. Next we have

$$\Pr[\mathrm{Change_n}] = \Pr[\mathrm{Change_n}|A]\Pr[A] + \Pr[\mathrm{Change_n}|A^c]\Pr[A^c] \le \Pr[\mathrm{Change_n}|A] + 2m\left(\frac{2}{e}\right)^{\frac{n}{2m}} .$$

Hence, we may assume that the event A holds. Let us fix an index i of a counter. Then, according to Eq. (1) for sufficiently large n we have

$$\mathrm{E}[C_i] > \log_2(X_i) - 1 = \log_2(X_i/e) .$$

Moreover, for sufficiently large n we have $\mathrm{var}[C_i] < 1$. From the Chebyshev inequality we deduce that

$$\Pr[C_i > \frac{1}{2}\mathrm{E}[C_i]] \le \frac{4}{\mathrm{E}[C_i]}$$

so under the assumption that the event A holds we get

$$\Pr[C_i > \frac{1}{2}\log_2\frac{n}{2em}] \ge 1 - \frac{4}{\log_2\frac{n}{2em}} .$$

The increments of approximate counters are independent, hence

$$\Pr[\bigwedge_{i=1}^{m}\left(C_i > \frac{1}{2}\log_2\frac{n}{2em}\right)] \ge \left(1 - \frac{4}{\log_2\frac{n}{2em}}\right)^m$$

(on the event A). Observe now that

$$\bigwedge_{i=1}^{m}\left(C_i > \frac{1}{2}\log_2\frac{n}{2em}\right) \to \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\right)^{C_i} < \sqrt{\frac{2me}{n}}$$

Putting all the above pieces together and using Lemma 4.1 we finally get

$$\Pr[\mathrm{Change_n}] \leq \sqrt{\frac{2me}{n}} + \left(1 - \frac{4}{\log_2 \frac{n}{2em}}\right)^m + 2m\left(\frac{2}{e}\right)^{\frac{n}{2m}} ,$$

so the theorem is proved . ■

B. Controlled Use of Blocks

Lets suppose now that each time we need to modify a block we shall use the block with minimal value of the corresponding probabilistic counter. We call this scenario of block using a *controlled use of blocks*.

Notice that if a value of a approximate counter is C then the probability of changing its value after a single call of the increment procedure is 2^{-C}, hence an expected number of calls of the increment procedure needed to change its value is 2^C.

The expected number of steps to change the initial state $(0, 0, \ldots, 0)$ of the counter $\vec{C}$ to $(1, 1, \ldots, 1)$ is m. Next, to change $(1, 1, \ldots, 1)$ into $(2, 2, \ldots, 2)$ we need (in average) 2^1 steps. In general, in order to change $\vec{C}$ from $(a, a, \ldots, a)$ to $(a+1, a+1, \ldots, a+1)$ we need $2^a m$ steps. Using this observations and making some additional calculus we get the following result:

Theorem 4.4: Let L_n denote the number of changes of the system $\vec{C}$ after n updates in the controlled use of blocks. Then

$$\mathrm{E}[L_n] \sim m \log_2(\frac{n}{m} + 1) .$$

Therefore if we use the system of approximate counters in an optimal way n times then there will be only $O(\ln n)$ changes to the system, so only so many times we should flash its contents into flash memory.

V. Experiments

In this section we shall discuss some numerical experiments which have been made with the probabilistic counters in context of flash memory.

A. Comparison of the real and probabilistic counter

In this experiment we recorded randomly k write events among m blocks of flash memory (Fig. 2). For each block we maintain the probabilistic counter (p_i) and the real counter (r_i). Note, that the real counter is recorded only for the need of the experiment and will not be maintained in the real environment. We repeated the experiment n times. For each block we estimate the precision defined by $Prec_i = p_i/r_i$. The average precision for one experiment is: $Prec_j = \frac{1}{m}\sum(Prec_i)$. We also calculate the standard deviation for the experiment defined as $Std = \sqrt{\sum (p_i - Prec)^2}/m$.

We estimated the average precision and standard deviation for 10 experiments and we get $Prec \approx 1.002$ and $Std \approx 0.068$. This experimental result confirms the high precision of the approximate counters.

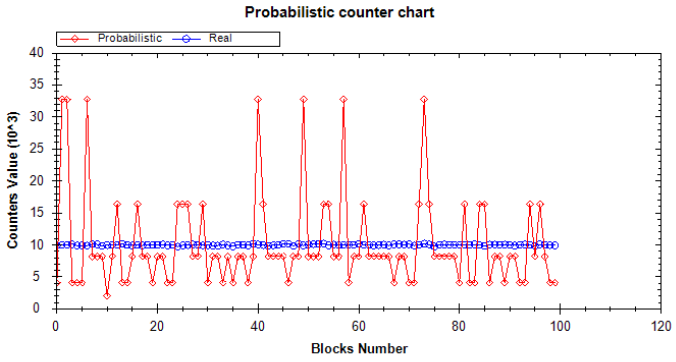

Fig. 2. Typical counter contents after one experiment for m=100 and k=100000

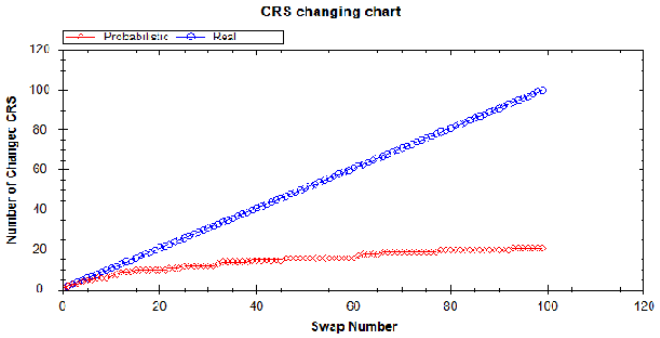

Fig. 3. The chart shows real counter values when the block with minimum approximate counter value is erased. The experiment was made for m=100 and k=100000

B. Data distribution method

In the second experiment we simulated the data distribution in the flash memory blocks. As we mentioned before, the data in the memory blocks should be distributed. In our approach, we use the approximate counter to decide into which memory block to write the data. In our experiment we erase that block, which has the smallest value of the approximate counter. If we erase the block j, we increment the real counter value(r_j) and we use the probabilistic method for the estimation of the approximate counter (p_j). In figure 3 we compared the values of both counters.

C. Swap simulation

In this experiment we simulate the situation where the CRS is loaded into the main memory and then after several operations on counter values swapped back to the blocks of flash memory. We compare the cases when the probabilistic counters and the real counters are stored in the CRS. In our experiment we assume that our CRS consists of 10 blocks and 100 erase operations are performed on the counters of CRS loaded into the main memory. After that the CRS blocks are swapped back to the flash memory. We repeat that procedure 100 times. Figure 4 illustrates our experiment.

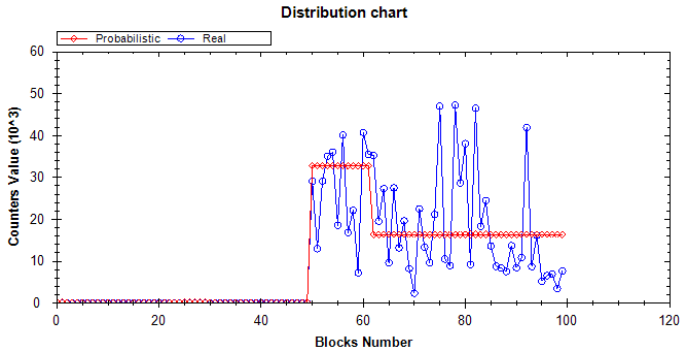

Fig. 4. Typical number of CRS changes where 100 swap operations were performed on 10 CRS blocks and in every swap operation 100 erase events occurs.

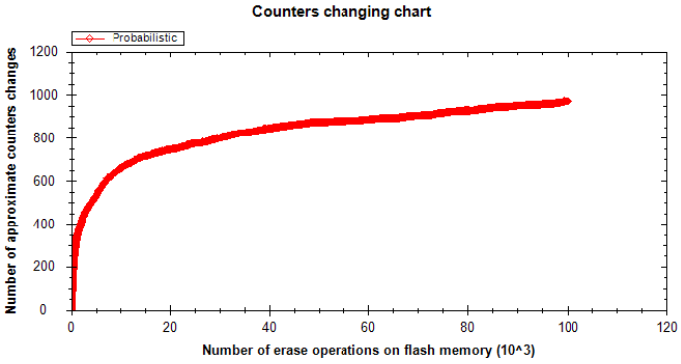

Fig. 5. Global approximate counters changes after erase operations for m=100 and k=100000

D. Counters changing

In this experiment we show the correlation between the number of flash memory changes and the number of counters changes. We changed 100000 times the flash memory consisted of 100 blocks. We see that in the first phase the value of approximate counters increases rapidly and after the certain number of modifications goes up very slowly. Figure 5 illustrates this experiment. A theoretical background of this behavior is explained in sec. IV.

VI. Conclusions and Future Work

In this paper we proposed a system of approximate counters for flash memory devices. In our approach each block of the memory is associated with one counter. The counters are stored in the special area of flash memory called CRS. We showed that the CRS needs only 0.004 percent of the memory. Of course it strongly depends on flash memory architecture. In this paper we discussed statistical properties of the proposed method and we described several experiments which confirm the correctness of our method. The **very low memory requirements** is not the only advantage of our approach. The other one is that the approximate counter values **don't need to be changed so often** as it is in the case of the real counter. It's a very important issue in flash memory environment. In the future investigations we want to combine our approximate counters with the erase reclamation policies.

References

[1] S. Nath and A. Kansal, "Flashdb: dynamic self-tuning database for nand flash," in *IPSN*, T. F. Abdelzaher, L. J. Guibas, and M. Welsh, Eds. ACM, 2007, pp. 410–419.

[2] C.-H. Wu, L.-P. Chang, and T.-W. Kuo, "An efficient r-tree implementation over flash-memory storage systems," in *GIS*. ACM, 2003, pp. 17–24.

[3] C.-H. Wu, T.-W. Kuo, and L.-P. Chang, "An efficient b-tree layer implementation for flash-memory storage systems," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, 2007.

[4] K. M. Lofgren, R. D. Norman, G. B. Thelin, and Gupta, "Wear leveling techniques for flash eeprom systems," *US Patent 6,594,183*, 1998.

[5] ——, "Wear leveling techniques for flash eeprom systems," *US Patent 6,081,447*, 1999.

[6] J. M. Marshall and C. D. H. Manning, "Flash file management system," *US Patent 5,832,493*, 1998.

[7] E. Jou and J. I. J. H., "Flash memory wear leveling system providing immediate direct access to microprocessor," *US Patent 5,568,423*, 1996.

[8] S. W. Han, "Flash memory wear leveling system and method," *US Patent 6,016,275*, 1998.

[9] R. Morris, "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[10] P. Flajolet, "Approximate counting: A detailed analysis," *BIT*, vol. 25, no. 1, pp. 113–134, 1985.

[11] M. Mitzenmacher and E. Upfal, *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, January 2005.