Implementing error detection in fast counting Bloom filters

P. Reviriego and J.A. Maestro

Bloom filters have found numerous applications in computing and networking systems. They are used to determine whether a given element is present in a set. Counting Bloom filters (CBFs) are an extension of Bloom filters that supports the removal of elements from the set. Traditional Bloom filters require several memory accesses to determine whether an element is present in the set. Recently, fast CBFs that can complete a search operation with only one memory access have been presented. Modern electronic systems are prone to soft errors. These errors can corrupt the contents of memories, causing system failures.

In the case of Bloom filters, errors can cause failures where an element that is in the set is classified as not being in the set and the other way around. To avoid those failures, a per-word parity bit is commonly added to detect errors in memories. It is shown that error detection can be implemented in fast CBFs without adding any parity bit.

This is achieved by exploiting the properties of the filters to implement error detection.

Introduction: In many networking and computing applications, there is a need for fast classification. Bloom filters are a simple data structure that provides membership check to a set without needing to store the elements of the set [1]. In more detail, a Bloom filter uses a set of $k$ hash functions $h_1, h_2, \ldots, h_m$ that map the incoming elements to positions in an array of $m$ bits. When inserting an element $x$ in the set, the bits in positions given by $h_1(x), h_2(x), \ldots, h_m(x)$ are set to one. Similarly, when checking if an element is in the set, those bits are read and if all of them are set to one, the element is considered to be in the set. A Bloom filter can return a false positive if those bits may have been set to one by other elements. The false positive rate can be minimised by properly dimensioning the bit array and the number of $k$ hash functions [1]. On the other hand, in the absence of errors, a Bloom filter cannot produce false negatives. A search operation in a Bloom filter requires, in general, $k$ memory accesses to read the bits in positions $h_1(x), h_2(x), \ldots, h_k(x)$.

The number of memory accesses has a direct impact on performance and several techniques have been recently proposed to reduce it. For example, fast Bloom filters [2] can complete a search operation in just one memory access. This is achieved by noting that most memory systems use large words, for example, 64 bits. Then, instead of using a large array of $m$ bits, smaller arrays of size $w$ bits are used, where $w$ is the memory word size. The fast Bloom filter is composed of $l = m/w$ such arrays and each of them is used as a standard Bloom filter. The insertion procedure is modified so that initially a hash function $h_w$ is used to select one of the $l$ words and then $k$ hash functions $h_1, h_2, \ldots, h_k$ are used to set $k$ bits to one in that word. Similarly, a search operation uses $h_w$ to select a word and then checks the bits in the positions given by $h_1, h_2, \ldots, h_k$.

Proposed error detection schemes: The construction of CBFs has some features that can be used for error detection. In particular, insertions increment exactly $k$ counters and deletions decrement also $k$ counters. This means that the sum of all the counters in the filter will always be a multiple of $k$. Therefore, single errors can be detected by checking this property. For a traditional CBF, this check would require reading the contents of the entire filter and adding the $m$ counters $(m$ being typically a large value of hundreds or thousands). This limits the usefulness of the scheme and makes it impractical to detect errors during search operations so that failures can be avoided.

For fast CBFs, the same reasoning applies. The key difference is that now the condition can be checked independently for each word. This means that only the data read as part of a search operation is needed to perform the check. However, when the CBF on each word is implemented using traditional counters, there is still a significant complexity for error checking. The counters in the word (now at most a few tens) have to be added to detect errors. When the counters are encoded using a hierarchical scheme as the one used in [3], the error checking can be performed more efficiently. This is based on the observation made in the Introduction that the number of ones in a word is equal to $k$ times the number of elements that have been stored in that word for that coding scheme. Based on this, two error detection schemes are proposed as follows.

The first error detection scheme has three steps, as shown in Fig. 2. It first counts the number of ones in a word, then computes the modulo $k$ on the number of ones and finally checks whether the result is zero. This algorithm can be implemented as part of search operations so that errors are detected before they can cause a failure. As noted in the previous works [3, 4], many modern processors have instructions to count the number of ones in a word so that the first step can be implemented efficiently.

For some values of $k$, the error detection scheme can be further simplified. This is done with the second error detection scheme that can be used when the number of hash functions used in the filter, $k$, is even. In this case, the sum of the number of ones stored in a word will always be even. Therefore, a simple parity check that computes the xor of all the bits in the word can be used to detect errors. This second method is also illustrated in Fig. 2. The second scheme is equivalent to the
traditional parity bit protection used for memories, but in the case of fast CBFs an extra parity bit is not needed. This method can also be implemented as part of search operations and has a lower complexity than the first one.

In terms of error detection, both methods detect all single bit errors. The first method can also detect some multiple bit errors depending on the bits affected and also on the value of \( k \). The second method will also detect multiple bit errors that affect an odd number of bits. In any case, single bit errors are the most critical pattern as most soft errors affect a single bit in a memory word.

**Conclusion:** In this Letter, two schemes have been proposed to efficiently implement error detection in fast CBFs. The two methods can detect all single bit errors and can be implemented as part of search operations with low overhead. This ensures that errors are detected before they can cause a failure. The proposed schemes do not require any additional memory and rely on the features and properties of fast CBFs to detect the errors.

© The Institution of Engineering and Technology 2014
25 August 2014
doi: 10.1049/el.2014.3097
One or more of the Figures in this Letter are available in colour online.

P. Reviriego and J.A. Maestro (Departamento de Ingeniería Informática, Universidad Antonio de Nebrija, C. Pirineos 55, Madrid, Spain)
E-mail: previrie@nebrija.es

**References**