August 2020

Cardinality Estimation Using A Bloom Filter

Anonymous

Cardinality Estimation Using A Bloom Filter

ABSTRACT

In certain applications, e.g., online advertising where user data may be utilized, it is a requirement to estimate the cardinality of data values. This disclosure presents a lightweight mechanism to determine data cardinality. A Bloom filter is updated for each key by setting bits that are identified based on hashing the key. To determine whether there are multiple keys in a set, the count of bits in the Bloom filter that are set is obtained and is compared with a threshold value. If the threshold value is met, it is determined that the data set has at least the corresponding cardinality, e.g., at least two keys, at least three keys, etc.

KEYWORDS

- Cardinality estimation
- Key collision
- Bloom filter
- Probabilistic data structure
- Online advertising
- User-identifiable information (UII)

BACKGROUND

In many situations, there are restrictions on how data is stored and/or utilized. For example, user-identifiable information (UII) can only be accessed and utilized with specific user permission and in compliance with privacy standards and rules.

In online advertising, advertisers that deliver ads via an advertising network can send parameters, e.g., name-value pairs, as part of an advertising event. It is possible to track such parameters in terms of the users that are associated with the event. Certain parameters may be

user-specific (e.g., include UII) and need to be filtered, while other parameters that are associated with multiple users are not user-specific (include no UII) and therefore, can be utilized directly. Filtering UII allows the advertising network to avoid storing UII. However, to enable such filtering, it is necessary that UII be identified.

DESCRIPTION

When there is a requirement to track different keys (e.g., that may or may not include UII) that are utilized in a particular context, an important goal is to if there is more than one unique key. To serve this requirement in a lightweight manner, it is important that the solution to detect uniqueness of keys use as little memory as possible. Further, when the keys include restricted data (e.g., UII), it is important that the original keys not be recoverable from any data structures utilized to track keys.

A Bloom filter is a probabilistic data structure that enables a quick and memory efficient lookup of whether an element is present in a set. A lookup of the Bloom filter is probabilistic in that it can confirm that an element either definitely is not in the set or is likely to be in the set. To provide a lightweight, non-reversible data structure that enables determination of the existence of more than one unique key, this disclosure utilizes a Bloom filter.

1. Initial value of Bloom filter: all 56 bits are zero

[00000000000000000000000000000000000000000000000000000000]

2. Bloom filter after insertion of a first key (key = {3136136361}) - 16 bits are set based on hashing the key

[00011110011001101100110110110000000000000000000000000000]

3. Bloom filter after insertion of a second key (key = {25666151}) - 5 additional bits are set based on hashing the key (some bits may be in common with those set for the first key

[00011110011001101100110110110000000000000000000011111]

Fig. 1: Bloom filter updates

Fig. 1 shows an illustrative example of a Bloom filter that is updated for each key that is utilized. Initially, all bits of the Bloom filter are zero. Update of the Bloom filter involves setting a subset of bits per key, identified by hashing the key. For example, a 56-bit Bloom filter can be utilized, with 16 bits used per key such that hashing the key identifies 16 bits of the Bloom filter that are then set . Per the property of the Bloom filter, bits that correspond to different keys are different with a very high probability; in other words, the likelihood of two different keys corresponding to the same set of bits (collision) is very low.

Thus, insertion of two or more keys results in 17 or more bits being set when two different keys are inserted. Further, an additional 8 bits can be utilized as an instance counter. The instance counter is usable to determine whether a sample of sufficient size has been generated. The Bloom filter and the instance counter are combined in a 64-bit value which provides a standard and efficient word size usable in modern computers.

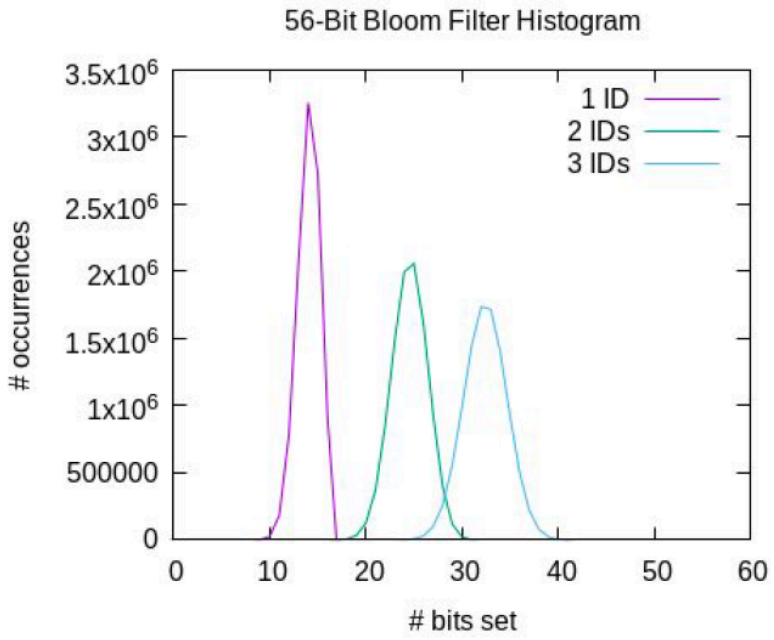

Fig. 2: Simulation result – number of bits set vs. keys inserted

To test whether multiple unique keys have been inserted, the number of bits of the Bloom filter that have been set are counted. If the bit count meets or exceeds a threshold, it can be concluded that there are multiple keys. Fig. 2 shows the results of a simulation with millions of iterations performed with insertion of one, two, and three keys (shown in different colors). As can be seen in Fig. 2, when 17 or more bits in the Bloom filter are set, it is an indicator that two or more unique keys have been inserted. With 25 or more bits set, there's about a two-thirds likelihood that three keys have been inserted. Thus, when 16 bits per key are utilized to update the Bloom filter, the value of the threshold (denoted as B) can be selected as 17 to confirm whether at least two keys have been used.

B can be adjusted based on the required precision and recall. Selection of a threshold B involves a tradeoff between precision and recall. Tests using between 1-10 keys show that precision increases with higher thresholds while recall decreases. For example, in testing with between 1-10 keys, the 17-bit test has a false negative rate of one in a million.

Therefore, if 17 or more bits of the Bloom filter are set, it can be determined with a high degree of confidence that two keys have been utilized. Confirming that at least two keys have been utilized is sufficient to determine that the information is not user-identifiable.. An appropriate value of the threshold can be chosen by experimenting with different values and selecting the one that provides a suitable precision/recall tradeoff. In the use of Bloom filter to ensure uniqueness of keys, recall failures may be tolerable, since a false indication that a key is UII results in a failure mode that preserves the privacy of user information.

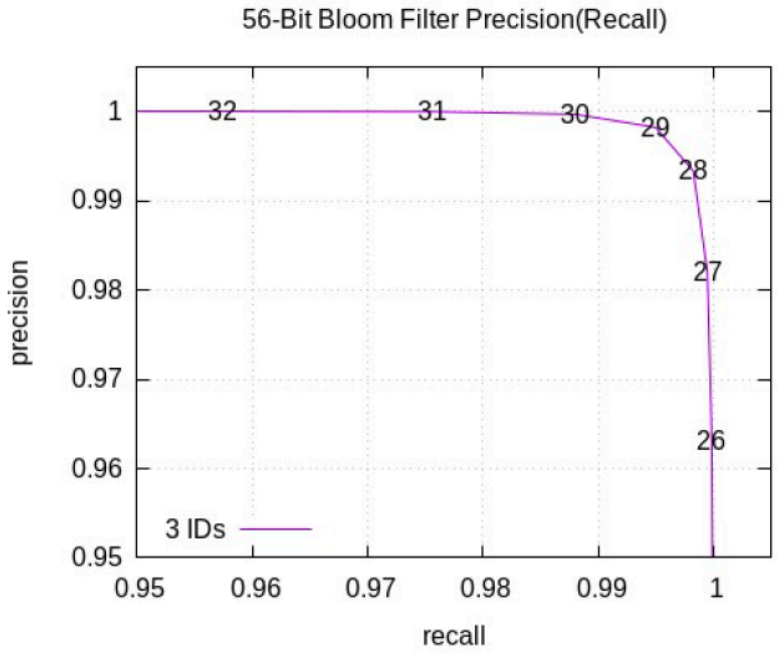

Fig. 3: precision-recall tradeoff curve

In a noisy system, the target can be set at three unique keys, rather than two unique keys. Fig. 3 shows an example of a precision/recall tradeoff curve for 3 keys. Different points on the curve in Fig. 3 are labeled with corresponding values of a threshold B that can be utilized to confirm whether at least 3 keys have been utilized. As seen in Fig. 3, When 32 bits of the Bloom filter are set, it is a near certain indication that three 3 keys have been inserted into the Bloom filter. While there is a low probability that insertion of two keys results in 32 bits of the Bloom

filter being set (in the extremely rare case of when hashing of two keys result in disjoint sets of 16 bits each), a value of B=33 can guarantee that there are at least 3 keys inserted.

Alternatively, the bit count can be used in conjunction with other criteria, rather than use a fixed value of B. Further, a four-key (or higher) threshold can also be used. However, the precision-recall tradeoff in such cases is more severe and gets worse for higher key thresholds.

CONCLUSION

In certain applications, e.g., online advertising where user data may be utilized, it is a requirement to estimate the cardinality of data values. This disclosure presents a lightweight mechanism to determine data cardinality. A Bloom filter is updated for each key by setting bits that are identified based on hashing the key. To determine whether there are multiple keys in a set, the count of bits in the Bloom filter that are set is obtained and is compared with a threshold value. If the threshold value is met, it is determined that the data set has at least the corresponding cardinality, e.g., at least two keys, at least three keys, etc.

REFERENCES

1. Bloom filter
2. Precision and recall