Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Bloofi: Multidimensional Bloom filters

Adina Crainiceanu [a,*], Daniel Lemire [b]

[a] US Naval Academy, United States
[b] LICEF Research Center, TELUQ University of Quebec, Canada

ARTICLE INFO

Keywords:
Bloom filter index
Multidimensional Bloom filter
Federated cloud
Data provenance

ABSTRACT

Bloom filters are probabilistic data structures commonly used for approximate membership problems in many areas of Computer Science (networking, distributed systems, databases, etc.). With the increase in data size and distribution of data, problems arise where a large number of Bloom filters are available, and all of them need to be searched for potential matches. As an example, in a federated cloud environment, each cloud provider could encode the information using Bloom filters and share the Bloom filters with a central coordinator. The problem of interest is not only whether a given element is in any of the sets represented by the Bloom filters, but also which of the existing sets contain the given element. This problem cannot be solved by just constructing a Bloom filter on the union of all the sets. Instead, we effectively have a multidimensional Bloom filter problem: given an element, we wish to receive a list of candidate sets where the element might be.

To solve this problem, we consider three alternatives. Firstly, we can naively check many Bloom filters. Secondly, we propose to organize the Bloom filters in a hierarchical index structure akin to a B+ tree that we call Bloofi. Finally, we propose another data structure that packs the Bloom filters in such a way as to exploit bit-level parallelism, which we call Flat-Bloofi.

Our theoretical and experimental results show that Bloofi and Flat-Bloofi provide scalable and efficient solutions alternatives to search through a large number of Bloom filters.

Published by Elsevier Ltd.

1. Introduction

Bloom filters [3] are used to efficiently check whether an object is likely to be in the set (match) or whether the object is definitely not in the set (no match). False positives are possible, but false negatives are not. Due to their efficiency, compact representation, and flexibility in allowing a trade-off between space and false positive probability, Bloom filters are popular in representing diverse sets of data. They are used in databases [23], distributed systems [5], web caching [14], and other network applications [4]. For example, Google BigTable [6] and Apache Cassandra [29] use Bloom filters to reduce the disk lookups for non-existent data. As digital data increases in both size and distribution, applications generate a large number of Bloom filters, and these filters need to be searched to find the sets containing particular objects.

Our work is motivated by highly distributed data provenance applications, in which data is tracked as it is created, modified, or sent/received between the multiple sites participating in the application, each site maintaining the data in a cloud environment. Bloom filters can be maintained by each individual site and shared with a central location. For each piece of data, we need to find the sites holding the data. Thus, we may need to search through a large number of Bloom filters stored at the central location.

Indexing Bloom filters is different from indexing generic objects to improve search time. There is one level of indirection between the elements searched for, and the objects

* Corresponding author. Tel.: +1 410 293 6822; fax: +1 410 293 2686.
E-mail addresses: adina@usna.edu (A. Crainiceanu), lemire@gmail.com (D. Lemire).

http://dx.doi.org/10.1016/j.is.2015.01.002
0306-4379/Published by Elsevier Ltd.

directly indexed by the index structure. In particular, each Bloom filter is a compact representation of an underlying set of elements. The question of interest is an *all-membership* query: given a particular element (not a Bloom filter), which underlying sets contain that element? The query subject is an element, but the objects we are indexing and searching through are Bloom filters, so what we are creating is a meta-index. The traditional index structures, such as hash indexes, B+trees, R trees and their distributed versions [1], do not directly apply in this case as we are indexing Bloom filters and not the keys themselves. All we are given from each site is a Bloom filter.

There has been significant work in using Bloom filters in various applications, and developing variations of Bloom filters. Counting filters [14,15] support deletions from the Bloom filter; compressed Bloom filters [21] are used with web caching; stable Bloom filters [11] eliminate duplicates in streams, spectral Bloom filters [7] extend the applicability of Bloom filters to multi-sets, multi-class Bloom Filter (MBF) [20] use per-element probabilities. Yet there has been few attempts to accelerate queries over many Bloom filters, what we call the *multidimensional Bloom filter problem*, even though our problem is closely related to signature file methods (see Section 8) where one seeks to index set-value attributes.

To solve this problem, we propose *Bloofi* (*Bloo*m *f*ilter *i*ndex), a hierarchical index structure for Bloom filters. Bloofi provides probabilistic answers to *all-membership* queries and scales to tens of thousands of Bloom filters. When the probability of false positives is low, Bloofi of order d (a tunable parameter) can provide $O(d\log_d N)$ search cost, where N is the number of Bloom filters indexed. Bloofi also provides support for inserts, deletes, and updates with $O(d\log_d N)$ cost and requires $O(N)$ storage cost. In designing Bloofi, we take advantage of the fact that the bitwise OR between Bloom filters of same length, constructed using the same hash functions, is also a Bloom filter. The resulting Bloom filter represents the union of the sets represented by the individual Bloom filters. This property allows us to construct a tree where the leaf levels are the indexed Bloom filters, and the root level is a Bloom filter that represents all the elements in the system. This tree is used to prune the search space (eliminate Bloom filters as candidates for matches) while processing *all-membership* queries. Our performance evaluation shows that Bloofi performs best when the false positive probability of the union Bloom filter (a Bloom filter that is the union of all the indexed Bloom filters) is low and provides $O(d \times \log_d N)$ search performance in most cases, with $O(N)$ being the storage cost and $O(d \times \log_d N)$ the maintenance cost. Bloofi could be used whenever a large number of Bloom filters that use the same hash functions need to be checked for matches.

Bloom filters are constructed over bitmaps, i.e., vector of Booleans. With bitmaps, we can exploit bit-level parallelism: on a 64-bit processor, we can compute the bitwise OR between 64 bits using a single instruction. We use bit-level parallelism with Bloofi to optimize the construction of the data structure. However, we have also designed an alternative data structure that is designed specifically to exploit bit-level parallelism (henceforth Flat-Bloofi). Though not as scalable as Bloofi, it can be fast when the number of Bloom filters is moderate.

This paper is an extended version of "Bloofi: A Hierarchical Bloom Filter Index with Applications to Distributed Data Provenance" [9] published in the Second International Workshop on Cloud Intelligence Cloud-I 2013. The paper was completely revised, and the new version introduces an additional data structure, Flat-Bloofi, a new implementation for Bloofi that improves the performance by an order of magnitude, and a new performance evaluation.

The rest of this paper is structured as follows: Section 2 describes a distributed data provenance application for Bloofi. Section 3 briefly reviews the concept of Bloom filter. Section 4 introduces Bloofi, a hierarchical index structure for Bloom filters. Section 5 introduces the maintenance algorithms and a theoretical performance analysis. Section 6 introduces Flat-Bloofi, a data structure for the multidimensional Bloom filter problem, designed to exploit bit-level parallelism. Section 7 shows the experimental results. We discuss related work in Section 8 and conclude in Section 9.

2. Motivation: application to distributed data provenance

In this section we describe the distributed data provenance application that motivated our work on Bloofi. Let us assume that a multinational corporation with hundreds of offices in geographically distributed locations (sites) around the world is interested in tracking the documents produced and used within the corporation. Each document is given a universally unique identifier (uuid) and is stored in the local repository, in a cloud environment. Documents can be sent to another location (site) or received from other locations, multiple documents can be bundled together to create new documents, which therefore are identified by new uuids, documents can be decomposed into smaller parts that become documents themselves, and so on. All these "events" that are important to the provenance of a document are recorded in the repository at the site generating the event. The events can be stored as RDF triples in a scalable cloud triple store such as Rya [25]. The data can be modeled as a Directed Acyclic Graph (DAG), with labeled edges (event names) and nodes (document uuids). As documents travel between sites, the DAG is in fact distributed not only over the machines in the cloud environment at each site, but also over hundreds of geographically distributed locations. The data provenance problem we are interested in solving is finding all the events and document uuids that form the "provenance" path of a given uuid (all "ancestors" of a given node in the distributed graph).

Storing all the data, or even all the uuids and their location, in a centralized place is not feasible, due to the volume and speed of the documents generated globally. Fully distributed data structures, such as Chord [28] or P-Ring [10], require even more communication (messages) than the centralized solution, increasing the latency and bandwidth consumption, so they are also not feasible due to the volume and speed of the documents generated globally. Moreover, local regulations might impose restrictions on where the data can be stored. However, since all the global locations belong to the same corporation, data exchange and data tracking must be made possible.

Without any information on the location of a uuid, each provenance query for a uuid must be sent to all sites. Each site can then determine the local part of the provenance

Please cite this article as: A. Crainiceanu, D. Lemire, Bloofi: Multidimensional Bloom filters, Information Systems (2015), http://dx.doi.org/10.1016/j.is.2015.01.002