Remove Minimum (RM): An Error-Tolerant Scheme for Cardinality Estimate by HyperLogLog

Pedro Reviriego[1], Jorge Martínez[2] , Ori Rottenstreich[3], Shanshan Liu[4] and Fabrizio Lombardi[4]

Abstract—Estimating the number of distinct elements is required in many computing applications. For example, we may want to estimate the number of unique users accessing a service, or the number of different words in a text. This problem is known as count distinct or cardinality estimate and has been widely studied. An accurate naïve approach is to keep a list of distinct elements and for each upcoming one check such list and add it when it is not found. However, this is not practical for large cardinalities because it would require storing a large number of elements. Over the years, many algorithms have been proposed to provide cardinality estimates without storing elements and using less memory. One of the state-of-the-art algorithms for cardinality estimate is the HyperLogLog (HLL); it provides a good estimate over a large range of cardinality values using a small array of counters. As HLL is implemented in computing systems, it is exposed to soft errors that can corrupt bits stored in memories or registers. To avoid data corruption, memories are commonly protected with Error Correction Codes (ECCs). ECCs however incur in significant overhead because protection needs additional memory cells per word to store the parity check bits as well as additional computation for checking them. For achieving data protection, an alternative approach that can be of low-overhead exploits features of the implemented algorithms to detect and correct errors. In other scenarios, the algorithm must be executed on a system that is not protected with parity or ECCs and thus protection must be implemented at the algorithmic level. In this paper, we first study the impact of soft errors on the HLL algorithm by performing simulation by error injection. The results show that the algorithm is quite robust and can filter out most errors. However, for large cardinalities, there are some errors that can cause a large discrepancy in the HLL estimate. This occurs when the uppermost bit with a value of one in a counter is flipped to a zero. Based on the analysis of the experimental results and the HLL algorithm, a protection technique is proposed that effectively mitigates the impact of soft errors at a small overhead. The proposed Remove Minimum (RM) scheme has been validated by error injection experiments to show that it effectively protects against soft errors on the HLL counters.

Index Terms—Cardinality, Soft Errors, HyperLogLog, Error Tolerance

1 Introduction

Estimating the number of distinct elements in a stream, also known as cardinality or count distinct estimate is a key problem in computer systems. It is needed for example when estimating the number of unique users that access a service, or the number of unique words in a document. With the advent of the big data computing, its importance has considerably increased, because bigger data streams have to be processed and more efficient algorithms are needed [1], [2]. The cardinality estimate is also used in other fields, such as networking, for example to control the spreading of worms or to detect nodes that have a large number of connections that are commonly linked to Distributed Denial of Service (DDoS) attacks [3], [4]. A trivial algorithm to compute the cardinality is to store the different elements on a data structure. This can be done by first checking if each upcoming element is already maintained and adding it when it is not. This provides the exact cardinality but it requires storage space that grows linearly with the cardinality, thus is not practical when the number of distinct elements is large.

In many cases, the exact value of the cardinality is not needed and a reasonably good estimate is sufficient. Over the years many algorithms have been proposed to estimate cardinality [5], [6], [7], [8]. For example, Linear Probabilistic Counting Arrays (LPCAs) map elements to a bit array using a hash function and set one bit. Then the number of ones in the array is used to estimate the cardinality [5]. A disadvantage of LPCAs is that they provide a good estimate over a limited range of cardinality values. They also need a number of bits that is linear with the number of distinct elements. To overcome these limitations, sampling can be used, or the resolution of the bitmap can be dynamically adjusted [9]. Another option is to replace the bits with logarithmic counters, such that the range of cardinality values covered is extended [7]. This provides a good estimate for a wide range of cardinality values, while requiring a small amount of memory. An example of such an algorithm is the HyperLogLog (HLL) [8] that is widely used in computing systems [1]. HLL is for example supported by most cloud computing providers including Amazon Web Services [10] and Microsoft Azure [11] and used by most hyperscalers including Google and Facebook [1].

Using a hash function, the HyperLogLog algorithm maps each element to a counter in an array of counters. Then a second hash function is computed and the maximum number of leading zeros in the result for the elements mapped to that position is stored in the counter. The cardinality estimator is obtained by computing the harmonic mean of two to the power of the counter values and multiplying it by a correction factor [8]. The use of logarithmic counters makes it possible to estimate large cardinalities. For example, using five bits per

[1]*Pedro Reviriego is with Universidad Carlos III de Madrid, Leganés 28911, Madrid, Spain. email:* revirieg@it.uc3m.es
[2]*Jorge Martínez is with Universidad Antonio de Nebrija, 28040, Madrid, Spain. email:* jmartine@nebrija.es
[3]*Ori Rottenstreich is with the Department of Computer Science and the Department of Electrical Engineering, Technion, Israel. email:* or@technion.ac.il
[4]*Shanshan Liu and Fabrizio Lombardi are with Northeastern University, Dept. of ECE, Boston, MA 02115, USA. emails:* ssliu@ece.neu.edu, lombardi@ece.neu.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3013746, IEEE Transactions on Dependable and Secure Computing

counter cardinalities of billions can be estimated with only small deviations. Therefore, using a byte per counter is more than sufficient for most applications and very attractive for big data too [1]. The number of counter bits can be reduced to three if some restrictions are placed on the range of cardinality values that need to be estimated [12]. The HLL algorithm has also been used to simultaneously estimate the cardinality of multiple sets [13], [14].

Modern electronic systems are subject to various errors that can lead to a system failure [15]. For example, radiation induced soft errors change the value of a bit stored in memory or in a register from one to zero or from zero to one. If the change is not detected or corrected this may lead to data corruption and/or a system failure. For systems that run critical applications, Error Correction Codes (ECCs) can be used to protect the memories such that when a bit is flipped by an error, the system is capable to correct it [16]. The use of ECCs implies an overhead as additional memory bits are needed to store parity checks and circuitry needs to be added to encode (decode) the data when writing (reading) to (from) memory. In some cases, it may be sufficient to just detect errors and then a single parity bit can be used, so incurring in a lower overhead [17]. A different approach to protect an algorithm is to exploit its properties to detect and correct errors thus reducing the overhead needed for protection [18]. This is known as Algorithm-Based Error Tolerance (ABET) and has been widely used for example to protect signal processing algorithms like the Fast Fourier Transform [18] as well as also data structures like Bloom Filters [19]. However, to the best of the authors' knowledge, ABET has not been explored for state-of-the-art cardinality estimate algorithms like HLL. Moreover, the impact of soft errors on HLL has not been studied. Given the wide adoption of HLL, it is of interest to consider the effects of soft errors and to propose efficient protection techniques.

As the first contribution of this paper, we study the impact soft errors may have on the HLL cardinality estimate. To that end, we have performed simulations by error injection that show that most errors have a negligible impact on the estimate itself. However, there are errors that have a large and therefore significant impact for large cardinality values. Based on the analysis, we propose Remove Minimum (RM), an efficient protection scheme. RM is based on the observation that when an error on a HLL counter introduces a large change in the cardinality estimate, then that counter has the minimum (least) value across all HLL counters. Therefore, by not using it when computing the cardinality estimate, we ensure that the error has a negligible impact. The proposed scheme has been validated by error injection to show that soft errors on the counters do not have a significant impact on the cardinality estimate.

The rest of the paper is organized as follows. Section 2 covers the preliminaries by briefly describing the HLL algorithm, the error model considered and techniques that are commonly used to protect against soft errors. The effects of soft errors on HLL are evaluated in Section 3 that also provides an analysis of their impact. The proposed Remove Minimum scheme is presented in section 4 and evaluated in section 5. Section 6 discusses the effect of soft errors on Linear Probabilistic Counting Arrays (LPCAs), showing that the analysis presented in this paper can be extended to other algorithms. Conclusions and directions for future work can be found in Section 7.

2 Preliminaries

This section provides a brief description of the HyperLogLog (HLL) algorithm for cardinality estimate; it also describes the techniques used to protect memories and computational units from soft errors and presents the error model considered in the rest of this paper.

2.1 HyperLogLog

The HyperLogLog (HLL) algorithm [8] uses an array of M registers $r_1, \ldots, r_M$ for computing the number of distinct elements in a stream. An element x is mapped to a single register r_i using a hash function $h(x)$. Each element is mapped to a single register that is always the same regardless of how many times the element is mapped to the array. This is illustrated in Figure 1. Therefore, accessing an element in HLL requires a single memory access.

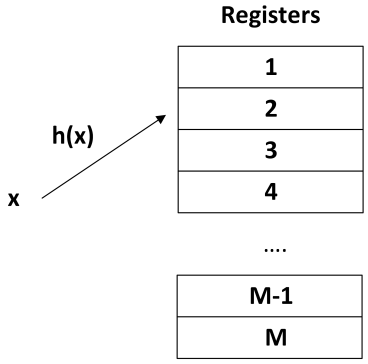

Fig. 1: Diagram of the array of registers used in HLL

Then, another hash function $g(x)$ is computed on the element and used to update the register value. So, each register keeps track of the maximum value of the position of the leftmost one (preceded by all zeros) in the bit vectors $g(x)$ for all elements mapped to that register. For example, a register to which two elements x and y are mapped with $g(x) = 00010011$ with the fourth bit as the left most bit of one and $g(y) = 00001001$ with the fifth bit as the left most bit of one stores a value of five because this is the position of the rightmost one preceded by zeros which in this case corresponds to element y. An update of a register in HLL is illustrated in Figure 2. In this case, the element y is added by first selecting the register with $h(y)$ and then updating its value to five based on $g(y)$. If y appears again later and it is added, then it will not increment the counter value. The principle behind these counters is that the probability of having a value with t trailing zeros is related to the number of elements mapped to the counter. This allows the counters to capture a wide range of cardinalities using only a few bits.

To estimate the cardinality in HLL, first the harmonic mean of the register contents is computed as:

$$Z = \frac{1}{\sum_{i=1}^{M} 2^{-r_i}}, \tag{1}$$

where r_i is the value of the i^{th} register.

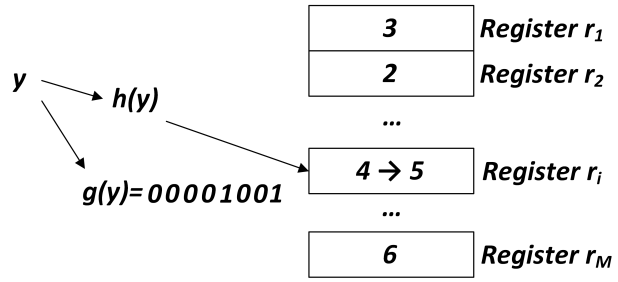

Fig. 2: Example of a register update in HLL

The cardinality estimate is then obtained by multiplying by a constant factor that depends on the number of counters used M:

$$C = \alpha_M \cdot M^2 \cdot Z \tag{2}$$

The factor is given by M^2 and a parameter α_M whose value is approximately 0.72 [8]. The HLL algorithm is shown in algorithm 1 in which the function ρ computes the position of the first one in the bits.

Algorithm 1 The *HyperLogLog* algorithm

Input: Stream of elements $S = (s_1, \ldots, s_k)$ Output: Estimated number of distinct elements in S

1: $r_0 = \ldots = r_{M-1} = 0$, $d = \log_2(M)$
2: $\alpha_M = 0.7213/(1 + 1.079/M)$ for $M \geq 128$
3: **for** $s_i \in S$ **do**
4: $(b_1, \ldots, b_d) = h(s_i)$
5: $(b_{d+1}, \ldots, b_{d+t}) = g(s_i)$
6: $j = (b_1, \ldots, b_d)$
7: $r_j = \max(r_j, \rho((b_{d+1}, \ldots, b_{d+t})))$
8: **end for**
9: $Z = \left(\sum_{j=0}^{M-1} 2^{-r_j} \right)^{-1}$
10: $C = \alpha_M \cdot M^2 \cdot Z$

The HLL algorithm finds a worse estimate for cardinalities smaller than $\frac{5 \cdot M}{2}$ and for very large cardinalities. In the first case, the solution proposed in the original HLL algorithm is to use counters arranged as a linear probabilistic counting array (LPCA) [8]. So, when the HLL estimate is low, then each counter is treated as a bit, and the LPCA cardinality estimate is used. An LPCA maps each element to an array of bits using a hash function and by setting the position given by the hash function to one. Then, the cardinality is estimated based on the number of ones in the array. Further details are given in section 6 when discussing the impact of soft errors on LPCAs. In the second case, the HLL estimate can be refined [8]. With these enhancements, HLL achieves a relative deviation of approximately $\frac{1}{\sqrt{M}}$ such that a specific accuracy level in the estimate can be obtained by selecting the number of counters M.

Hereafter in this paper, we focus on the cardinality range for which no correction is needed. This is reasonable because (as shown later in section 6 when discussing the impact of soft errors on LPCAs) soft errors have a negligible effect on HLL when the cardinality is small while for very large cardinalities, the impact will not significantly depend on the refined estimate.

The HLL algorithm has been extended over the years. For example, to provide alternative methods to estimate the cardinality for different ranges [20], to compute the cardinality estimate dynamically as elements are received to achieve a better accuracy [21] or to generalize the algorithm to estimate other statistics [22]. The use of a pool of shared counters to estimate the cardinality of multiple sets has also been considered to reduce memory requirements when the cardinality is estimated for many sets [13], [14]. Finally, privacy and security of HLL have also been studied in [23], [24]. All those works confirm the wide interest of HLL.

2.2 Error Model

The error model considered in this paper consists of soft errors that change either the value of a bit stored in memory [25], [26] or the output of a logic gate. Soft errors are typically caused by radiation. Traditionally, radiation-induced soft errors were only an issue for systems that operate in harsh environments like space, but they have become one of the major concerns for modern computing systems as technology scales [27]. A soft error appears randomly when a particle (e.g., a neutron) hits a memory cell or a circuit node; it can lead to data corruption or even system failure if no error detection or correction techniques are employed. These events are rare and thus for a given system it is assumed that there is a single soft error at a given time. For cardinality estimate as studied in this paper, radiation-induced soft errors cause random bit flips on the data stored, or random errors in an operation; they may lead to data corruption, or even failure, depending on when and where the errors occur.

For digital circuits, when the error affects a combinational node, it is known as a Single Event Transient (SET). A SET can only cause a failure if the error is stored in a register or memory cell. Instead, when the error affects a register or memory cell, it is known as a Single Event Upset (SEU), and the corrupted value is kept in the system until the register or memory is overwritten. This may lead to data corruption or system failure.

SEUs are the dominant type of soft errors due to a number of reasons. The first one is that many SETs are masked by the logic that is located after the affected node or do not reach a register input at the sampling time [28], [29]. Moreover, memories account for the majority of circuit area in many modern computer systems and chips. Finally, the memory cells are also prone to suffer radiation induced soft errors; so, soft errors in memory are a major concern for modern electronic systems [30]. Soft errors in computational units are also a challenge because they can affect for example arithmetic operations. Memories can be efficiently protected against soft errors using Error Correction Codes (ECCs); instead protection mechanisms for computational units tend to incur in a larger overhead as will be discussed in more detail in the next section.

In the case of the HyperLogLog algorithm, the main memory structure used is the array of counters. Therefore, the error model that is considered for the memory, is based on SEUs in the registers. In particular, single bit flips are considered. For computational errors, the main element of the HyperLogLog algorithm is the update operation that is executed for each element in the stream. Therefore, the error model deals with the errors that affect the update operations. Additionally, as the occurrence of a soft error is not relatively frequent, it will be assumed that during an HLL computation at most one soft

error occurs. This is in line with the assumptions commonly upheld to protect memories [16] and electronic circuits [30].

2.3 Protection against Soft Errors

Several techniques can be used to protect circuits against soft errors [31]. For memories, efficient protection can be provided by using Error Correction Codes (ECCs). ECCs add several parity bits per word to detect and correct errors. The most widely used codes are the single parity check (that can detect single bit errors) and Single Error Correction-Double Error Detection (SEC-DED) codes (that can correct single bit errors and detect double errors) [16]. The use of an ECC implies several overheads. Additional memory cells are needed to store the parity bits; also, circuitry must be added to compute these bits when writing to the memory and to use them to detect and correct errors when reading. In terms of additional memory, for a word of size w, parity introduces an overhead of $\frac{1}{w}$ while SEC-DED has an overhead of approximately $\frac{2+log_2(w)}{w}$. Therefore, the provision of SEC-DED incurs in a non-negligible overhead for practical word sizes; for example, for a 64-bit word, the overhead is 12.5%.

In this respect, an important consideration is that in many cases, the designer uses a hardware platform on which the memory is given and cannot be changed. In such common scenario, it is interesting to have the ability to implement protection at the algorithmic level, so that the algorithm can operate reliably even if the memory is unprotected or just parity protected for error detection.

For protection against errors, computational units can be replicated. When two copies are used, errors can be detected and with three copies, single errors can be corrected by voting [31]. Replication can be done by either physically having several copies and/or temporally by performing the same computation several times. For example, when running a calculation on a processor, it can be executed twice, and the results compared for the detection of errors. Independently of the replication scheme, the overhead is large in either circuitry (physical replication), or in execution time (temporal replication) with at least a 2x factor over an unprotected implementation. When protection is implemented by recomputing the result several times, the impact depends largely on the number of times that a function is executed. For example, in the case of HLL, the update procedure is run on each element of the stream, while the estimate is only computed once or at most a few times. Therefore, it is beneficial to have algorithmic techniques to protect the update operation without resorting to re-computation. Unfortunately, Algorithm-Based Error Tolerance techniques tend to be specific for each algorithm or type of algorithm and none has been proposed so far for HLL.

3 Analysis of Soft Errors on HLL

This section discusses the impact of soft errors on an unprotected HLL. First, the potential effects of soft errors in both memory and computational units are discussed. Subsequently, simulation by error injections is performed and based on the results, an initial analysis is presented. Then, the values of the counters in HLL are studied and modeled to better understand the effects of errors. Finally, errors are injected selectively on the different bits of the counters and the distribution of the values is used to present a more detailed analysis. This material provides the basis for the protection technique that is presented in the next section.

3.1 Initial Analysis

The effect of a soft error in a counter is evident; it flips one bit of the counter changing its value. When a bit changes from 0 to 1 (1 to 0), the error increases (decreases) the counter value and thus the cardinality estimate. Consider the effects of an error in a computational unit. As discussed previously, the main concern is the update operation that is run on each element of the data stream. The estimate operation is executed significantly less and thus, it can be protected by recomputation with a modest impact on the system performance. For an update, the error can affect the computation of $h(x)$. This means that an incorrect counter is read and updated. Similarly, the computation of $g(x)$ or the value obtained from it $v(x)$ can also be erroneous. In both cases, the end result in the worst-case outcome is to write an incorrect value to one of the HLL registers. This means that also errors in the computational units during an update can be modeled as a change in the value of one of the counters. Therefore, next the analysis and simulations focus on errors that modify the value of a counter.

To evaluate the impact of soft errors on the HLL counters, an existing software implementation has been used and modified for simulation by error injections. To improve the accuracy for low cardinalities, this implementation uses the LPCA estimate when the HLL estimate generates a value smaller than 2.5 times the number of HLL counters. Therefore, in the following discussion, only cardinality values larger than 2.5 times the number of counters are evaluated. Lower values will be considered when discussing the impact of errors on LPCAs in Section 6. The modified code used to implement and test the proposed scheme is available in a public repository so that other researchers can reproduce our results[1].

A configuration with $M = 1024$ counters of eight bits has been used and one bit error has been introduced per simulation. The procedure is as follows. First, a set with the desired number of distinct elements is randomly generated[2]; then, these elements are added to HLL. Once all elements have been added, the bit error is injected and then the cardinality is estimated. Injecting the bit error after adding all elements represents the worst case because the error cannot be corrected or mitigated by subsequent additions of elements. The process has been repeated for every bit on each counter, so that all possible errors are tested. This simulation was run for 1000 different sets of elements. The maximum, minimum and average values of the deviation of the cardinality estimate versus the error free estimate were logged to assess the impact of errors on the HLL counters.

The results for different cardinalities are shown in Table 1. The columns show the worst-case negative (minimum), the average, and worst-case positive (maximum) relative deviations from the error-free estimate over the 1000 runs. The results were similar across all the runs with standard deviations over the 1000 different sets smaller than 0.80% in all cases. For some values, errors can have a large effect, while for others the impact is limited. In particular, for small cardinalities, the HLL estimate seems to be robust against soft

1. https://github.com/mladron/ETHLL
2. Note that since HLL relies on hash functions, if they are well behaved, the nature of the input data should not influence the cardinality estimate

errors. This can be explained by considering the formula of the HLL estimator. When the cardinality is not significantly larger than the number of counters, each counter has a small value. Therefore, an error for which a counter takes a large value is equivalent to removing it from the estimate. Since the number of counters used in HLL is large, the impact is small. However an error that makes the counter smaller, does not make much of a difference because all counters are small.

This is better illustrated with an example. Consider a cardinality of 10,000 such that on average approximately ten distinct elements map to each counter. Then, a counter tends to have values in the range of two to nine. Consider an error that flips the fifth bit of a counter, so that it takes a value larger than sixteen. Then, the contribution of that counter to the harmonic mean is reduced by a factor of $2^{16} = 65536$. Therefore from a practical point of view this is equivalent to eliminating its contribution. However, since the number of counters M in HLL is usually large, the effect on the estimate is small. Consider next an error on the third bit for a counter that has a value of four. After the error, the counter takes a value of zero and instead of adding $\frac{1}{16}$ adds $\frac{1}{1}$ to the denominator of Z. However, the denominator of Z takes a value of approximately $\frac{\alpha_M \cdot M^2}{C}$ which for a cardinality $C = 10,000$ is significantly larger than one and thus, the increment due to the error is small.

For larger cardinalities, a soft error can introduce a large deviation in the cardinality estimate, because counters take large values. For example, for a cardinality of one million that corresponds to approximately 1,000 elements per counter, the value of a counter can easily be up to sixteen. Again, an error that makes a counter even larger has a limited impact, because in the worst case it removes the counter from the estimate. However, an error that makes a counter go from sixteen to zero, increases dramatically its contribution to the harmonic mean denominator, reducing the estimate. So, the contribution of the counter changes from $\frac{1}{65536}$ to 1 which is large compared to the denominator that takes a value of approximately 0.72. This effect for large cardinalities is clearly observed in the simulation results.

From the above discussion, it is evident that soft errors can have a large impact on the HLL estimate only when the cardinality is not significantly smaller than $\alpha_M \cdot M^2$ and only for errors that flip a one to a zero on the upper bits of the counter.

More formally, the maximum error that can be introduced by a single bit flip on a counter occurs when the bit flip sets the counter value to zero. This at most changes the value of the denominator for Z by one. Therefore, the maximum relative error R introduced in the cardinality estimate is:

$$R_{max} \approx \frac{\frac{1}{\frac{1}{Z}+1} - Z}{Z} = \frac{-1}{1+\frac{1}{Z}} \tag{3}$$

When Z is significantly smaller than one the error is small; however if Z is large, then the error can be large. As Z takes approximately the value $\frac{C}{\alpha_M \cdot M^2}$ the worst case relative error can be approximated by:

$$R_{max} \approx \frac{-1}{1+\frac{\alpha_M \cdot M^2}{C}} \tag{4}$$

The values of this approximation are shown in Table 2; as the cardinality increases, so does the potential impact of an error. Consider Table 1, the approximation matches well with the observed worst case deviations.

3.2 Counter Values in HLL

To better understand the impact of errors on the different counter bits, it is important to know the distribution of the counter values. Assume a set with cardinality C that is mapped to an HLL with M counters. Then, on average $a = \frac{C}{M}$ elements map to each counter. To better understand the values of the counters, consider one to which a elements are mapped. Then, the probability that none of the a elements has a value of $g(x)$ starting with t zeros can be computed for $t > 0$ as:

$$P_n(t) \approx \left(\frac{2^t - 1}{2^t}\right)^a = \left(1 - \frac{1}{2^t}\right)^a \tag{5}$$

Therefore, the probability that a counter has a value of t can be calculated as the probability of having $t-1$ and not having t zeros [20]. Since the second case is a subset of the first, the probability that the counter takes a value of t is approximated as per Eq. (6) by assuming that the probabilities of all hash values are equal.

$$P(t) = (1 - P_n(t-1)) - (1 - P_n(t)) = P_n(t) - P_n(t-1) \tag{6}$$

which is the probability that the counter has a value starting with $t-1$ and no value starting with t zeros, hence the first one is in position t.

For a closer look at the counter values, the probabilities $P(t)$ given by the previous equation have been computed for some of the cardinality values in Table 1 assuming $M = 1024$ and using $a = \frac{C}{M}$. The results are summarized in Figure 3. As the cardinality increases so do the counter values and in all cases most counters are concentrated on just a few values. Consider $C = 10^5$ as an example, the only counter values with a probability larger than 0.01% are those from 4 to 20. Instead for $C = 10^6$, the values with probabilities above 0.01% are those from 7 to 23.

As discussed before, the worst-case error in the HLL estimate occurs when a bit flip changes the counter value to zero. Assuming a single bit flip, this can only occur when the original counter value had only a bit set to one. This occurs when the value is a power of two: 1, 2, 4, 8, 16 and 32. Therefore, for $C = 10^5$, the worst-case impact on HLL occurs for errors on bits 2, 3, 4 that correspond to counter values of 4, 8 and 16. Instead, for $C = 10^6$ it occurs for errors on bits 3 and 4 only because a counter value of 4 is very rare.

3.3 Detailed Analysis

To further analyze the impact of soft errors, the same simulations have been performed for cardinalities of one hundred thousand and one million, but now by injecting errors on one bit position at a time. So first, inject errors on the upper bit of the counter and measure the impact; then on the next bit and so on. The results are shown in Tables 3 and 4.

Consider first the case of one hundred thousand cardinality. The bits on which errors cause a significant impact are bits 2, 3, 4. Consider also the counter value distribution of Figure 3 and the discussion on a previous subsection; then, the results are as expected and correspond to counter values of 4, 8 and 16 that are changed to zero as result of the bit flip.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3013746, IEEE Transactions on Dependable and Secure Computing

TABLE 1: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters

Cardinality	Minimum with Error	Average with Error	Maximum with Error
5000	-0.72%	0.03%	0.73%
10000	-1.42%	-0.02%	1.42%
50000	-6.76%	-0.27%	1.76%
100000	-12.90%	-0.58%	1.75%
500000	-42.02%	-2.03%	1.13%
1000000	-59.22%	-2.28%	1.14%
5000000	-87.93%	-3.48%	1.42%
10000000	-93.68%	-4.84%	1.39%

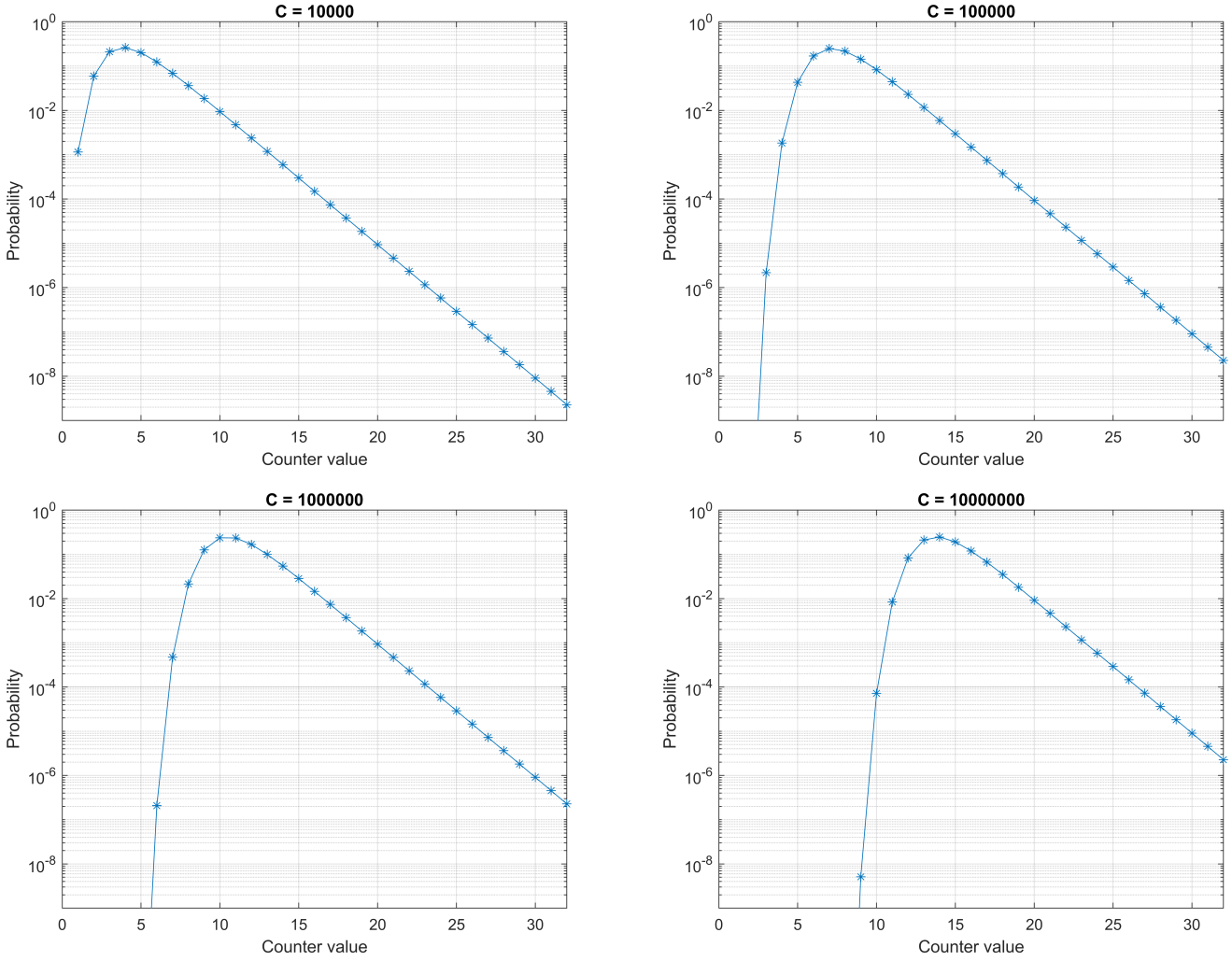

Fig. 3: Distribution of counter values when $a = \frac{C}{M}$ elements map to a counter with M=1024 and C = 10000,100000,1000000 and 10000000

Consider Table 4. In this case only bits 3 and 4 have a large impact on the cardinality estimate. This again is as expected from the counter value distribution of Figure 3 and the analysis of a previous subsection. However, bit 2 can also have a relevant impact although significantly smaller; this occurs when a counter value of 7 changes to 3, as result of bit 2 being flipped.

Finally, errors that change bits from 0 to 1 and thus increase the cardinality estimate, have a low impact in all cases with values below 2%. As summary, selective error injection on the different bits corroborate the previous analysis and observations: only errors that change a bit from 1 to 0 can have a large effect on the cardinality estimate and this can only occur for large cardinalities.

4 Protecting HLL Counters

After presenting the effects of soft errors on the counters for HLL cardinality estimate, in this section two protection techniques are presented. The first technique uses a parity bit per counter to detect errors and avoids the corruption of the estimator. The second technique exploits the HLL algorithm to protect against errors without adding parity bits.

TABLE 2: Maximum Relative Error (R_{max}) given by eq. (4)

Cardinality	Error
5000	-0.7%
10000	-1.3%
50000	-6.2%
100000	-11.7%
500000	-39.8%
1000000	-57.0%
5000000	-86.7%
10000000	-93.0%

Algorithm 2 Adding an element x to a parity protected HLL

```
Compute h(x)
Access counter in position h(x)
Check counter parity
if No parity error then
    Read and modify counter
    If modified recompute the parity
end if
```

4.1 Parity Protection for HLL

A simple approach to protect HLL counters against soft errors is to have a parity bit per counter. The procedures to add a given element x to a parity protected HLL and to compute the cardinality estimate are shown in algorithms 2 and 3. The parity needs to be checked each time an element is added to the HLL, even if the counter is not modified. This poses a significant overhead, because adding elements is typically the most frequent operation. In the second algorithm, when computing the cardinality estimate, if a parity error is detected on any counter, it is removed from the calculation. This ensures that only counters with correct values are used and as the number of counters is large, the effect on the estimate of not using one is small and can be neglected as discussed previously. Therefore, although discarding a counter can be compensated by using $M-1$ instead of M in the HLL estimate, this is not done in algorithm 3 to simplify the implementation.

However, the use of parity comes at a cost, because now an additional memory bit is needed per counter. For eight-bit counters, this represents an overhead of 12.5%. Additionally, the parity has to be checked every time a counter is read and recomputed when it is updated. Therefore, parity protection has a non-negligible cost in terms of memory and additional operations. Additionally, parity cannot protect against multiple bit errors as when the number of bits in error is even, the parity will not detect the errors.

4.2 Remove Minimum (RM) Protection for HLL

The analysis of the impact of soft errors on HLL counters in the previous section has shown that the main issue is errors that significantly reduce the counter value. So, the counter affected by the error has a significant smaller value than the remaining counters. The remaining errors have a very small impact on the estimated cardinality.

To avoid errors that significantly reduce the counter value and introducing large deviations in the HLL estimate, a simple process is to detect when the minimum (least) counter value is significantly smaller than other counters. In such a case, simply change its value to the lowest value among the remaining counters. This scheme (referred to as Remove Minimum (RM) hereafter in the paper) does not require to add any redundancy to the counters. RM also does not require any additional computation during the measurement phase. It only requires to keep track of the minimum (least) value when computing the harmonic mean and check if it needs to be removed.

Algorithm 3 Estimate cardinality in a parity protected HLL

```
for i ← 1 to M do
    Access counter in position i
    Check counter parity
    if No parity error then
        Add counter to the harmonic mean
    end if
end for
Apply constant factor
```

Algorithm 4 Adding an element x to an RM protected HLL

```
Compute h(x)
Access counter in position h(x)
Read and modify counter
```

The procedures to add a given element x to an RM protected HLL and to compute the cardinality estimate are shown in algorithms 4 and 5. In this case, protection does not require any additional operation when adding elements to HLL. This is interesting because as discussed previously this is typically the most common operation. Protection related operations are added only when computing the cardinality estimate; they are simple comparisons and keeping the minimum values.

The proposed RM algorithm can modify the HLL estimate in the absence of errors. This occurs when the minimum counter value is smaller than the second minimum by a threshold τ or more. To reduce this effect, a large value of τ can be used but this reduces the ability of RM to correct errors. Therefore, the probability of having a minimum different than the second counter value must be analyzed.

Consider Figure 3, the lower values have a steep probability curve. As, for $C = 10^6$, a counter value of 5 has a probability smaller than $4 \cdot 10^{-14}$ while a value of 8 has a probability of 0.021. So, the difference between the minimum and the second lowest counter values is small. Likely, at least two counter values with the minimum value may be present; in this case, RM does not modify the HLL estimate. Consider again $C = 10^6$ with $M = 1024$ counters, then the probability of a counter taking a value of 8 is approximately 0.021 and thus having 1024 counters, on average approximately 21 counters would be present with such value. Therefore, if that is the minimum value, RM must not modify the HLL estimate in an error free case. More generally, it is highly unlikely to have a minimum counter value c_{min} and not have counters with value $c_{min}+1$. Therefore, a threshold value of $\tau = 2$ ensures that in the absence of errors, the proposed RM technique has a negligible impact on the HLL estimate.

Consider the worst case of an error when the proposed RM technique is used. The lowest value that a counter can take without being corrected is $c_{min} - \tau + 1$. The error adds $\frac{1}{2^{c_{min}-\tau+1}}$ to the denominator of Z but now c_{min} increases because Z and the cardinality increase and thus they reduce

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3013746, IEEE Transactions on Dependable and Secure Computing

TABLE 3: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters

Bit	Cardinality	Minimum with Error	Average with Error	Maximum with Error
7	100000	0.00%	0.10%	1.75%
6	100000	0.00%	0.10%	1.75%
5	100000	0.00%	0.10%	1.75%
4	100000	-12.90%	0.07%	1.75%
3	100000	-12.86%	-3.70%	1.75%
2	100000	-12.19%	-1.16%	1.64%
1	100000	-4.92%	-0.16%	0.70%
0	100000	-1.69%	-0.02%	0.47%

TABLE 4: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters

Bit	Cardinality	Minimum with Error	Average with Error	Maximum with Error
7	1000000	0.00%	0.10%	1.14%
6	1000000	0.00%	0.10%	1.14%
5	1000000	0.00%	0.10%	1.14%
4	1000000	-59.22%	-1.15%	1.14%
3	1000000	-59.13%	-17.25%	1.14%
2	1000000	-14.48%	-0.04%	0.53%
1	1000000	-3.28%	-0.10%	0.43%
0	1000000	-1.12%	-0.03%	0.28%

TABLE 5: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters when protected with RM

Cardinality	Minimum with Error	Average with Error	Maximum with Error
5000	-0.72%	0.03%	0.73%
10000	-1.42%	-0.02%	1.42%
50000	-3.34%	-0.06%	1.77%
100000	-3.32%	-0.07%	2.20%
500000	-2.18%	-0.03%	1.13%
1000000	-2.20%	-0.05%	1.14%
5000000	-2.57%	-0.11%	1.69%
10000000	-2.50%	-0.10%	1.39%

the impact of the error. This is equivalent to have $2^{\tau-1}$ additional counters with value c_{min}. Assume that for example, a counter with minimum value contributes 8 times more than the average counter to the denominator of Z and a threshold of two, then the worst case of an error is equivalent to adding 16 average counter values to the estimate. As the number of counters is typically large, the impact of the error is small. For example, when $M = 1024$, the worst case impact is $\frac{16}{1024}$, which is less than 2%.

From the previous discussion, it seems that using a threshold value of $\tau = 2$ provides good results, ensuring both a very small impact in the absence of errors and also when there are errors.

From a designer's perspective, it is of important to know the worst-case impact that a soft error can have when RM is used. An upper bound on the magnitude of the impact can be derived as follows. Initially, the minimum value of the counters can be practically bounded by computing the expected value and subtracting four, because the probability that lower values occur, is negligible as shown in Section 3.2:

$$c_{min} \approx log_2\Big(\frac{C}{M}\Big) - 4 \tag{7}$$

The worst case for negative impact occurs when a counter takes the lowest value that is not removed, i.e., $c_{min} - \tau + 1$. Instead, for positive impact, the worst case occurs when a counter that had the minimum value c_{min} increases as result of the error. In both cases, the impact on the denominator of Z can be bounded by $2^{-(c_{min}-\tau+1)}$ and $2^{-c_{min}}$ respectively. Therefore, the relative error in the estimate can be bounded when negative by:

$$R_{Negmax} \approx \frac{\frac{1}{\frac{1}{Z}+2^{-(c_{min}-\tau+1)}} - Z}{Z} = \frac{-2^{-(c_{min}-\tau+1)} \cdot Z}{1 + Z \cdot 2^{-(c_{min}-\tau+1)}} \tag{8}$$

and when positive by:

$$R_{Posmax} \approx \frac{\frac{1}{\frac{1}{Z}-2^{-c_{min}}} - Z}{Z} = \frac{2^{-c_{min}} \cdot Z}{1 - Z \cdot 2^{-c_{min}}} \tag{9}$$

As discussed previously, Z takes approximately the value $\frac{C}{\alpha_M \cdot M^2}$ and $2^{-c_{min}} = 2^4 \cdot \frac{M}{C}$ (as per Eq. (7)); thus $2^{-c_{min}} \cdot Z = \frac{2^4}{\alpha_M \cdot M}$, which would be siginificantly smaller than one for practical values of M. Therefore, the worst-case relative errors can be approximated by:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2020.3013746, IEEE Transactions on Dependable and Secure Computing

Algorithm 5 Estimated cardinality in an RM protected HLL

```
min2 ← maxvalue
min ← maxvalue
for i ← 1 to M do
    Access counter in position i
    if counter is minimum so far then
        Add min2 to the harmonic mean
        min2 ← min
        min ← counter
    else if counter is smaller than min2 then
        Add min2 to the harmonic mean
        min2 ← counter
    else
        Add counter to the harmonic mean
    end if
end for
if (min2 − min) ≥ τ then
    Add min2 to the harmonic mean
else
    Add min to the harmonic mean
end if
Add min2 to the harmonic mean
Apply constant factor
```

$$R_{Negmax} \approx \frac{-2^4 \cdot 2^{\tau-1}}{\alpha_M \cdot M} \tag{10}$$

and

$$R_{Posmax} \approx \frac{2^4}{\alpha_M \cdot M} \tag{11}$$

The bounds (i.e., R_{Negmax} and R_{Posmax}) provide the designer with a simple method to estimate the worst-case impact of soft errors on the cardinality estimate when HLL is protected with the proposed RM technique.

Finally, as the proposed RM scheme is based on detecting changes in the counter value that make it smaller than the rest of the counters, it does not matter if the error affects one or several bits. Therefore and differently from parity, the proposed scheme can deal with multiple bits on the counter values.

4.3 Parity Protection vs Remove Minimum (RM)

After presenting the two protection techniques (i.e., parity protection and Remove Minimum (RM)), it is interesting to compare them and summarize their main features. In terms of implementation overheads, RM has obvious advantage because it does not require any additional memory bits or operations for updates. Instead, when parity is used, a bit must be added per counter and the parity must be checked on each update. In terms of the impact of errors, the erroneous counter is discarded for parity protection, such that the cardinality estimate is computed on the remaining $M-1$ counters. For RM, also one counter (the one with the lowest value) is discarded, but only if it is τ below the value of the next lowest value. In practical terms, $\tau = 2$ means that the worst-case absolute deviation can be approximately twice when using RM compared to parity protection. Therefore, choosing between the two techniques will mostly depend on the memory configuration (with or without parity) of the target system. Finally, in terms of error control capability, parity protection will only protect against an even number of bit errors, while RM can protect against any number of bit errors in a counter.

5 Evaluation

In this section, the proposed RM scheme is evaluated in terms of its effectiveness in protecting against soft errors and also in terms of the overheads introduced in the computation of the HLL.

5.1 Error Protection

To validate the proposed RM scheme, three sets of experiments have been performed. The same configuration as in Section 3 has been used: $M = 1024$ counters of eight bits and the RM τ has been set to 2 as discussed in the previous section.

In the first set of simulations, HLL was run with no errors and the estimates provided by the unprotected and the RM protected HLL were the same or very similar. This was done to ensure that the removal of the minimum counter when it is significantly smaller than the remaining counters does not impact the HLL estimate under normal operation.

In the second set of experiments, errors were injected on randomly selected bits (as in Section 3), but this time on the RM protected HLL. The results are summarized in Table 5. The impact on the cardinality estimate is smaller than 3.5% in all cases. The theoretical bounds given by Eqs. (10) and (11) are -4.4% and 2.2% in this case and the magnitude of the errors stays within these limits. This shows the effectiveness of the proposed RM scheme to protect the HLL counters.

In a third set of experiments, errors were injected selectively on each bit (as in Section 3), but this time on the RM protected HLL. The results are summarized in Tables 6 and 7; now errors on all bits have a low impact on the cardinality estimate and again it is below 3.5% in all cases.

5.2 Computational Overhead

The two most significant operations in HLL are the update of the structure with an element and the estimate of the cardinality. As discussed previously, the update operation is used very frequently, because it is run on every element of the data stream, while the estimate operation is only called when the user wants to calculate the cardinality estimate. The proposed scheme does not introduce any change in the HLL update procedure and thus updates have no overhead compared to an unprotected HLL. However, for the estimate, the proposed scheme needs to identify the counter with the minimum value and check the threshold to determine if it is removed from the estimate calculation. This requires additional operations that introduce an overhead. To quantify this overhead, the time required to perform an estimate has been measured for both the unprotected and the RM protected HLLs. The results show that on average, the HLL estimate requires 230 and 570 μseconds for the unprotected and RM protected HLLs respectively when running on an Intel Core i7 processor. This overhead is in part due to the implementation for RM that first reads the array of counters to locate the minimum and then reads it a second time to compute the estimate. In any case, the

TABLE 6: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters when protected with RM

Bit	Cardinality	Minimum with Error	Average with Error	Maximum with Error
7	100000	0.00%	0.10%	2.20%
6	100000	0.00%	0.10%	2.20%
5	100000	0.00%	0.10%	2.20%
4	100000	-1.76%	0.10%	2.20%
3	100000	-3.32%	-0.35%	2.20%
2	100000	-3.13%	-0.45%	1.64%
1	100000	-0.69%	-0.16%	1.98%
0	100000	-0.46%	-0.02%	1.75%

TABLE 7: Deviation on the estimated cardinalities over 1,000 runs of exhaustive injections of soft errors in the counters when protected with RM

Bit	Cardinality	Minimum with Error	Average with Error	Maximum with Error
7	1000000	0.00%	0.10%	1.14%
6	1000000	0.00%	0.10%	1.14%
5	1000000	0.00%	0.10%	1.14%
4	1000000	-2.13%	0.08%	1.14%
3	1000000	-2.20%	-0.61%	1.14%
2	1000000	-0.53%	-0.03%	0.57%
1	1000000	-0.42%	-0.10%	0.57%
0	1000000	-1.07%	-0.03%	0.57%

TABLE 8: Deviations in the cardinality estimates of an LPCA due to errors

Cardinality	Minimum with Error	Average with Error	Maximum with Error
500	-0.34%	0.18%	0.34%
1000	-0.27%	0.08%	0.27%
1500	-0.30%	0.05%	0.30%
2000	-0.40%	0.02%	0.40%

estimate is calculated in less than one millisecond and since the estimation is called in the scale of at least seconds, the overhead has a negligible impact on the processor utilization and performance.

6 Soft Errors on LPCAs

In addition to HLL, other count distinct algorithms rely on the use of an array of simple estimators. For example, Linear Probabilistic Counting Arrays (LPCAs) use a vector of M bits [5]. As discussed before for small cardinalities, HLL uses LPCA instead of the plain HLL estimate. This is done by considering each counter as a bit that is zero if the counter is zero and one otherwise [8].

In an LPCA, each element x is mapped to one of the bits using a hash function $h(x)$ and sets that bit to one. Then the cardinality is estimated as:

$$C = M \cdot log\Big(\frac{M}{V}\Big) \tag{12}$$

where V is the number of bits that are zero after all elements have been added to the array.

Consider the impact of a bit flip on an LPCA. It changes the number of bits that are zero from V to $V+1$ or $V-1$. However, in most cases this has a small impact because M is large. Therefore, LPCAs are resilient to soft errors and do not need to be protected.

To validate this analysis, exhaustive error injection has been performed on an LPCA with $M = 1024$ with sets of cardinalities of $C = 500, 1000, 1500, 2000$. The results for 1,000 runs are summarized in Table 8; in all cases, the impact on the cardinality estimate is small as correctly predicted by the analysis presented previously.

The same reasoning applies to other variants of LPCAs that use several resolution ranges [9]. This shows that the analysis of soft errors on count distinct algorithms beyond HLL can lead to interesting observations, or to the development of very efficient protection techniques.

7 Conclusion and Future Work

This paper has considered the effects of soft errors on HyperLogLog, a state-of-the-art algorithm for cardinality estimate. Soft errors on an HLL implementation were first injected to evaluate their impact on the estimated cardinality. The results have shown that most of the errors have little or no effect. However, there are a few errors that can severely affect the estimate; these correspond to bit flips that significantly reduce the value of an HLL counter.

After studying the effects of soft errors, protection of HLL has been considered and two protection techniques have been proposed. The first technique uses a parity bit per counter to locate errors and remove the affected counter from the estimate. The parity protection is effective but it introduces a

significant overhead in terms of memory as well as additional operations. The second technique referred to as the Remove Minimum (RM), exploits the features of the HLL algorithm to provide effective protection without requiring additional memory. A further advantage of RM is that it does not modify the procedure to add an element in HLL which is typically the most frequent operation. The proposed RM protection technique has been implemented and tested. The results have shown that it provides effective protection such that the cardinality estimate is not significantly modified in the presence of soft errors.

The work presented in this paper can be extended to several directions. For example, initially, derivation of the estimates for the average and variance of the impact of a soft error on the RM protected HLL would complement the worst-case analysis presented in this paper. The application of the proposed RM scheme to variants of HLL like [1] or [13] is also of interest. In fact, [1] only introduces minor modifications to the algorithm; they are related to the length of the hash function and the estimate and representation for low cardinalities; therefore, RM seems to be applicable also to this variant. As for [13], counters are shared to estimate the cardinalities of several sets, but they are used in a similar fashion as in the conventional solution. In this case, a soft error on a given counter would impact the cardinality estimates of all sets that map to it, but the impact on each of them would be similar and thus, RM may also be applicable. Finally, a further topic is to extend the Algorithm-Based Error Tolerance approach used in this paper to protect the HyperLogLog algorithm to other algorithms for cardinality estimate.

Acknowledgments

Pedro Reviriego would like to acknowledge the support of the TEXEO project TEC2016-80339-R funded by the Spanish Ministry of Economy and Competitivity and of the Madrid Community research project TAPIR-CM grant no. P2018/TCS-4496. The work of Ori Rottenstreich was partially supported by the Taub Family Foundation as well as by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate.

References

[1] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *International Conference on Extending Database Technology (EDBT)*, 2013.

[2] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2010.

[3] M. Cai, K. Hwang, J. Pan, and C. Papadopoulos, "Wormshield: Fast worm signature generation with distributed fingerprint aggregation," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 2, pp. 88–104, 2007.

[4] Y. Liu, W. Chen, and Y. Guan, "Identifying high-cardinality hosts from network-wide traffic measurements," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 547–558, 2016.

[5] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.

[6] Z. Bar-yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *RANDOM*, 2002.

[7] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *European Symposium on Algorithms (ESA)*, 2003.

[8] P. Flajolet, E. Fusy, O. Gandouet, and et al., "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *International Conference on Analysis of Algorithms (AOFA)*, 2007.

[9] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, 2006.

[10] L. Blog, "Data matters," *[Online] https://looker.com/blog/practical-data-science-amazon-announces-hyperloglog*, 2013.

[11] M. Azure, "Documentation," *[Online] https://docs.microsoft.com-/en-us/azure/kusto/query/dcount-aggfunctionestimation-accuracy*, 2020.

[12] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: Improving the performance of cardinality estimation of large data streams," in *IEEE Infocom*, 2017.

[13] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *ACM SIGMETRICS*, 2015.

[14] D. Ting, "Approximate distinct counts for billions of datasets," in *ACM International Conference on Management of Data (SIGMOD)*, 2019.

[15] N. Kanekawa, I. Eishi, T. Suga, and Y. Uematsu, *Dependability in electronic systems: Mitigation of hardware failures, soft errors, and electromagnetic disturbances*. 2011.

[16] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124–134, 1984.

[17] S. Liu, P. Reviriego, and F. Lombardi, "Detection of limited magnitude errors in emerging multilevel cell memories by one-bit parity (obp) or two-bit parity (tbp)," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019 (In press).

[18] Syng-Jyan Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 849–854, 1994.

[19] A. Sanchez-Macian, P. Reviriego, J. A. Maestro, and S. Liu, "Single event transient tolerant bloom filter implementations," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1831–1836, 2017.

[20] O. Ertl, "New cardinality estimation algorithms for HyperLogLog sketches," *CoRR*, abs/1702.01284, 2017.

[21] D. Ting, "Streamed approximate counting of distinct elements: Beating optimal batch methods," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2014.

[22] E. Cohen, "HyperLogLog Hyperextended: Sketches for concave sublinear frequency statistics," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2017.

[23] D. Desfontaines, A. Lochbihler, and D. A. Basin, "Cardinality estimators do not preserve privacy," *CoRR*, abs/1808.05879, 2018.

[24] P. Reviriego and D. Ting, "Security of HyperLogLog (HLL) cardinality estimation: Vulnerabilities and protection," *IEEE Communications Letters*, vol. 24, no. 5, pp. 976–980, 2020.

[25] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," *SIGARCH Comput. Archit. News*, vol. 43, no. 1, pp. 297–310, 2015.

[26] J. Suh, M. Manoochehri, M. Annavaram, and M. Dubois, "Soft error benchmarking of L2 caches with PARMA," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 85–96, 2011.

[27] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[28] I. Polian, J. P. Hayes, S. M. Reddy, and B. Becker, "Modeling and mitigating transient errors in logic circuits," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 537–547, 2011.

[29] F. Wang and Y. Xie, "Soft error rate analysis for combinational logic using an accurate electrical masking model," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 137–146, 2011.

[30] M. Ottavi, S. Pontarelli, D. Gizopoulos, C. Bolchini, M. K. Michael, L. Anghel, M. Tahoori, A. Paschalis, P. Reviriego, O. Bringmann, *et al.*, "Dependable multicore architectures at nanoscale: The view from europe," *IEEE Design & Test*, vol. 32, no. 2, pp. 17–28, 2015.

[31] M. Nicolaidis, "Design for soft error mitigation," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 405–418, 2005.

Pedro Reviriego received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BASE-T transceivers. From 2004 to 2007, he was a Distinguished Member of Technical Staff with the LSI Corporation, working on the development of Ethernet transceivers. From 2007 to 2018 he was with Nebrija University. He is currently with Universidad Carlos III de Madrid working on high speed packet processing and fault tolerant electronics.

Jorge Martínez received the B.Sc. degree in computer science from Tecnológico de Monterrey, México in 1989, the M.Sc. degree from Universidad Complutense de Madrid, Madrid, Spain, in 2013 and the Ph.D. from Universidad Antonio de Nebrija, Madrid, Spain in 2017. He is currently with Universidad Antonio de Nebrija. His research interests include fault tolerance and reliability.

Ori Rottenstreich is an assistant professor at the department of Computer Science and the department of Electrical Engineering of the Technion, Haifa, Israel. In 2015-2017 he was a Postdoctoral Research Fellow at the Department of Computer Science, Princeton university. Earlier, he received the B.Sc in Computer Engineering (summa cum laude), and Ph.D degree from the Electrical Engineering Department of the Technion, Haifa, Israel in 2008 and 2014, respectively.

Shanshan Liu received the M.S. degree and Ph.D. degree in mi-croelectronics and solid-state electronics from Harbin Institute of Technology, Harbin, China, in 2012 and 2018, respectively. She is currently a Post-doctoral researcher with the Department of Electrical and Computer Engineering, Northeastern University, Boston, US. Her current research interests include fault tolerant design in high performance computer systems.

Fabrizio Lombardi received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the masters degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests are bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He has ex-tensively published in these areas and coauthored/edited seven books. He was the Editor-In-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 2007 to 2010 and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2013 to 2017, IEEE TRANSACTIONS ON NANOTECHNOLOGY from 2014 to 2019. He is currently the Vice President for Publications of the IEEE Computer Society and a member of the executive committee of the IEEE Nanotechnology Council.