Computing and Informatics, Vol. 37, 2018, 1313–1338, doi: 10.4149/cai_2018_6_1313

FINGERPRINT BASED DUPLICATE DETECTION IN STREAMED DATA

Amritpal Singh, Shalini Batra

Department of Computer Science and Engineering
Thapar University
Patiala, Punjab, India
e-mail: amritpal.singh203@gmail.com, sbatra@thapar.edu

Abstract. In computing, duplicate data detection refers to identifying duplicate copies of repeating data. Identifying duplicate data items in streamed data and eliminating them before storing, is a complex job. This paper proposes a novel data structure for duplicate detection using a variant of stable Bloom filter named as FingerPrint Stable Bloom Filter (FP-SBF). The proposed approach uses counting Bloom filter with fingerprint bits along with an optimization mechanism for duplicate detection. FP-SBF uses d-left hashing which reduces the computational time and decreases the false positives as well as false negatives. FP-SBF can process unbounded data in single pass, using k hash functions, and successfully differentiate between duplicate and distinct elements in $O(k+1)$ time, independent of the size of incoming data. The performance of FP-SBF has been compared with various Bloom Filters used for stream data duplication detection and it has been theoretically and experimentally proved that the proposed approach efficiently detects the duplicates in streaming data with less memory requirements.

Keywords: Duplicate detection, stable Bloom filter, d-left hashing, FingerPrint bits, streaming data

1 INTRODUCTION

With the exponential increase in data generation resources, paradigm of analytics has changed from static to dynamic processing especially in data mining applications. Advancements in application areas of IoT [4] and cloud computing [3] have shifted the focus of big data analytics towards streaming data analytics. In appli-

cations like network monitoring, sensor networks, data management in web applications, etc. [11], data is generated in various forms which include IP of system in network applications, URL of web page visited, sensor readings, unique user name used by user in social networking sites, etc. [26].

On-line monitoring of data streams and eliminating duplicates is an important issue in stream analytics. For redundant data elimination, primary focus should be on differentiating between duplicate and distinct element in the data stream. Detecting duplicates in streams becomes more difficult because of unbounded nature of data coming at high rate and necessity of processing data in one pass by using limited amount of memory [7].

Conventional databases and traditional data mining techniques are efficient for stored data analytics but for in-streamed data, where data is ariving continously, it is not feasible to store the data into a database and then perform mining operations since all such applications demand time bound query output.

In applications where efficiency is more important than accuracy, use of probabilistic approaches and approximation algorithms can serve as a key ingredient in data processing. Probabilistic methodologies provide quick answers with an allowable error rate compared to deterministic approaches that give exact matches, and which are slow and memory consuming [20].

1.1 Standard Bloom Filter

Bloom Filter (BF) [8], a space efficient probabilistic data structure, is used to represent a set S of n elements from a universe U. It consists of an array of m bits, denoted by $BF[1, 2, \ldots, m]$, initially all set to 0. To describe the elements in the set, the filter uses k independent hash functions $h_1, h_2, \ldots, h_k$ with their value ranging between 1 to m; assuming that these hash functions independently map each element in the universe to a random number uniformly over the defined range. For each element $x \in S$; the bits $BF[h_i(x)]$ are set to 1 s.t. $1 < i < k$. Given an item y, its membership is checked by examining the BF whether the bits at positions $h_1(y)$, $h_2(y)$, $\ldots$, $h_k(y)$ are set to 1. If all $h_i(y)$ $(1 < i < k)$ are set to 1, then y is considered to be part of S, otherwise y is definitely not a member of S. The accuracy of a Bloom filter, (f_p) depends on the filter size m, the number of hash functions k, and the number of elements n. User can predefine false positive error according to application's requirement.

$$f_p = (1 - e^{-kn/m})^k. \tag{1}$$

1.2 Application Domains

Some real time problems related to different domains are explored in this section, where duplicate detection is required to perform analysis on the streaming data.

Duplicate item identification is a common problem faced by social networking sites like Twitter, Facebook, Instagram, etc. where multiple copies of same event

(tweet or post) are generated for same input by multiple users, which keep on appearing continuously from different sources on the user's screen. In such scenarios, the primary need is to identify duplicate events and group them together to improve the user's experience [21]. Another important event related to duplicate detection is URL crawling. Search engines regularly crawl the Web to enlarge their collections of Web pages. While scanning a URL, search engine's task is to identify the new web pages and add them to its repository. So, the basic task in this scenario is comparing each scanned URL with all existing URLs in its database to identify duplicate URLs [18]. In network monitoring applications, selecting distinct IP addresses is a task associated with duplicate detection. In networks, a particular server is checked for unique hits. This analytics facilitates in understanding the pattern of traffic which helps in efficient allocation of network resources [13]. Web advertising is easy and most effective way to publicize a product where advertisers pay web site publishers for number of clicks on their advertisements. Fake clicks may be generated (by using scripts) to increase the profit of the publisher. To detect the duplicate users in clicks is thus associated with duplicate detection task [22].

We propose a novel duplicate detection technique using a variant of stable Bloom filter [13] named as FingerPrint Stable Bloom Filter (FP-SBF). The proposed approach uses stable Bloom filter with fingerprint bits and optimization mechanism which reduces the computational time and decreases the false positives as well as false negatives using d-left hashing. Optimized deletion mechanism is used to evict the data from Bloom filter to make more space for incoming data. It uses constant space irrespective of the size of incoming data and accommodates more number of elements before reaching the saturation stage. Based on above mentioned improvements in existing approach, the proposed Bloom filter outperforms the stable Bloom filter in detecting the duplicates in streaming data, both theoretically and experimentally.

The plan of this paper is as follows: Section 2 provides literature study on Bloom filter and its variants. Section 3 states the duplicate detection problem and challenges associated with it. In Section 4, proposed approach is discussed in detail. Section 5 provides experimental results on both real datasets and synthetic data, comparing existing approaches with the proposed approach. Section 6 discuses use case scenarios for proposed approach. Finally, Section 7 concludes the paper with possible future extensions in this area.

2 RELATED WORK

2.1 Duplicate Detection

Duplicate detection is the task of detecting unique entries in unbounded data streams. Duplicate elimination is a crucial intermediate step in data processing and analytics of the incoming streams. The problem of finding approximate duplicate items has been studied in the contexts of data management and Web applications. Data streams are generally unbounded and traditional DBMS approaches of accom-

modating whole stream in the memory leads to very high memory utilization. We need to provide results for the query in minimum time with less computational cost and minimum memory requirements.

There are many solutions for the duplicate detection problem [21] but our main focus is on the solutions based on usage of Bloom filters for efficient detection of duplicate datasets in streaming data. In window based approaches, for every new input arriving from the stream, an old entry is evicted by adjusting the size of window. Recent work done in window based framework for duplicate detection using Bloom filter is by Metwally et al. [22]. In their work three type of window based approaches are defined: landmark window, sliding window and jumping window. In landmark window approach, incoming data is processed as disjoint portions of the stream, which are separated by landmarks. Landmarks can be defined either in terms of time, e.g. on a daily or weekly basis, or in terms of the number of elements observed. In landmark window approach, whole data from Bloom filter is deleted upon reaching the new landmark. Since individual element deletion is not performed in this approach, a simple Bloom filter is used. In sliding window approach size of window is not fixed, it grows as the incoming data from stream grows. The element deletion is also not allowed in this approach so simple Bloom filter is used in implementation. The full size of window is maintained to query old data. Querying process in sliding window has more computational cost as compared to landmark window. Landmark window requires less space as compared to sliding window but there are chances of some missed values in case of landmark window because upon reaching new landmark previous data is deleted. The jumping window is based on idea of dividing the individual element window into smaller sub windows and usage of counting Bloom filter to accommodate more number of elements. The latest sub window active for insertion is referred as jumping window or current window. As the latest sub-window crosses its threshold, jumping window is moved to next sub-window and oldest sub-window is deleted. So these window based approaches either need dynamic size that leads to extra computation and expensive query process or they remove a large chunk of data together which may lead to the higher false negatives [27].

Another solution is use of buffering and caching methods, used in many database, network applications, URL caching and web crawling. Some Bloom based buffering techniques include double buffering and A^2 buffering. In double buffering [10] two different buffers of size $\frac{m}{2}$ are maintained. As one filter crosses its threshold, insertion starts in second and when second filter is full, i.e., this filter reaches its threshold, then whole data from the first one is drained and insertion is again started in the first filter. For query process, first active filter, i.e., current filter in which insertion is being done recently, is checked and if query returns false then previous filter is scanned. In A^2 buffering [28] similar approach is used, in this, when active filter reaches its 50 % capacity, the insertion is started in both filters, and when active filter is full, its data is evicted by resetting it to zero. In buffering approaches element wise deletion is not allowed so a simple Bloom filter is used. Buffering approaches store data for short time with large redundancy. Handling data streams

with buffering leads to wastage of memory; further it is computationally costlier to check the threshold of Bloom filter at every step.

Recently variant of Bloom filter for duplicate detection in stream named as Stable Bloom Filter (SBF) has been proposed by Deng and Rafiei [13]. Counting Bloom filter with c bits in counter is used as the base of its implementation. Before inserting any element in the stream, it is checked that whether the incoming element is queried earlier in the stream or not. If query returns TRUE, i.e., values of counters corresponding to hash indexes are high, then element is marked as duplicate and element is not inserted, but if query returns FALSE indicating that the element is observed for the first time in the stream, element is added. Before insertion begins, SBF makes space for incoming elements by randomly decrementing the values of counter by one in each iteration. In insertion process, k hash functions are computed and correspondingly hash value counters are set to Max, where $Max = 2^c - 1$. The main advantage of this approach is that it provides query results on streaming data in $O(k)$, independent of the size of incoming data, using constant memory.

Another technique proposed for duplicate detection is Reservoir Sampling based Bloom Filter (RSBF), a hybrid of reservoir sampling and Bloom filter [14]. It uses k Bloom filters, each of size s bits. Insertion of an element is performed after query process. One hash function of each Bloom filter is reserved. If hash index associated with each Bloom filter is found HIGH then element is considered as duplicate and insertion is not performed. Else, insertion is performed by setting corresponding k bits HIGH in k Bloom filters. To accommodate streaming data in fixed size Bloom filters, random deletion is done by resetting k randomly selected bits to LOW. RSBF uses same amount of memory as SBF, has fast convergence rate and shows significant improvement in false negatives but it does not provide a significant improvement in false positives.

Streaming Quotient Filter (SQF) is another duplicate detection algorithm based on probabilistic data structures [15]. In proposed model, Dutta et al. use Quotient filter for approximate membership query instead of Bloom filter. SQF maintains a hash table and bucket associated with each element in hash table. Hashing is done at two levels: first element is mapped to hash table and then corresponding to hash table entry, hashing is done in bucket by using quotienting technique. SQF performs efficiently in detecting false positives and false negatives but the preprocessing time for converting data into binary form for quotienting and hashing at two levels increases the computational time many folds. Another drawback associated with SQF is that because of clustering, operations like insertion and searching are difficult to perform in parallel.

2.2 Counting Bloom Filter (CBF) and Its Variants

Counting Bloom filter, introduced by Fan et al. [17], uses a counter of c bits where range of counter is $\{1, 2^c - 1\}$. Based upon the hash indexes computed, i.e. $\overset{k}{_{i=1}} h_i(x) = \overset{k}{_{i=1}} H_i(x)$, insertion and deletion operations are performed on the counters.

Whenever an element is added or deleted from the CBF, the corresponding counters are incremented or decremented, respectively. CBF is primarily used to answer frequency queries. Although it can be efficiently used for applications where deletion operation is required, the memory overhead for CBF as compared to standard Bloom filter is significantly large and determining value of counter is a difficult process. Number of variants of CBF have been proposed to overcome such issues and maximize the storage of counting Bloom filter with minimum memory requirements and less false positive and false negative rates.

Some important variants of CBF are discussed in this section. In Spectral Bloom filter [12] value of smallest counter is increased when a new element is inserted and query process returns minimum count for an element from the counter selected. It is mainly used for storing the multi sets data and supports frequency query. Another improvement in CBF was done by improving hashing technique and called d-left counting Bloom filter [9], in which CBF with d-left hashing was used to calculate index values of hash functions by dividing Bloom filter into sub tables. Use of this efficient hashing in CBF leads to less number of collisions. It is used for element lookups and fingerprint matching applications. Deletable Bloom filter [24], another variant of CBF, optimizes the process of deletion by using probabilistic approach for element removal. It keeps the record of regions with high collisions, i.e., the regions where probability of deletion is quite high. The main aim of this variant is to minimize false negatives. It finds application in source routing to avoid loops, middle-box services like load balancer, firewalls, etc.

Recent variants of CBF include Variable Increment Counting Bloom Filter (VI-CBF) and FingerPrint based Counting Bloom Filter (FP-CBF). In VI-CBF [25] two different sets of hash functions are maintained, first set of hash functions, i.e. $H_{i=1}^{k}$, decide the indexes on which increment is performed, and another set of hash functions, i.e. $G_{i=1}^{k}$, decide the value from a set $D_L = \{L, L+1, \ldots, 2^L - 1\}$ by which increment is performed on the selected indexes. Same process is followed in deletion operation. It decreases the false positives by a huge factor as compared to standard CBF. Some limitations of VI-CBF are that implementation is more complex, calculations for two hash functions lead to extra computational cost and more bits are required to avoid overflow of counters.

Pontarekki et al. proposed FP-CBF [23], where each counter has some extra bits denoted as fingerprint bits. The main idea behind the use of these bits is to provide a second level check in querying operation and reduce false positive and false negatives. Here each index of d bits is divided into counter bits (c) and fingerprint bits ($f = d - c$). In insertion process, indexes for update are computed (based on the k hash functions used) and corresponding to these indexes all counters are incremented by one. A separate hash function H_{fp} is used to update the fingerprint cell of all selected indexes with XOR operation. In querying process, values of counters are checked and if all values are high then second level check is performed by matching H_{fp} value of queried element with fingerprint cells. If values of H_{fp} in all fingerprint cells match, query returns TRUE value. FP-CBF provides more accurate results with minimum computational effort.

2.3 d-Left Hashing

Hashing is another important aspect to determine the accuracy of a Bloom filter. Variants of CBF use advance hashing techniques like double hashing [1], multiple hashing [6], perfect hashing scheme [5], etc. Recently cuckoo hashing [16] has been used in CBF which has shown drastic decrease in error rates. Major concern in cuckoo hashing is that computational cost associated with swapping of elements is quite high and pre-processing cost in insertion of each element is also quite high. Another important hashing, d-left hashing refereed as perfect hashing technique, can be used to reduce the collisions by great extent. Further, it is easy to implement and requires less computational and pre-processing cost.

d-left hashing is a minimal perfect hashing approach, where a hash table of size M is divided into d sub tables of size m_d, where $m_d = \frac{M}{d}$. Each element in sub table is referred as bucket B_i and b denotes the maximum capacity of bucket. During insertion operation, k hash functions are used to compute the hash indexes and select the buckets. New item is placed in the bucket which is least loaded. If there is tie between two buckets then bucket on left side is selected to perform insertion operation.

For an element $u_i \in U$, hash functions are calculated as:

$$\forall_{i=1}^{k} \left(\kappa_i(u_i) = \ell_1(u_i) + i \times \ell_2(u_i) \ mod \ m\right). \tag{2}$$

Each hash function $\kappa_i(.)$ selects i^{th} bucket from a sub array. Double hashing is used for hash function generation, where two independent hash functions $\ell_1(x)$ and $\ell_2(x)$ are used to generate k hash functions such that $\forall i | i < k$. To get best results of this scheme, m should be prime, so size of array and number of buckets should be choosen in such a way that $\frac{m}{d}$ returns a prime number. This technique leads to less inter-hash function collision; further, usage of only two hash functions to generate all k hash functions decreases the computational overhead. Theoretical and experimental evidences prove that it helps in reduction of collision as a huge factor [9, 19].

3 PROBLEM STATEMENT

The main emphasis of the paper is to detect duplicate data sets in the data stream. The problem can be summarized as: Given an input stream $S = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ with N elements; where $N \to \infty$, i.e. unbounded stream of data, identify whether x_i appears in S or not in a given space M, where $(M << N)$. Challenges associated with duplicate detection include:

1. Preprocessing effort for each element should be minimum for fast results.
2. Eviction of stale data is necessary as the memory size of the filter is fixed.
3. Response time for query should be minimum and independent of the size of data.

4 PROPOSED WORK

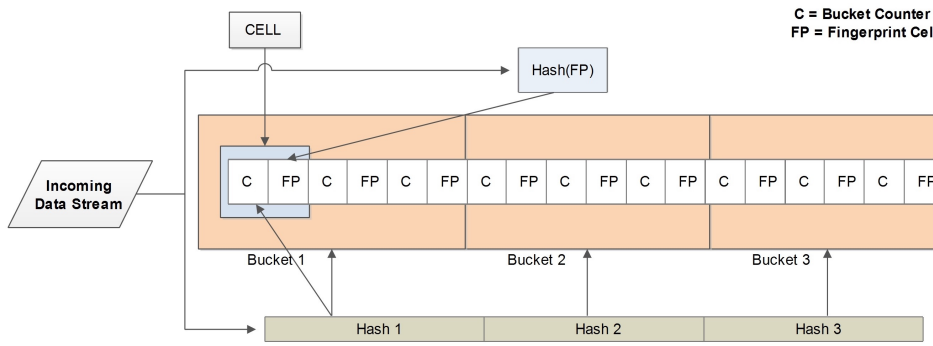

Figure 1. Structure of FP-SBF

To solve the duplicate detection task efficiently using minimum memory some approximation based processing techniques need to be applied on streaming data. A variant of Stable Bloom filter (SBF) named as FingerPrint Stable Bloom filter (FP-SBF) is proposed to detect duplicates in streaming data. FP-SBF is an array of indexing $[1, \ldots, m]$ of size M where each element of array is represented by d bits (i.e. $M = m \times d$). d bits of each element are further divided into two parts: c bits for buckets and f bits for fingerprint as indicated in Figure 5. Bucket bits (c bits) are used as counter, where range of counter values is $\Re_c \leftarrow \{1, \{Max = (2^c - 1)\}\}$ and $f = (d - c)$ bits are fingerprint bits, also called FingerPrint Cell (FPC). FP-SBF uses d-left hashing with $(k + 1)$ hash functions, where k hash functions are used to select the appropriate index from m elements and update bucket's counter to Max and one hash function H_{fp} is used to update fingerprint cell of each selected index. Since available memory space is fixed, old data should be evicted to make room for new data. Parameter ξ called Eviction Rate (ER) is used to control the eviction of stored data in Bloom filter.

The task of detecting duplicate from the streams using FP-SBF consists of following steps (Algorithm 1):

1. **Detection:** Query the existing data to find whether current data element x_i exists in the filter or not.
2. **Deletion:** Remove the data randomly by decrementing the values of buckets to make spaces for new incoming elements.
3. **Insertion:** Insert the data in the filter by updating the corresponding bucket and fingerprint cells, if the element is not present in the array.

Algorithm 1 Duplicate Detection in Streams (DDS) using FP-SBF

```
1: procedure DDS(FP-SBF[], S, H_k, H_fp) ▷ Check whether x_i element is present in the stream or not
2:     for ∀x_i ∈ S do
3:         if Detection(x_i) == TRUE then ▷ Duplicate detected
4:             Element appeared previously in the filter, no insertion required
5:         else
6:             if Random(ξ) > Threshold then ▷ Check for threshold and then insertion is performed
7:                 Delete()
8:             end if
9:             Insert(x_i)
10:        end if
11:    end for
12: end procedure
```

4.1 Detection

To detect an element x_i in FP-SBF, $k+1$ hash functions H_1^k and H_{fp} are used. Corresponding to k hash functions, indexes are generated, i.e. $h_1^k \leftarrow H_1^k$, and following checks are performed (Algorithm 2):

1. If any of the bucket corresponding to h_1^k is set to zero, duplicate detection mechanism returns false, i.e. $x_i \notin (x_1, \ldots, x_{i-1})$.
2. If all buckets are non-zero, $H_{fp}(x_i)$ is computed for x_i and all fingerprint cells of indexes h_1^k are checked. If $H_{fp}(x_i)$ is not found in any FPC, duplicate detection mechanism returns FALSE, i.e. $x_i \in (x_1, \ldots, x_{i-1})$.
3. If step 1. and 2. (mentioned above) are false then x_i is duplicate, i.e., it is previously seen in the stream $(x_1, \ldots, x_{i-1})$, hence no need to perform further operations.

Deletion and insertion operations are invoked when detection process fails, i.e., element is not found in the stored stream. False positives may occur in the detection due to hash function collision of two different elements, i.e., bucket set high by x_i returns detection results as TRUE, resulting in identification of a distinct element as duplicate element.

A false negative in duplicate detection on streamed data occurs when a duplicate element (x_i) is wrongly reported as a distinct element. Eviction process in FP-SBF leads to decrement the value of buckets to zero, i.e., for an element which is present in stream, the value of bucket is decremented to zero and this element is regarded as a distinct element although it is a duplicate element.

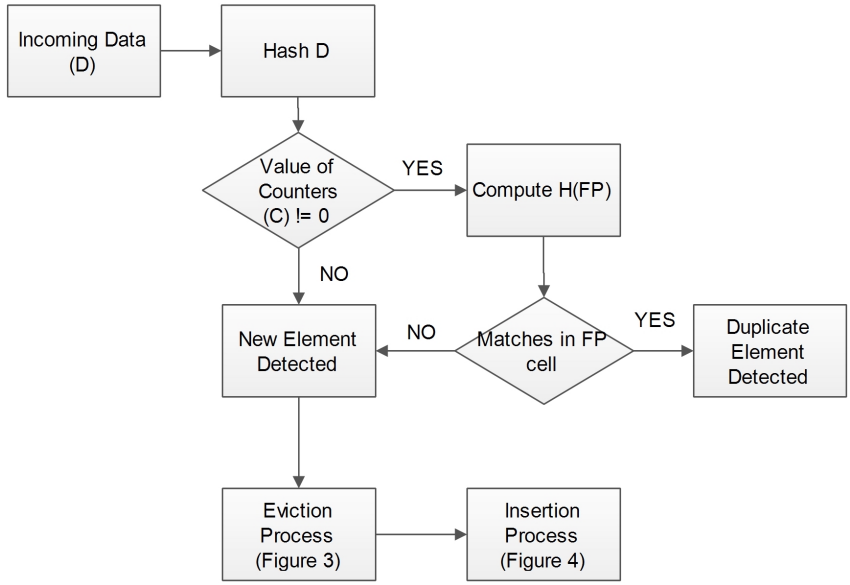

Figure 2. Flowchart of detection process in FP-SBF

4.2 Deletion or Eviction of Data

To accommodate unbound data in constant memory, it is necessary to evict the data at regular intervals. FP-SBF applies an optimized deletion approach which evicts the data quickly and decreases false positives (Algorithm 3).

Algorithm 2 Detection

procedure DETECTION($FP\text{-}SBF[], x_j, H_k, H_{fp}$)
  $\forall i|(1 < i < k)$ Calculate hash $h_{i=1}^{k}(x_j) \leftarrow H_{i=1}^{k}(x_j)$
  if $(\forall i|(1 < i < k)), [h_{i=1}^{k}(x_j) > 0]$ **then**
    $h_{fp}(x_j) \leftarrow H_{fp}(x_j)$
    if $(\forall i|(1 < i < k)), [FPC_i = h_{fp}(x_j)]$ **then**
      Return *TRUE*
    else
      Return *FALSE*
    end if
  else
    Return *FALSE*
  end if
end procedure

k indexes are selected randomly with probability p and decremented by a value z, s.t. $z \in (1, 2, \ldots, 2^c - 1)$. Deletion process is invoked after i iterations where i is selcted randomly. This process is controlled by Evicting Rate (ER) parameter $\xi()$ $\Re_\xi \rightarrow \{0, 1\}$). Steps followed to accomodate incoming data are:

1. In every iteration, check if $Random(\xi)$ returns a value more then defined threshold, where $Random(\xi)$ is a function which generates random values as per the value of ξ provided to it. If $Random(\xi)$ is greater than defined threshold then deletion is performed else steps 2. to 4. are skipped
2. If step 1. is false, i.e., threshold is not reached, select k index, i.e., $h_1^k(D)$ for deletion where probability of any cell being selected is $(\frac{p}{m})$.
3. Select a random value of z for each selected index $h_1^k(D)$ from a given range $\Re_z \rightarrow \{1, 2^c - 1\}$.
4. For each index selected in step 3. decrement the value of bucket by z.

Deletion frequency can be increased by increasing value of ξ. Fingerprint cell of selected indexes remains unaffected in the deletion process.

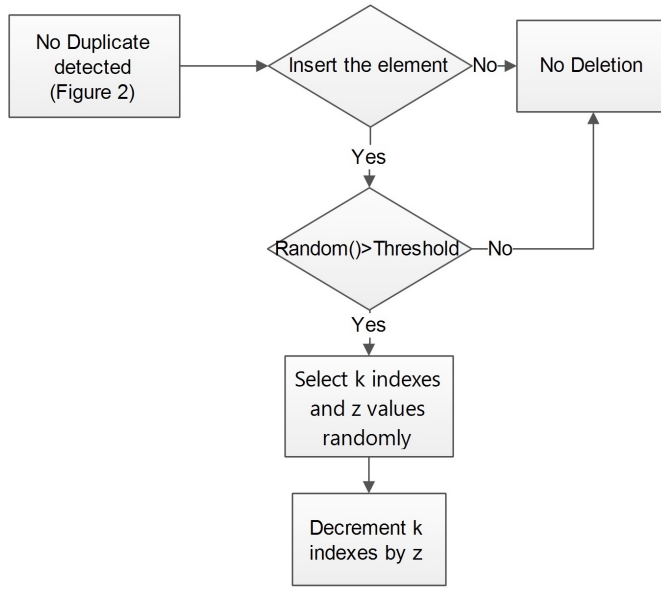

Figure 3. Flowchart of eviction process in FP-SBF

4.3 Insertion

To insert an element x_i, $k + 1$ hash functions are used. Steps followed for insertion of an element are (Algorithm 4):

Algorithm 3 Deletion Process

1: **procedure** DELETE($FP\text{-}SBF[], p_d, \Re_z$)
2: `Select` k `cells from m with probability` p
3: $L_{i=1}^{k}$ are the selected cells for decrement operation.
4: `Select a value to be decremented from each bucket`
5: $z \leftarrow Random(\Re_z)$
6: $(\forall i | (1 < i < k))\ Bucket[L_i] = (Bucket[L_i] - z)$
7: **end procedure**

1. Hashing is done to get the indexes, i.e. $h_1^k(x_i) \leftarrow H_1^k(x_i)$.
2. Buckets are updated:
$$Set(h_1^k(x_i)) = Max$$
where $Max = (2^c - 1)$.
3. Fingerprint cell is updated:
$$h_{fp}(x_i) \leftarrow H_{fp}(x_i).$$
$h_{fp}(x_i)$ is used to update the corresponding FPC of selected indexes. $FPC_i \leftarrow (FPC_i)$ XOR $(h_{fp}(x_i))$.

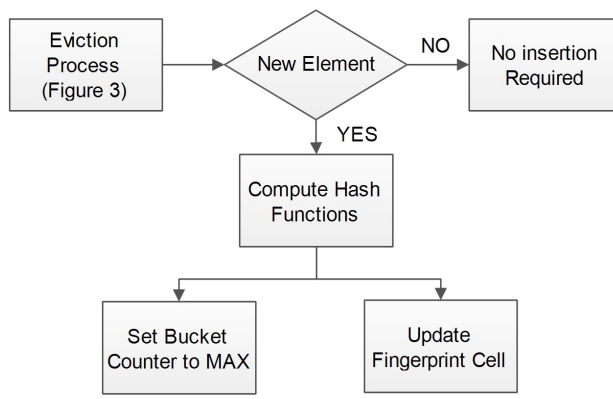

Figure 4. Flowchart of insertion in FP-SBF

4.4 Stable Property (SP)

Stable property assures that after ℓ iterations, the fraction of zeros in SBF/FP-SBF will be fixed, i.e., they will not depend on any parameter. In FP-SBF number of iterations ℓ required to achieve SP is depended on eviction rate ξ. It has been proved

Algorithm 4 Insertion in FP-SBF

1: **procedure** INSERT($FP\text{-}SBF[], x_j, H_k, H_{fp}$)
2: $(\forall i|(1 < i < k))$ Calculate hash $h_{i=1}^{k}(x_j) \leftarrow H_{i=1}^{k}(x_j)$
3: $(\forall i|(1 < i < k))Set(Bucket[h_i] \leftarrow Max)$
4: $h_{fp}(x_i) \leftarrow h_{fp}(x_j)$
5: $(\forall i|(1 < i < k))$ `do`
6: $FPC_i \leftarrow (FPC_i)$ XOR $h_{fp}(x_j)$
7: **end procedure**

theoretically and experimentally that SP plays an important role in determining false positive rate.

Theorem 1. In FP-SBF with m cells, each cell is updated to Max with a probability $p_i = (\frac{k}{m})$ and each cell is decremented by a value $z \in (1, \ldots, 2^c - 1)$ with a probability $p_d = (\frac{p}{m})$. The probability that cells become zero after N iterations as $N \to \infty$ is constant. i.e.

$$\lim_{N\to\infty} \sum_{i=1}^{N} Pr(ASBF_i = 0) \Rightarrow Constant$$

where $ASBF_N$ is value of cell at the end of N iterations.

Proof. For each element of stream, three possible operations are detection, deletion and insertion. Only deletion and insertion will effect the values of buckets. After N operations, a bucket can be set to $Max \leq N$ times and decremented with certain value $< N$ times. Let p_i be probability of a bucket to be selected for insertion given by $p_i = \frac{k}{m}$; p_d be probability of a bucket to be selected for deletion, i.e. $p_d = \frac{p}{m}$, and ξ be Evicting rate for FP-SBF. A_l denotes that no insertion has been performed in the bucket in recent l iterations ($l < N$); probability of event A_l is given by:

$$Pr(A_l) = (1 - p_i)^l p_i. \tag{3}$$

A_N denotes that no insertion is performed in any bucket in N iterations, its probability is:

$$Pr(A_N) = (1 - p_i)^N. \tag{4}$$

Since no insertion operation is performed in a particular bucket for N iterations, probability of having value zero in that bucket following event A_N is:

$$Pr(ASBF_N = 0|A_N) = 1. \tag{5}$$

P_0, probability that after deletion operation value of bucket is zero is:

$$P_0 = \xi \times p_d \times Pr(P_i = 0|(FP - SBF[i] = x)) \tag{6}$$

where value of $Pr(P_i = 0|(FP - SBF[i] = x))$ is dependent on the current value in the bucket and z value selected from Deletion Value Set $(1, 2^c - 1)$ is given by:

$$Pr(P_i = 0|(FP - SBF[i] = x)) = \frac{\Delta[(z \geq x)\&(z \in DVS)]}{\Delta(z \in DVS)}. \tag{7}$$

$\Delta()$ is function which counts the number of values of $z \in DVS$, to calculate the sample space and favorable events for deletion.

When a bucket follows event A_l, the probability that after N iterations where $N > l$ the decrement operation will reset the bucket value to zero is given by:

$$Pr(ASBF_N = 0|A_l) = \sum_{j=Max}^{l} \binom{l}{j} P_0^j (1 - P_0)^{l-j}. \tag{8}$$

Thus, for a random element, probability $Pr(ASBF_N = 0)$ that the bucket is zero after N iterations is given by:

$$\begin{aligned} Pr(ASBF_N = 0) = &\sum_{l=Max}^{N-1} [Pr(ASBF_N = 0|A_l) Pr(A_l)] \\ &+ Pr(ASBF_N = 0|A_N) Pr(A_N). \end{aligned} \tag{9}$$

In FP-SBF, if bucket is not set to *Max* in l iterations, more then one operations are required to decrement its value to zero, i.e., for $l \leq Max$ cell can be decreased to zero and for $l = N$ A_N event occurs. So from Equation (9) it is proved that $\lim_{N\to\infty} Pr(ASBF_N = 0)$. Hence, proposed FP-SBF follows stable property principle of standard SBF efficiently and with less computational complexity. □

Definition 1 (Convergence Rate (CR))**.** From the stable point property, each bucket has fixed probability of being set to *Max* and constant probability of being reset to zero after certain iterations. P_0, probability that a cell becomes zero is same for all buckets. Thus, expected number of zeros in FP-SBF converges exponentially and Convergence Rate (CR) can be derived using Equation (9):

$$CR = Pr(ASBF_N = 0) - Pr(ASBF_{N-1} = 0). \tag{10}$$

Definition 2 (Stable point)**.** It is the expected fraction of zeros in SBF, when data is unbounded. Using Theorem 1 probability of values in bucket being set to zero is constant, i.e.

$$\begin{aligned} Pr(ASBF_N = 0) = &\sum_{l=Max}^{N-1} [Pr(ASBF_N = 0|A_l) Pr(A_l)] \\ &+ Pr(ASBF_N = 0|A_N) Pr(A_N). \end{aligned} \tag{11}$$

Let H be the number of cells that are set to Max and L be the number of cells that are decremented to zero after certain number of iterations (from Theorem 1), then expected number of zeros in FP-SBF, i.e. stable point property of FP-SBF, is given by (Z_{SP}):

$$Z_{SP} = \left(\frac{1}{1 + \frac{1}{L(\frac{1}{H} - \frac{1}{m})}} \right)^{Max}. \tag{12}$$

The optimization in the decrement operation will help to achieve stable point in FP-SBF in less number of iterations with less computational complexity, which will further help to detect parameters like false positives and negatives at earlier stage.

4.5 False Positives (FPs) Analysis

Theorem 2. When SBF reaches a stable point, the FPs is given by:

$$FP = F_{FP\text{-}SBF} - RF_{FPC}$$

where $F_{FP\text{-}SBF}$ is error due to collision in buckets of FP-SBF and RF_{FPC} denotes the reduction factor in FPs due to fingerprint cells.

Proof. False positives are generated when distinct element in the stream is wrongly reported as duplicate. FPs are directly dependent on number of zeros at particular point in FP-SBF, i.e., when stable point is reached FPs can be estimated. Change in fingerprint cell of each bucket helps to improve FPs by providing a second level check. In case of collision in the buckets, $F_{FP\text{-}SBF}$ is dependent on the number zeros in the filter. More the number of zeros, less the collisions and less FPs. When stable point is reached, number of zeros becomes constant and after certain iterations, H buckets are set to Max and L are decremented to zero; $F_{FP\text{-}SBF}$ is given:

$$F_{FP\text{-}SBF} = \left(\frac{1}{1 + \frac{1}{L(\frac{1}{H} - \frac{1}{m})}} \right)^{Max}. \tag{13}$$

RF_{FPC} is the reduction factor in detection procedure due to fingerprint cells when all buckets corresponding to hash indexes are high. It acts like a second level check on these f bits. Since the possibility of FPs is one out of 2^f cases in fingerprint cells, it helps in reducing the FPs by a fixed factor. Assuming that insertion operation is performed I times on k indexes (equal to number of hash functions) and fingerprint cells are updated by XOR operation; RF_{FPC}, the probability of not having collision in f bits in m FPC's is:

$$RF_{FPC} = \left[\left(\frac{2^f - 1}{2^f} \right) \left(\frac{1}{m} \right) \binom{I}{1} \left(1 - \frac{1}{f} \right)^{I-1} \right]^k. \tag{14}$$

From Equations (13) and (14), FPs in FP-SBF are given by:

$$FPs = \left(\frac{1}{1+\frac{1}{L(\frac{1}{H}-\frac{1}{m})}}\right)^{Max} - \left[\left(\frac{2^f-1}{2^f}\right)\left(\frac{1}{m}\right)\binom{I}{1}\left(1-\frac{1}{f}\right)^{I-1}\right]^k. \quad (15)$$

Thus, the use of fingerprint cell helps to reduce the FPs by a significant factor and improve the accuracy of the duplicate detection system. □

4.6 False Negative Rate (FNs) Analysis

Theorem 3. At stable point FNs for FP-SBF are given by:

$$FNs = FR_{FP\text{-}SBF} + E_{FPC}$$

where $F_{FP\text{-}SBF}$ is error due to deletion operation on buckets of FP-SBF and E_{FPC} denotes the error in fingerprint cell due to the collision of H_{fp} function.

Proof. A false negative in duplicate detection on streamed data occurs when a duplicate element (x_i) is wrongly reported as distinct element. This happens during decrement operation when some buckets associated with hashed indexes of x_i are decremented to zero, before appearing in the stream or there is a mismatch in fingerprint cells when all the buckets have high value corresponding to x_i.

Suppose x_i appears second time in the stream after δ iterations and $h_{j=1}^{j=k}(x_i)$ denotes the corresponding hash indexes and p_{ij} is probability that a particular cell C_j is set to Max. In δ iterations, some bucket from $h_{j=1}^{j=k}(x_i)$ are reduced to zero. The probability of error due to buckets resetting $(FR_{FP\text{-}SBF})$ is same as in Equation (9); given by:

$$FR_{FP\text{-}SBF} = 1 - \prod_{j=1}^{k}(1 - Pr(ASBF_\delta = 0)) \quad (16)$$

and probability that after δ iterations a particular bit of FP-SBF is zero, i.e. $Pr(ASBF_\delta = 0)$, is given by:

$$\begin{aligned} Pr(ASBF_\delta = 0) = \sum_{l=Max}^{\delta-1} [Pr(ASBF_N = 0|A_l)Pr(A_l)] \\ + Pr(ASBF_N = 0|A_N)Pr(A_N). \end{aligned} \quad (17)$$

So FN is function of δ and p_{ij} given by:

$$FR_{FP\text{-}SBF} = 1 - \prod_{j=1}^{k}(1 - Pr(\delta_j, p_{ij})). \quad (18)$$

E_{FPC}, chance of false negatives due to error in H_{fp} hash function in fingerprint cell is due to the collision in changing bits because of XOR operation in δ iterations. In m size array, selected k FPCs are checked and mismatch in any of them leads to failure in detection process, i.e., a duplicate is reported as distinct element. After f iterations, chances that erroneously reported elements are 1 out of 2^f for δ iterations using k hash function is given by:

$$E_{FPC} = \left[\left(\frac{1}{2^f}\right) \left(\frac{1}{m}\right) \binom{\delta}{1} \left(1 - \frac{1}{f}\right)^{\delta-1} \right]^k. \tag{19}$$

FNs for FP-SBF is given by:

$$FNs = \left(1 - \prod_{j=1}^{k} (1 - Pr(ASBF_\delta = 0))\right) + \left[\left(\frac{1}{2^f}\right) \left(\frac{1}{m}\right) \binom{\delta}{1} \left(1 - \frac{1}{f}\right)^{\delta-1} \right]^k. \tag{20}$$

Use of FPCs shows great improvement in decreasing the FPs of FP-SBF in all cases. □

Theorem 4. For given inputs k, Max, f, FPs and H_{fp}; processing each data item of the stream requires $O(k+1)$ time which is independent of the size of Bloom filter and the incoming data stream.

Proof. In duplicate detection, the primary goal is minimization of error rates while using constant space, and time complexity in the detection process should be independent of the nature and size of data stream, i.e., there should be constant processing time for each element. First step in duplicate detection is to check whether an element is seen previously in the stream or not. For this first k hash functions are computed for buckets and then one hash function is calculated for fingerprint cell. For given parameters k, Max, f, FPs with constant values, there is no effect of element and the detection process is also independent of the size of Bloom filter (m) (Algorithm 1).

From Algorithm 1 and analysis performed above it is concluded that processing time in duplicate detection is only dependent on number of hash functions, i.e. $O(k+1)$. □

5 OBSERVATIONS AND ANALYSIS

The theoretical analysis has been provided in previous sections for some important parameters, i.e. Max, H, L, m, ξ, false negatives (FN) and false positives (FP); where H is number of cells set to Max in insertion operation (i.e. equal to number of hash functions), L denotes the number of cells selected for the decrement operation and m is fixed amount of memory used for the Bloom filter.

False positives can be bounded according to user specified requirements. Since the false negatives in the fixed amount of memory are depended on deletion operation

they cannot be bounded in specified limits. Two parameters, desired false positive (FP) rate and size of the Bloom filter (m) are taken as input from user and other parameters like Max, H, L are selected in such a way that false negatives are minimal. The parameter ξ is also user defined and helps to control the frequency of deletion operation.

For user defined FPs, m and for constant values of Max and H; L is defined as:

$$L = \left(\frac{1}{\left(\frac{1}{\left(1 - FPs^{1/H}\right)^{1/Max}} - 1 \right) (1/H - 1/m)} \right). \tag{21}$$

Equation (21) helps to find value of L, i.e. number of cells selected for decrement operation. From the value of L (calculated in Equation (21)) p_d, the probability of selecting cells for decrement, and P_0, probability that after certain decrement operations value of cell is zero, can be derived.

Parameter H is equal to the number of hash functions used (k). $E(FN)$ denotes the expected number of false negatives in the stream. Optimal value of H should be selected to minimize the FNs. With $\tilde{N}$ as the number of false negatives in a stream of N elements, $E(FN)$ is:

$$E(FN) = \sum_{i=1}^{\tilde{N}} \left(Pr(FNR_i) \right), \tag{22}$$

$$E(FN) = \sum_{i=1}^{\tilde{N}} \left(\left(1 - \prod_{j=1}^{k} (1 - Pr(ASBF_\delta = 0)) \right) + \left[\left(\frac{1}{2^f} \right) \left(\frac{1}{m} \right) \binom{\delta}{1} \left(1 - \frac{1}{f} \right)^{\delta - 1} \right]^k \right). \tag{23}$$

The value of Max is dependent on size of input data and the number of hash functions used. Optimal value of Max can be derived from Equation (23) by minimizing FNs. For efficient memory utilization Max should be set $2^c - 1$. The remaining bits that are not allocated to counter are used as fingerprint bits. For fixed amount of cells when value of Max is increased, the effectiveness of fingerprint cell is reduced.

The optimization in deletion process is controlled by user defined parameter ξ and $Rand(\xi)$ function is used to set the frequency of deletion operation; larger the value of ξ more frequently the deletion operation is performed and vise versa.

All the experiments have been performed on i7-3612QM CPU @ 2.10 GHz with 8 GB of RAM. To maintain the uniformity in the results *CityHash* 64 bit library [2] is used to calculate two hash functions in double hashing. Data set of 100 k elements with 70 % distinct entires and 30 % duplicate entires has been generated using R-studio.

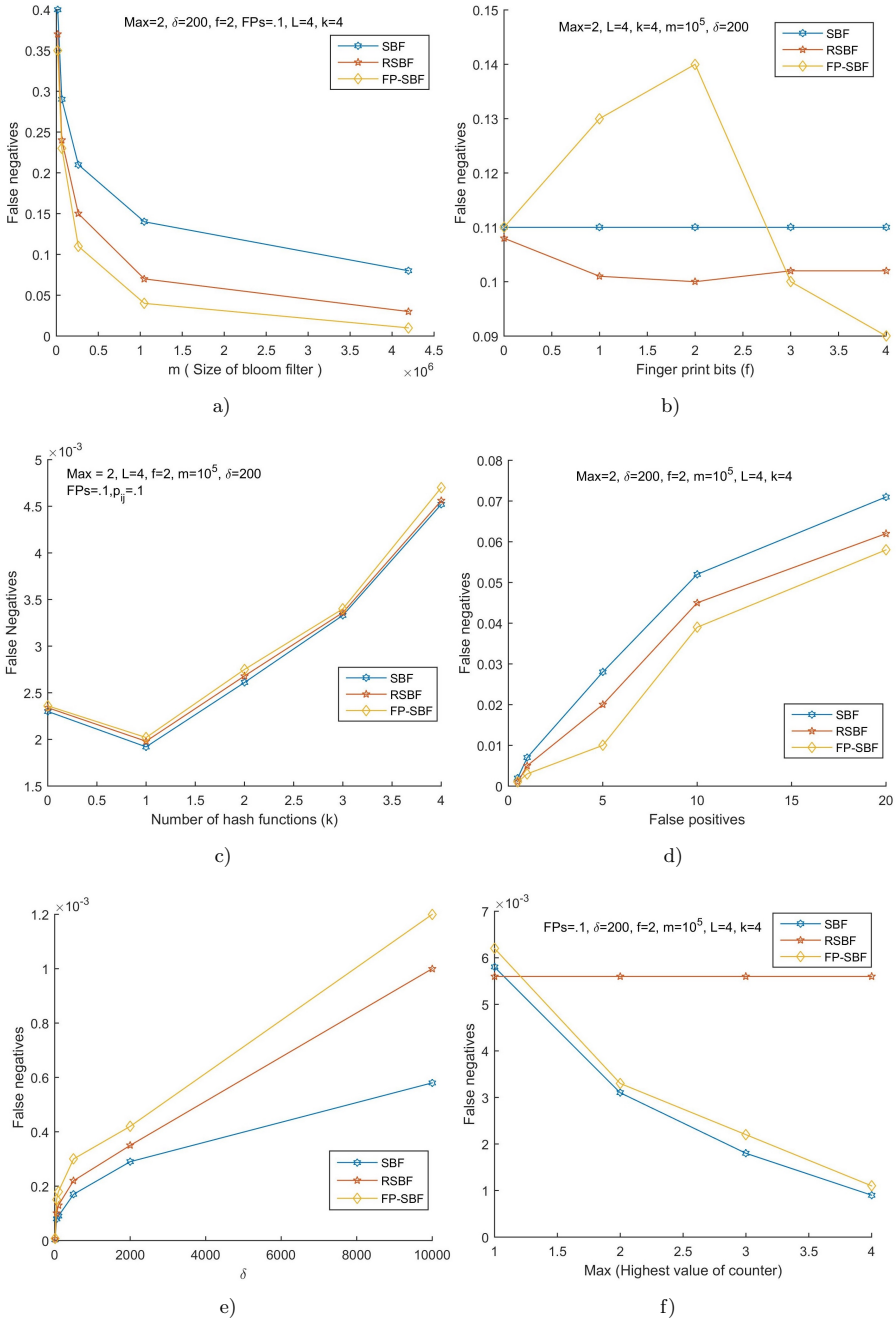

Figure 5. False negatives variation with different parameters

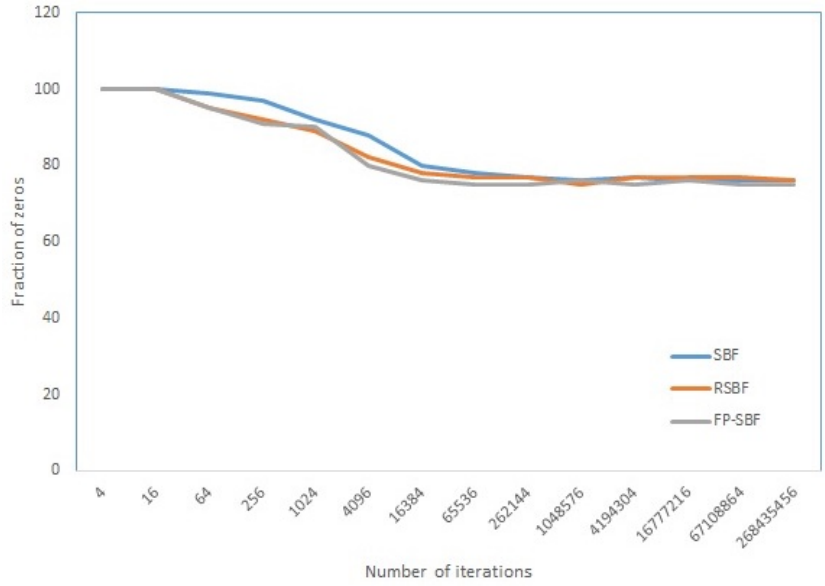

Figure 6. Stable point

Comparative analysis has been performed between SBF, RSBF and FP-SBF. All the experiments indicate that the proposed approach outperforms SBF and RSBF; both the approaches used for duplicate detection in the streamed data using Bloom filters.

Figure 5 shows the impact on false negatives with variations in size of Bloom filter (m), size of counter in bucket (Max), number of fingerprint bits (f), number of hash functions (k) and false positives (FP). All results have been evaluated using fixed values for the required parameters.

As shown in Figure 5 a), false negatives decrease with the increase in size of Bloom filter; the more the space, the less the effect of deletion operation and the less the false negatives. Figure 5 b) indicates the change in false negatives with respect to fingerprint bits in FP-SBF; since the results of SBF and RSBF are not effected by value of f, so false negatives remain the same. In FP-SBF first false negative increases when f is small; for $f = 3$, the accuracy is same as in the RSBF, but as the value of f increases, the acurracy of the proposed scheme increases. Figure 5 c) indicates that as the number of hash functions increases more positions need to be checked, increasing the chances of false negatives. With the change in predefined false positives the changes in false negatives are indicated by Figure 5 d) showing that the more the FPs, the bigger error is allowed; the less deletion operations, the less false negatives appear. δ denotes the average number of iterations between two similar items in the stream, effect of δ on false negatives is shown in Figure 5 e) which clearly shows that as the

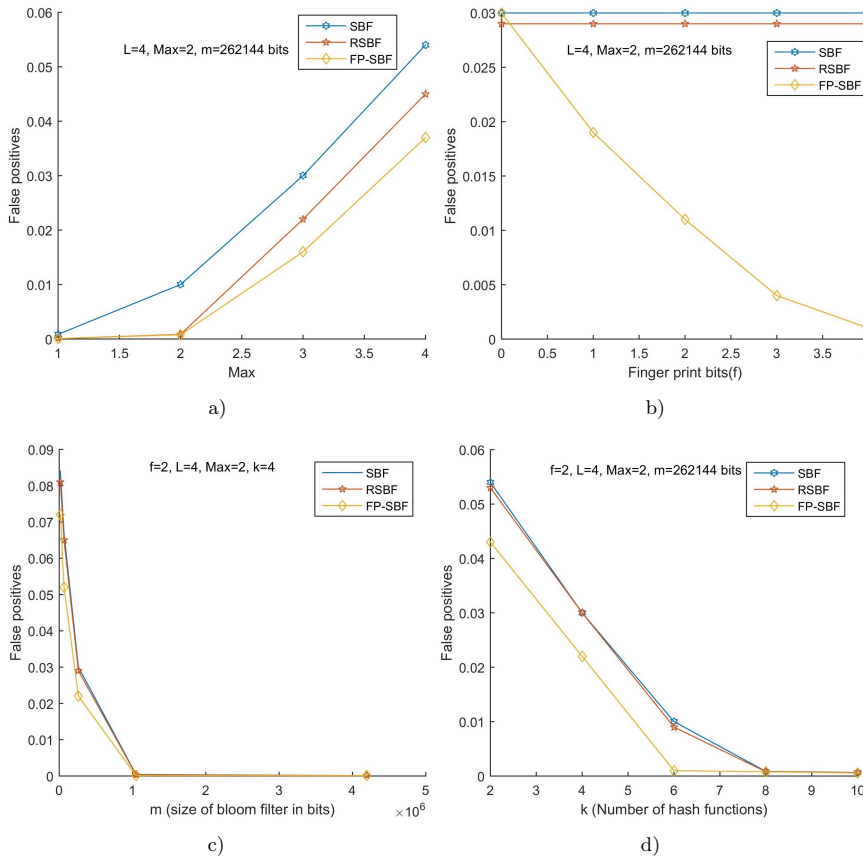

Figure 7. False positives variation with different parameters

value of δ increases, duplicates are detected after more iterations, hence more deletion operations are performed between two similar elements, so chances are high that a duplicate is detected as a distinct element. Figure 5 f) shows the change in false negatives with respect to bucket size *Max*; the larger the bucket size, the more operations are required to reset it to zero and the less false negatives appear.

Figure 6 depicts the number of iterations required to achieve stable point, i.e. constant fraction of zeros in the Bloom filter. Initially, the filter is empty so number of zero is 100 %. As the number of iterations increases, more insertion operations are performed and number of zeros is reduced; after some time deletion operation is also performed and later number of zeros becomes constant. Both RSBF and FP-SBF perform efficiently in achieving the stable point but FP-SBF has the advantage of controlling the deletion operation by using optimized

deletion process which reduces the number of iterations required to reach stable point.

Figure 7 provides the false positives analysis which shows that with the use of finger print bits false positives are drastically reduced in the FP-SBF, as compared to RSBF and SBF. In Figure 7 a) false positives increase with the size of counter in bucket, but the use of fingerprint bits in the FP-SBF helps to reduce it to a great extent. Figure 7 b) shows that the more the number of fingerprint bits used the less false positives appear in duplicate detection task. Figure 7 c) indicates that the larger the size of Bloom filter used, the less false positives appear. Number of hash functions is always a critical factor in the Bloom filter, Figure 7 d) shows that the more hash functions used, the less false positives appear.

6 DISCUSSION

FP-SBF uses fingerprint bits to improve the accuracy of the task and there are cases when false negatives increase with the reduction in false positives. Few use cases are discussed below to analyse the output in various scenarios.

- Parameter tunning in FP-SBF which can decrease the false positives to minimum level is:

$$MFP((Max > 3), (f > 3), (m > 5 \times 10^5), (k > 5)). \tag{24}$$

Case I – where FP-SBF can be used successfully – includes:

  - IoT data streams, where data is coming from number of sensors, FP-SBF can be used to check the identity of the sensors.
  - Detection of the first time user in real time data streams of online shopping platforms for promotional strategies.
  - Detection of the active users in social networking websites like Twitter, Facebook, etc., from the number of posts registered in the given time span t.

- Parameter tunning in FP-SBF which can decrease the false negatives to minimum level is:

$$MFN((\delta < 1\,000), (Max > 4), (f > 4), (m > 4 \times 10^6), (2 < k < 5)). \tag{25}$$

Case II – where FP-SBF might not be the best option – includes:

  - Medical domain, where life saving critical decisions need to be taken.
  - Surveillance and monitoring of real time data for security and safety purposes.
  - Online detection of financial frauds like credit card fraud, money laundering activities, illegal betting, etc.

7 CONCLUSION

Duplicate detection task on streaming data in one pass is among the most important tasks associated with in-stream analytics. To accommodate unbounded data and detect the duplicate in stream, the proposed framework FP-SBF uses advanced stable Bloom filter with fingerprint bits to decrease error rate. d-left hashing has been used which leads to less collisions and update minimum number of counters in insertion operation. Further d-left hashing improves the accommodation capacity of Bloom filter and improve accuracy in results. A randomized approach in deletion process is used to reduce the computational overhead as compared to existing technique. Results achieved clearly indicate that the proposed framework performs efficiently for duplicate detection problem and further parameters can be tunned according to specific application's requirement.

Acknowledgment

The first author would like to acknowledge the financial support given to him by the Department of Computer Science and Technology under the Department of Electronics and Information Technology (DeitY) to complete his doctoral studies.

REFERENCES

[1] Korwar, A.: Bloom Filters. `http://www.cse.iitk.ac.in/users/arpk/articles/BloomFilters.pdf`. [Online; May 2010].

[2] Pike, G.—Alakuijala, J.: Introducing CityHash. `https://opensource.googleblog.com/2011/04/introducing-cityhash.html`. [Online; April 11, 2011].

[3] What Is Cloud Computing? `http://aws.amazon.com/what-is-cloud-computing/`. [Online; 2013].

[4] Alur, R.—Berger, E.—Drobnis, A. W.—Fix, L.—Fu, K.—Hager, G. D.—Lopresti, D.—Nahrstedt, K.—Mynatt, E.—Patel, S. et al.: Systems Computing Challenges in the Internet of Things. arXiv preprint arXiv:1604.02980, 2016.

[5] Antichi, G.—Ficara, D.—Giordano, S.—Procissi, G.—Vitucci, F.: Blooming Trees for Minimal Perfect Hashing. IEEE 2008 Global Telecommunications Conference (IEEE GLOBECOM 2008), 2008, pp. 1–5, doi: 10.1109/glocom.2008.ecp.305.

[6] Azar, Y.—Broder, A. Z.—Karlin, A. R.—Upfal, E.: Balanced Allocations. SIAM Journal on Computing, Vol. 29, 1999, No. 1, pp. 180–200, doi: 10.1137/s0097539795288490.

[7] Babcock, B.—Babu, S.—Datar, M.—Motwani, R.—Widom, J.: Models and Issues in Data Stream Systems. Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02), ACM, 2002, pp. 1–16, doi: 10.1145/543613.543615.

[8] Bloom, B. H.: Space/Time Trade-Offs in Hash Coding with Allowable Errors. Communication of the ACM, Vol. 13, 1970, No. 7, pp. 422–426, doi: 10.1145/362686.362692.

[9] Bonomi, F.—Mitzenmacher, M.—Panigrahy, R.—Singh, S.—Varghese, G.: An Improved Construction for Counting Bloom Filters. In: Azar, Y., Erlebach, T. (Eds.): Algorithms – ESA 2006. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4168, 2006, pp. 684–695, doi: 10.1007/11841036_61.

[10] Chang, F.—Feng, W.-C.—Li, K.: Approximate Caches for Packet Classification. Proceedings of the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004), 2004, Vol. 4, pp. 2196–2207, doi: 10.1109/infcom.2004.1354643.

[11] Chen, H.—Chiang, R. H. L.—Storey, V. C.: Business Intelligence and Analytics: From Big Data to Big Impact. MIS Quarterly: Management Information Systems, Vol. 36, 2012, No. 4, pp. 1165–1188, doi: 10.2307/41703503.

[12] Cohen, S.—Matias, Y.: Spectral Bloom Filters. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03), ACM, New York, NY, USA, 2003, pp. 241–252, doi: 10.1145/872757.872787.

[13] Deng, F.—Rafiei, D.: Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters. Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06), ACM, New York, NY, USA, 2006, pp. 25–36, doi: 10.1145/1142473.1142477.

[14] Dutta, S.—Bhattacherjee, S.—Narang, A.: Towards Intelligent Compression in Streams: A Biased Reservoir Sampling Based Bloom Filter Approach. Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12), ACM, 2012, pp. 228–238, doi: 10.1145/2247596.2247624.

[15] Dutta, S.—Narang, A.—Bera, S. K.: Streaming Quotient Filter: A Near Optimal Approximate Duplicate Detection Approach for Data Streams. Proceedings of the VLDB Endowment, Vol. 6, 2013, No. 8, pp. 589–600, doi: 10.14778/2536354.2536359.

[16] Fan, B.—Andersen, D. G.—Kaminsky, M.—Mitzenmacher, M. D.: Cuckoo Filter: Practically Better Than Bloom. Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14), ACM, 2014, pp. 75–88, doi: 10.1145/2674005.2674994.

[17] Fan, L.—Cao, P.—Almeida, J.—Broder, A. Z.: Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking (TON), Vol. 8, 2000, No. 3, pp. 281–293, doi: 10.1109/90.851975.

[18] Kapoor, A.—Arora, V.: Application of Bloom Filter for Duplicate URL Detection in a Web Crawler. 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), 2016, pp. 246–255, doi: 10.1109/cic.2016.042.

[19] Kirsch, A.—Mitzenmacher, M.: Less Hashing, Same Performance: Building a Better Bloom Filter. Random Structures and Algorithms, Vol. 33, 2008, No. 2, pp. 187–218, doi: 10.1002/rsa.20208.

[20] Liberty, E.—Nelson, J.: Streaming Data Mining. Presented at Princeton University by Yahoo Research Group, online 2012.

[21] Merlin, J. S.—Mary, A. V. A.: An Approach for Quick and Efficient Detection of Duplicate Data-Survey. International Journal of Applied Engineering Research, Vol. 11, 2016, No. 5, pp. 3430–3432.

[22] Metwally, A.—Agrawal, D.—El Abbadi, A.: Duplicate Detection in Click Streams. Proceedings of the 14th International Conference on World Wide Web (WWW '05), ACM, New York, NY, USA, 2005, pp. 12–21, doi: 10.1145/1060745.1060753.

[23] Pontarelli, S.—Reviriego, P.—Maestro, J. A.: Improving Counting Bloom Filter Performance with Fingerprints. Information Processing Letters, Vol. 116, 2016, No. 4, pp. 304–309, doi: 10.1016/j.ipl.2015.11.002.

[24] Rothenberg, C. E.—Macapuna, C. A. B.—Verdi, F. L.—Magalhaes, M. F.: The Deletable Bloom Filter: A New Member of the Bloom Family. IEEE Communications Letters, Vol. 14, 2010, No. 6, pp. 557–559, doi: 10.1109/lcomm.2010.06.100344.

[25] Rottenstreich, O.—Kanizo, Y.—Keslassy, I.: The Variable-Increment Counting Bloom Filter. IEEE/ACM Transactions on Networking, Vol. 22, 2014, No. 4, pp. 1092–1105, doi: 10.1109/tnet.2013.2272604.

[26] Singh, M. P.—Hoque, M. A.—Tarkoma, S.: Analysis of Systems to Process Massive Data Stream. arXiv preprint arXiv:1605.09021, 2016.

[27] Wei, J.—Jiang, H.—Zhou, K.—Feng, D.—Wang, H.: Detecting Duplicates over Sliding Windows with RAM-Efficient Detached Counting Bloom Filter Arrays. 2011 IEEE Sixth International Conference on Networking, Architecture and Storage (NAS '11), 2011, pp. 382–391, doi: 10.1109/nas.2011.37.

[28] Yoon, M.: Aging Bloom Filter with Two Active Buffers for Dynamic Sets. IEEE Transactions on Knowledge and Data Engineering, Vol. 22, 2010, No. 1, pp. 134–138, doi: 10.1109/tkde.2009.136.

Amritpal Singh received his Ph.D. degree from Thapar Institute of Engineering and Technology (TIET), Punjab, India, with a minor in big data analysis and probabilistic data structures in 2018. He is working as Lecturer with Computer Science Department at TIET, Punjab, India. He served both industry and academia. His research interest includes probabilistic data structures, machine learning and big data.

Shalini Batra received her Ph.D. degree in computer science and engineering from Thapar University, Patiala, India, in 2012. She is currently working as Associate Professor with the Department of Computer Science and Engineering, Thapar University, Patiala, India. She has guided many research scholars leading them to Ph.D. and M.E./M.Tech. She authored more than 60 research papers published in various conferences and journals. Her research interests include machine learning, web semantics, big data analytics and vehicular ad-hoc networks.