# Bloom filters and their applications

Fedor Nikitin

June 11, 2006

## 1 Introduction

The bloom filters, as a new approach to hashing, were firstly presented by Burton Bloom [Blo70]. He considered the task of presenting a set as a sequence of bits, which is called hash code of the set. In comparison with the conventional hash-coding method, author offered another technique allowing to get more concise presentation of the set, but with some errors. There are applications in which errors with some allowable level are permitted. Allowing the possibility of these errors, we can reduce a space required for presentating a set in such applications. An example is a spell-checking system where we allow some amount of mistakes in presentation of dictionary.

We begin with the theory of Bloom filters. Subsection 2.1 deals with conventional hash-coding method. Firstly we introduce a hash-coding method, where no errors are permitted and then we explore how space requirements can be reduced by allowing errors in the system. In subsection 2.2 we give the definition of Bloom filter and compare this method with the previous one. All material presented in this section is based on original Bloom's paper [Blo70].

In subsection 2.3 we present generalization of Bloom filters for presentation of multi-set proposed in [CM03]. The main advantages of new concept are presented.

In the section 3 we consider one application of bloom filters. It deals with the so called differential file access problem. This problem usually arises in systems where simulteneous access to a file is required. The bloom filters are used there for a concise presentation of a differential file. The material of this section is based on [Mul83, Gre82]

# 2 Theory of Bloom filters

## 2.1 Conventional hash-coding method.

Conventional hash-coding method is a classical approach for the task of presenting a set. We start with it in order to cover possible approach to this task. The second reason is that Bloom in his paper [Blo70] also presents it and treats the concept of Bloom filter as a generalization or improvement of classical method.

Let us suppose that we have $n$ messages, each $b$ bits long. The hash area, that is the place where result will be stored, is $h$ cells of $b+1$ bits each. It is assumed, that $h > n$. *Conventional hash-coding method* involves a countable set $\{\xi_i\}_{i=1}^{+\infty}$ of random functions which are defined over the universal set of all possible messages whose values lie in $\{0, 1, \ldots, h-1\}$. For hashing a given set of $n$ messages we apply function $\xi_1$ to each message. We treat the result of applying as an address in the hash area for the message. We check if $(b+1)$th bit empty or not in this cell. In the first case we have to record the current message to the first $b$ bits and then set $(b+1)$th bit to 1. Otherwise, we apply random function $\xi_2$ and perform the same checking as at first step and so on until zero $(b+1)$th bit is encountered.

The next thing, which we are interested in, is how to check a given message for membership. This can be achieved by the following procedure. We apply $\{\xi_i\}_{i=1}^{+\infty}$ functions for the message. We stops if one of two cases has occured.

1. The $(b+1)$th bit in the cell of the hash area is equal to 1 and content of the first $b$ bits coincides with the message. In this case we say that the message is *accepted*.

2. The last bit of the cell is equal to zero. It means that the message does not belong to hashed set and we say that the message is *rejected*.

It is easily seen that the conventional hash-coding method does not allow any errors in the presentation of a given set. However, the hash-presentation of set takes more space than originally. The main benefit of method is that using of hash presentation of the original set we can perform a memebership queries faster.

An approach to avoid above mentioned problem of increased space required for storing the hash-presentation of a set is presented in [Blo70]. The is to organize the cells as before, but the cells are smaller and contain some code of the message instead entire message. If we choose the size of cell $c < b$, the space is reduced. But in this case the absence of errors cannot be guarantee.

## 2.2 The Bloom filter

In the previous subsection we saw that there is some method which allows to reduce space requirements by permiting errors. Bloom offered another method and showed that this method is more efficient than the previous one [Blo70].

Suppose that the hash area is organized as $N$ bits. Each bit has an address, which varies in $\{0, \ldots, N-1\}$. Initially, the value of all bits set to 0. The Bloom filter method involves $d$ independent random functions $\{h_i(\cdot)\}_{i=1}^{d}$, which are defined over universe $U$ with values belonging to the set $\{0, \ldots, N\}$. For mathematical convenience it is assumed that these functions have uniform distribution. For the given message $s \in S$ the results $\{h_i(s)\}_{i=1}^{d}$ of applying random functions can be treated as addresses in the hash area. For each of these addresses we set the corresponding bit to 1. The same procedure is repeated for each message from the set $S$. The final result is called a *Bloom filter* for the $S$. So we have packed the subset $S$ to sequence of $N$ bits.

To test a new element $s$ for membership a sequence of addresses $\{h_i(s)\}_{i=1}^{d}$ is generated. If all of the corresponding bits are equal to 1, the message is *accepted,* otherwise it is *rejected.*

The definition of Bloom filters implies that this technique can give false result, but the errors can be only false positive. False positive result means that the message is accepted, when it does not belong to the subset $S$. All possible situations are presented in the following table

| | positive | negative |
|---|---|---|
| true | — | — |
| false | possible | impossible |

Here "true" and "false" mean correctness of Bloom filter's membership query. "positive" and "negative" are the results of membership queries for Bloom filter. Sign "—" means that we have no error in this situation.

For the purposes of an analysis of the bloom filter method some measures should be defined. The most important one is *allowable fraction of errors.* It is defined by

$$P = \frac{n_a - n}{n_t - n},$$

where $n$ is the number of messages in the set to be hashed ($n = |S|$), $n_a$ is the number of messages in the universe accepted by bloom filter as a members of $S$ and $n_t$ is total number of messages in universe space ($n_t = |U|$). The next measure is the space which is needed for storing the presentation of a set. From the definition of Bloom filter it is seen that this factor is equal to $N$.

However, for the estimation of efficiency of Bloom filters some normalized measure should be chosen. The time measure is the average time which is required for rejecting message as a member for a given set.

## 2.3 The Spectral Bloom filter

The *Spectral Bloom Filter* (SBF) is a generalization of original Bloom filters for multi-set membership queries. This extension has a useful property that it allows to perform deletion and insertion operations over the presentation of a set.

A multi-set is extension of a set for the case, where an element can has multiplicity. It means that several counterparts of element may belong to the same set. Formally, it is defined as a pair $(S, f)$ of $S$ from universe $U$ and function $f : S \to \{1, 2, \ldots\}$. The value of function $f$ on element $x \in S$ is the frequency of element $x$ in the set $S$.

The hash area for SBF is organized as vector with $N$ counters $C_i$. Hence each counter $C_i$ has address and a value, which varies in $\{0, 1, 2, \ldots\}$. Initially, all counter values are equal 0. For the packing of given set $S$ to the hash area the same a hash functions $\{h_i(\cdot)\}_{i=1}^{d}$ are used. A procedure of adding new element $x \in S$ to the presentation of set $S$ is following. Firstly, we apply every hash function to element $x$. The values $\{h_i(x)\}_{i=1}^{d}$ are treated as addresses of counters to be increased by 1. We repeat this procedure for element $x$ as many times as many counterparts of element are in the set $S$. The SBF for the whole set $S$ can be produced by repeatedly adding all elements of $S$.

Unlike the regular Bloom filter we are interested in not only presence of given element in the set, also we want to define or, when it is impossible, to estimate the frequency $f_x$. The procedure of getting estimation $\hat{f}_x$ for the value $f_x$ is following: Let

$$v_x = \{C_{h_1(x)}, \ldots, C_{h_d(x)}\}$$

be a sequence of counter values which presents the element $x \in S$. The minimal value $m_x$ of sequence is called *Minimal Selection (MS)* estimator, that is

$$m_x = \min_{i \in \{1,\ldots,d\}} \{C_{h_i(x)}\}.$$

It turns out that for this estimation the following claim holds

**Claim 1** *For all $x \in U$, $f_x \leq m_x$. Furthermore, $f_x \neq m_x$ with probability $E_{SBF} \approx (1 - e^{-dn/N})^d$, where $n = |S|$.*

We skip the proof of this claim.

As we have already mentioned SBF allows easily perform deletion operation over presentation of a set. If we have an element $x \in S$ and our goal is to delete this element from the presentation we follow an easy procedure. Firstly we generate addresses of counters using hash functions and then decrease the values of corresponding counters by one. Obviously, in the case when we are sure that the element, which we are going to delete, belongs to the set $S$, this procedure works correctly.

From the definition of SBF it is seen that we can use these filteres for presentatiing a regular set. The reason is that a regular set is multi-set where function $f$ is identically equal to 1. For the presenting regular set SBF has an advantage of performing the deletion operation which can be useful in many applications. On the other hand regular bloom filter is more space-efficient in comparison with SBF.

# 3 An applications of bloom filters

## 3.1 Formulation of differential file access problem

Differential file access is one of the problems where Bloom filters can be utilized. There exist systems in which simultaneous access to the same file is needed for different users. For example it can be any on-line system which uses database for storing information. Many users can have intensions to update the information in this database. If we allow to perform several such operations at the same time, we can not guarantee that the information in the database will remain correct. This problem is usually solved by differential file access updating. The idea is to introduce a new file which is called a differential file and write all changes which were done by users to this file, while the main file of database remains unchanged. When the user performs some updating operation on the database the system has to check differential file and according to the records in this file add some records about new user's changes to the end of file. On the other hand if user's intention is just to get current information from database the system also has to check differential file access before sending information in order to make all changes which were done by another user up to this time. Then at some interval the database can be locked and an external program performs all changes which are stored in the differential file on the original database. The differential file becomes empty and we can use it again. We choose this interval when our system is not used. For some systems it can be, for example, night time.

The above described approach allows us to avoid in principle any collision

| | *Number of hash functions* | | |
|---|---|---|---|
| *Filter size (bytes)* | 4 | 6 | 8 |
| 24,576 | 0.356 | 0.431 | 0.516 |
| 28,672 | 0.273 | 0.333 | 0.407 |
| 32,768 | 0.210 | 0.252 | 0.322 |
| 40,960 | 0.125 | 0.146 | 0.197 |
| 49,152 | 0.078 | 0.082 | 0.107 |
| 57,344 | 0.056 | 0.055 | 0.063 |
| 65,536 | 0.035 | 0.029 | 0.044 |

in database. However a new problem appears. For most systems differential file is large and checking this file is time consuming. It makes our system slower and we are interested to speed up the checking for the differential file. The idea is to use bloom filter presentation of the differential file. Suppose that we already created Bloom filter for differential file. How the system should work now? When a user makes a query on the database, our system first of all checks if the corresponding records modified or not using the bloom filter. This operation can produce only true and false results. In the second case we guarantee that the record which user accesses to is unchanged. It follows from the property of Bloom filters, because they produce false positive errors, but not false negative ones. In the first case when result of checking is true we can not be sure, that the record is modified. Then we have to search the differential file with two goals. The first one is checking of correctness of bloom filter's answer and the second one, if answer is correct, is to find out all changes which are made on the original database.

## 3.2 Analysis of differential file access problem

One of instances of above described problem is under consideration in work [Gre82]. There the author presented results of empirical approach to selecting bloom filter parameteres in student database at Indian University.

During experiment 100 000 transactions on database were made. 30 000 unique key values were accessed and 7 000 unique values were updated, using these transactions. The author was interested in how the error rate depended on the number of hash functions and the filter size. This dependency was demonstrated by We see that the error rate decreases when the memory allocated for the bloom filter increases. This fact is not surprising. More interesting behaviour of error rate is observed when error rate is considered depending on the number of hash functions with the fixed bloom filter size.

We observe that on the first rows error rate becomes larger when the number of hash functions is increased. On the last rows more complicated behaviour of error rate is observed.

The work [Mul83] joins to [Gre82] and tries to analyze the above mentioned behaviour of the error rate from an analytical point of view. The main goal of this research was to derive mathematical formula for the error rate and compare theoretical and simulation approaches.

The probability of error rate is given by

$$ER = (1 - \alpha)(1 - (1 - 1/N)^{mk})^m$$

where

- $N$: The number of bits in filter.
- $m$: The number of transformation made.
- $k$: The number of different records modified — that is, the number of differential file records.
- $\alpha$: The fraction of changed records in the file.

The expected theoretical results are:

| | *# of hash functions* | |
|---|---|---|
| *Filter size (bytes)* | 4 | 6 |
| 24,576 | 0.214 | 0.302 |
| 32,768 | 0.109 | 0.142 |
| 49,152 | 0.036 | 0.036 |
| 65,536 | 0.014 | 0.011 |

We see that the theoretical values differ from the simulation ones. The reason is that the equation for the error rate was derived under the assumption that during transactions all records to be accessed are equally probable. However as mentioned in [Gre82] this assumption is invalid for the database at Indian University.

# 4 Conclusions

In this paper we have considered different types of hashing procedure. We started with conventional hash-coding method then explored more efficient technique which is called bloom filters. The Spectral bloom filters which

are generalizations of regular bloom filters are also covered. We ended with considering one application of bloom filters and showed how one can solve the task of choosing optimal, in some sense, parameteres of these filters. Two approaches were under our attention.

In this paper a lot of material is not covered. One interesting application arises in networking, where bloom filters help to organize storing information in distributed systems.

In spite of bloom filters were introduced by Burton Bloom in early 70's, They nowdays find new applications and further generalizations.

# References


[Blo70] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[CM03] S. Cohen and Y. Matias. Spectral bloom filters. *SIGMOID 2003*, pages 241–252, 2003.

[FBCA] L. Fan, A. Broder, P. Cao, and J. Almeida. Summary cache: a scalable wide-area web cache sharing protocol.

[Gre82] L. Gremillion. Designing a bloom filter for differential file access. *Communications of the ACM*, 25(9):600–604, 1982.

[Mit02] M. Mitzemacher. Compressed bloom filters. *IEEE/ACM transaction on networking*, 10(5):604–612, 2002.

[Mul83] J. Mullin. A second look at bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[Ram89] M. Ramakrishna. Pratical performance of bloom filters and parallel free-text searching. *Communications of the ACM*, 32(10):1237–1239, 1989.