Tree-structured Bloom Filters for Joint Optimization of False Positive Probability and Transmission Bandwidth

Yongquan Fu
Science and Technology Laboratory of Parallel and Distributed Processing
School of Computer
National Univ. of Defense Technology
yongquanf@nudt.edu.cn

Ernst Biersack
www.e-biersack.eu
erbi@e-biersack.eu

ABSTRACT

Bloom filters are frequently used to perform set queries that test the existence of some items. However, Bloom filters face a dilemma: the transmission bandwidth and the accuracy cannot be optimized simultaneously. This dilemma is particularly severe for transmitting Bloom filters to remote nodes when the network bandwidth is limited. We propose a novel Bloom filter **BloomTree** that consists of a tree-structured organization of smaller Bloom filters, each one using a set of independent hash functions. BloomTree spreads items across levels that are compressed to reduce the transmission bandwidth need. We investigate in detail under which conditions BloomTree performs better than the compressed Bloom filter and the standard Bloom filter.

Categories and Subject Descriptors

C.2.0 [**Computer Systems Organization**]: Data communications

Keywords

Bloom filter; tree

1. INTRODUCTION

A Bloom filter (BF) [1] is a space-efficient compact data structure answering the approximate set queries [2, 3]. The Standard Bloom Filter (denoted as **SBF**) represents the set by randomly hashing each item into k bit locations into a bit array, where k denotes the number of hash functions. The Bloom filter incurs a certain false positive (FP) probability, since the hash locations may be already be set to ones by different items. However, a Bloom filter never has false negatives, i.e. an item is hashed to the bit array, but the query fails to report the existence of this item. To control the false positive probability, the size of the Bloom filter must grow linearly with the number of items in the set. Further, when Bloom filters must be exchanged among remote nodes, the transmission bandwidth is an important performance metric. In this case, using a Compressed Bloom Filter (denoted as **CBF**) may decrease the transmission cost [4]. However, CBF usually has two hash functions, which significantly degrades the accuracy compared to the Bloom filters with the optimal hash functions. As a result, there is still a gap between the space efficiency and the accuracy.

SIGMETRICS'15, June 15-19, 2015, Portland, OR, USA.
ACM 978-1-4503-3486-0/15/06.
http://dx.doi.org/10.1145/2745844.2745881.

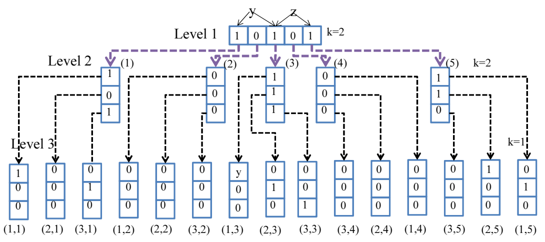


Figure 1: A three-level BloomTree instance. We represent a BloomTree with a tuple: $\{n_{\max}, d, \rho_1, k_1, m_{i,i>1}, k_{i,i>1}\}$, where $n_{\max}$ denotes the maximal number of items to be inserted, d represents the depth of the BloomTree, ρ_1 denotes the ratio between the number of bits and the number of items in the first-level filter, k_1 is the number of hash functions of the filter in the first level, $m_{i,i>1}$ is the size of the filter in the second and higher levels, and $k_{i,i>1}$ denotes the number of hash functions of the filter in the second and higher levels.

There exists a performance dilemma for BF based approximate set queries: *Given a fixed-size bit array, selecting the optimal number of hash functions that minimizes the FP probability will make compression ineffective, while selecting fewer hash functions will increase the compression efficiency but degrade the FP probability.* By how much a Bloom filter can be compressed depends on the probability $p \in [0, 1]$ that a bit in the array is zero [4]. For $p = 0.5$ the entropy of the bit array is at its maximum and no compression is possible: This is the case for the SBF when it uses the optimal number of hash functions. The closer p gets to either one or to zero, the higher the potential for compression.

We designed and implemented a novel Bloom filter called BloomTree (BT for short) that simultaneously decreases the average FP probability and optimizes the transmission cost. A BloomTree is a hierarchical organization of Bloom filters, as shown in Figure 1. Each vertex is a small-size filter. For each bit in the filter, this vertex has one separate descendent filter in the next level. BloomTree determines whether an item is hashed into it via independent tests. The number of such tests increases exponentially increasing with the number of levels in the tree, resulting in an exponentially decreasing FP probabilities as the tree grows in hight.

The tree structure provides novel opportunities for optimizing the FP probabilities, by improving the locality of the hash locations

Table 1: Some optimal configurations for BloomTree. The size n of the item set is set to $n_{\max} = 10^8$. Let W_{BT} denote the transmission size of a BloomTree instance. Let W_{CBF} and W_{SBF} be the transmission sizes of the CBF and SBF instances with the same geometric-mean FP probability with the BloomTree instance.

d	ρ_1	m_2	m_3	m_4	k_1	k_2	k_3	k_4	Posterior FP	$\frac{W_{BT}}{W_{CBF}}$	$\frac{W_{BT}}{W_{SBF}}$
2	0.5	3	–	–	6	1	–	–	0.84	0.33	0.39
3	0.5	4	3	–	5	3	2	–	0.21	0.43	0.55
4	0.5	5	3	3	6	3	2	2	1.02×10^{-5}	0.67	0.71

of data items, which in turn reduces the false positive probabilities of querying the items.

2. DESIGN

The BloomTree optimizes the false positives and decreases the transmission size through a novel tree-structured Bloom filter. As shown in Figure 1, a logical structure of the BloomTree has d levels of Bloom filters that are organized as a **tree**. The root of the tree is a SBF in the first level. For each bit in a SBF at level i, a **descendent** Bloom filter is appended at level $(i+1)$, $i < d$. The i-th level is also called the **ancestor level** for the $(i+1)$-th level. The level d is called the **leaf level**.

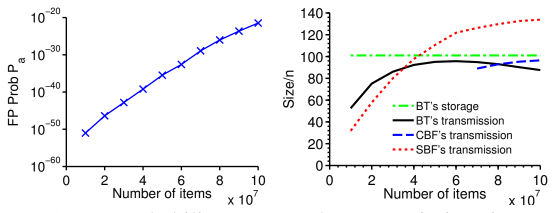
To insert an item, we first insert the item in the first-level Bloom filter like the SBF. We also record the hash positions of the item. We next insert the item into each descendent Bloom filter at these hash positions. The inserting process is recursively run until we complete the insertion on the bottom level. The query process is also a recursive process, where an item is said to be in the set *iff* all visited Bloom filters report that the item is in the set. Accordingly, a FP event occurs for a membership query *iff the visited Bloom filters all report that the item is in the set*.

Time Complexity: To ensure constant time complexity, the physical storage of the BloomTree is stored into a one-dimensional bit array. In the bit array, the ancestor-descendent links are computed on-th-fly with low time complexity. As a result, the storage structure of BloomTree is identical to that of the SBF. Thanks to the doubling hash method, we only use two hash functions to generate the hash values for all filters in the tree, as a result, we keep the hashing time of the overall query time on the BloomTree to be constant.

Accuracy: To control the number of bits that are set to ones, we recursively map each item to a small number of descendent filters layer by layer. As a result, some of the bits set for a given item have common prefixes through the tree, so those bits tend to be more clustered than for a traditional flat Bloom Filter.

The FP probability for an incoming item is a random variable, since the computed hash locations of each filter may lead to different branches in the hierarchical structure, as shown in Figure 1. As a result, optimizing the FP probability of the BloomTree is a stochastic-programming problem. We present a light-weight parameter tuning method with the sample average approximation method that minimizes the transmission size while limiting the FP probability to be under a fixed threshold with 95% confidence. As a result, we guarantee the FP probability for each incoming query "on average".

Space Efficiency: Observing that the bits that are set to ones are biased towards a subset of all possible bits to be set, the bit array can be efficiently compressed. For example, from Figure 1, we see that in the second and third level, many filters have all-0 bits or all-1 bits. As a result, the whole level can be efficiently compressed.



(a) FP probability (b) Transmission size

Figure 2: The FP probabilities and the transmission sizes as functions of the numbers of set items. The upper bound $n_{\max}$ of the set size is set to 10^8. We set d =4, ρ_1 = 1, m_2 = 4, m_3 = 4, m_4 = 5, k_1 = 6, k_2 = 3, k_3 = 3, k_4 = 2.

Results: We compare BloomTree with the standard Bloom filter and the Compressed Bloom filter. For a fair comparison, we create CBF and SBF instances whose geometric-mean posterior FP probabilities amount to those of the BloomTree instances. BloomTree's accuracy is determined by Bloom filters that are close to the leaves. The filters in the middle levels of the BloomTree serve as selectors for the SBFs close to the leaves. Increasing the number d of levels means that an exponential number of close-to-leaves Bloom filters are added. Therefore, BloomTree has exponentially decreasing FP probabilities as the depth d increases. From Table 1 we see that the BloomTree improves the space efficiency by 30-40% compared to the Compressed Bloom filter.

From Figure 2, we see that BloomTree's FP probability increases as we keep adding new items, since the percent of bits set to one at each level monotonically increases due to the constant storage of the BloomTree. We see that BloomTree requires the smallest transmission size compared to the CBF and the SBF as we continue to add more items. Moreover, the Compressed Bloom filters do not have a feasible parameter region for a wide range of experiments, since its transmission size becomes infinite due to too few hash functions.

Acknowledgments: This work was supported by the National Natural Science Foundation of China (Grant No. 61402509 and 61379052), the National High Technology Research and Development 863 Program of China (Grant No.2013AA01A213), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No.14JJ1026), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.20124307110015).

3. REFERENCES

[1] A. Z. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4), 2003.

[2] Y. Fu and Y. Wang. BCE: A Privacy-preserving Common-friend Estimation Method for Distributed Online Social Networks without Cryptography. In *Proc. of CHINACOM*, pages 212–217, 2012.

[3] Y. Fu, Y. Wang, and W. Peng. CommonFinder: A Decentralized and Privacy-preserving Common-friend Measurement Method for the Distributed Online Social Networks. *Computer Networks*, 64:369–389, 2014.

[4] M. Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM Trans. Netw.*, 10:604–612, 2002.