Filtering Malicious URLs Using Bloom Filter

Sangam Kedilaya
Computer Science
PES University
PES1201701139
kedilayasangam15@gmail.com

R Siva Girish
Computer Science
PES University
PES1201700159
sivagirish81@gmail.com

K S Sahazeer
Computer Science
PES University
PES1201700949
sahazeer123@gmail.com

Abstract

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. Bloom filters allow false positives but the space savings often outweigh this drawback when the probability of an error is controlled. Bloom filters have been used in database applications since the 1970s, but only in recent years have they become popular in the applications. The aim of this paper is to survey the ways in which Bloom filters have been used and modified in spam filter , with the aim of providing a unified mathematical and practical framework for understanding them and stimulating their use in future applications.

Keywords

Bloom filter; Hash function; boolean; vector;

I. Introduction

A bloom filter is space efficient probabilistic data structure, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not-in other words, a query returns either “possibly in set” or “definitely not in set”. Elements can be added, but not removed, the more elements that are added to the set, the larger the probability of false positive.Bloom proposed the technique for applications where amount of source data would require an impractically large amount of memory if conventional error-free hashing techniques where applied.While risking false positives, bloom filter have strong space advantage over other data structure representing sets, such as self-balancing binary search tree,tries,hash tables or simple arrays or linked list of the entries.

Most of these data structures require storing at least the data items themselves,which can require anywhere from a small number of bits, for small integer, to an arbitrary number of bits, such as for string.However bloom filter do not require to store data items at all.A bloom filter with 1% error requires only 9.6 bits per element.The one percent error rate can be reduced by a factor of ten by adding only about 4.8 bits per element.The theme unifying these diverse applications is that a Bloom filter offers a succinct way to represent a set or a list of items. A Bloom filter offers a representation that can dramatically reduce space, at the cost of introducing false positives.If false positives do not cause significant problems, the Bloom filter may provide improved performance. We call this the Bloom filter principle, and we repeat it for emphasis below

The Bloom filter principle: Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated.

II. Problem Statement

1) Use a bloom filter and analyse its performance on multiple hash functions and varying sizes of the filter.
2) Clone the Bloom filter used by google chrome browser to filter out malicious URLs

III. Dataset

Data collection for our spam filter application is from :

https://raw.githubusercontent.com/rlilojr/Detecting-Malicious-URL-Machine-Learning/master/dataset.csv

1) Tables were converted to CSV format.
2) All malicious URLs are stored using a bloom filter.Which results in ~ 99% size reduction.
3) A set of normal urls are tested using the bloom filter to find the false positive rate.

IV. Implementation

Here’s an overview of how a bloom filter stores data and produces a size reduction of around 98.6% :

1. A large input stream consisting of malicious urls is passed as input to the bloom filter.
2. A Bloom filter works on the basis of having multiple hash functions and a fixed size boolean array to store all the data.
3. The size of the boolean array is predetermined based on the number of inputs provided to the

bloom filter.A good Heuristic says that the size of the bloom filter should be :

$$Size = -\frac{n * ln(p)}{ln(2)^2}$$

Where n is the number of elements to be inserted into the bloom filter and p is the expected false positive probability.

4. The optimum number of hash functions is predetermined as well. A good heuristic says that the number of hash functions to be used is around :

$$K_{Hashes} = \frac{m}{n} * ln(2)$$

Where m is the size of the bloom filter and n is the number of elements to be inserted into the bloom filter.

5. Once the number of hash functions are known and the size of the bloom filter is determined.We construct a boolean vector array of dimensions Size and we use K_{Hashes} number of hash functions to hash each input and store it in the bloom array.
6. Each of the incoming input string is hashed using all K_{Hashes} hash functions.The hash value mod the size of the bloom array position is set to 1 or true.Similarly for all K_{Hashes} hash functions.
7. Upon completion of the aforementioned steps we have successfully created a bloom array and inserted all the elements into the array with a massive size reduction.
8. Now once the bloom filter is created we check whether a given input is already there in the bloom filter.
9. The input is hashed again using all the K_{Hashes} hash functions and the corresponding positions in the bloom array are checked.
10. If all the bits are set then the string may be present in the list else If even a single bit is not set then the string is definitely not in the boolean array.

V. Results

- There was a massive reduction in size by almost 98.6 %. The size of the filter is 55.00 KB which otherwise would have been 3.77 MB if any conventional data structure is used.
- We used around 56000 malicious URLs as input to the bloom filter and was tested against a million URLs having both malicious and non-malicious URLs. Around 3186 URLs were filtered incorrectly of which all were non-malicious which lead to a false positive rate of 0.003186.
- The bloom filter produced 0 False Negatives as expected.

VI. Acknowledgements

[1] Prof. Channa Bankapur, our guide and mentor for this project

VII. References

[1]https://raw.githubusercontent.com/rlilojr/Detecting-Malicious-URL-Machine-Learning/master/dataset.csv

[2]http://web.stanford.edu/class/cs166/handouts/100%20Suggested%20Final%20Project%20Topics.pdf