Importance-aware Bloom Filter for Set Membership Queries in Streaming Data

Puru Kulkarni, Ravi Bhoraskar, Vijay Gabale
Indian Institute of Technology (IIT), Bombay, India

Dhananjay Kulkarni
Asia Pacific Institute of Technology (APIT), Sri-Lanka

***Abstract*—Various data streaming applications like news feeds, stock trading generate a continuous sequence of data items. In such applications, based on the priority of different items, a set of items are more important than others. For instance, in stock trading, some stocks are more important than others, or in a cache-based system, the cost of fetching new objects into the cache is directly related to the the size of the objects. Motivated by these examples, our work focuses on developing a time and space efficient indexing and membership query scheme which takes into account data items with *different* importance levels. In this respect, we propose Importance-aware Bloom Filter (IBF), an extension of Bloom Filter (BF) which is a popular data structure to approximately answer membership queries on a set of items. As part of IBF, we provide a set of insertion and deletion algorithms to make BF importance-aware and to handle a set of unbounded data items. Our comparison of IBF with other Bloom filter-based mechanisms, for synthetic as well as real data sets, shows that IBF performs very well, it has low false positives, and low false negatives for important items. Importantly, we find properties of IBF analytically, for instance, we show that there exists a tight upper bound on false positive rate independent of the size of the data stream. We believe, IBF provides a practical framework to balance the application-specific requirements to index and query data items based on the data semantics.**

I. Introduction

Various sources, such as news feeds[7], stock trades [3], [2], [5], sensors [6], [4], and online bidding systems [18] generate data continuously. Recent literature [12] has classified such data as *data streams*. Since traditional techniques (for indexing and storing) used with relational databases [8] are not directly applicable in the context of data streams, there has been a significant research in building Data Stream Management Systems [12], [11], [23], [13], [26], [22], [25], and recently there have been some commercial products [19], [1] as well.

There exist numerous challenges in storing and querying a data stream. Our work focuses on indexing an unbounded stream of data items to answer *set membership* queries. Since data streams are continuous and unbounded, it is impractical to store the entire data during query processing, especially so in finite-memory systems. In such contexts, it is difficult to provide precise answers to queries. To limit the amount data to be stored, or to be processed per data item, time-based *sliding window* [9] techniques have been used for *continuous queries* (CQ) [9]. A continuous query executes over successive instances of the sliding time-window. Though continuous queries on sliding windows provide a way to limit processing load, they still have to deal with bursty data arrival, unpredictable data distributions, or finite window sizes. In our work, we consider a *landmark window*, where one end of the time-window is fixed, for example start-of-day, and the other end of the window is unspecified. With this setting, the system would index all or partial data items and answer membership queries.

Another critical observation that motivates our work is the notion of *importance* of data items (or tuples). The following examples introduce *data-importance* and motivates the need to perform importance-aware indexing and querying, and performance of the same.

Scenario 1: Consider a portfolio of stocks and an *importance* related to each. Assignment of importance to each stock can be based on one or more parameters—stock price, number of stocks, past and predicted stock-price fluctuations etc. Over the period of a day, the value of each stock fluctuates. Suppose an user is interested in knowing which stocks changed by more than 10% of its opening value. A less accurate answer regarding *less* important stocks is tolerable, whereas an inaccurate answer on *more* important stocks can result in sub-optimal actions to maximize portfolio value.

Scenario 2: A *web crawler* starts crawling from a particular web page, collects links on that page, and then crawls these links recursively, similar to a breadth-first-traversal of a graph. It is obvious that some links are popular and will appear on multiple web pages, e.g., a BBC news article may appear on international news listing of several web-sites. Crawlers need to accurately identify such popular links and avoid crawling them multiple times. An importance function in such cases can be correlated with the popularity of a web page or a web-site. A crawlers expectation of accuracy regarding the question of whether a page has been previously crawled can be related to its importance— popular pages require more accurate answers.

Scenario 3: Consider a data structure associated with a cache that can be looked-up to determine if an object is present in the cache. Objects stored in the cache are of different sizes and the *cost* to fetch them from a remote server is a function of their size. Assuming the look-up service provides a probabilistic answer, to minimize the *cost* of fetching, it is desirable for the look-up service to be more accurate for larger objects as compared to objects with smaller sizes.

These examples motivate us to look at schemes that exploit data importance during indexing and membership queries. We believe that the assumption that all data items have the same importance level is too simplistic and not universally applicable in practice.

In this respect, our work focuses on developing an indexing scheme assuming data items with *different* importance levels. We assume that available memory is limited, hence it is not possible to store an entire unbounded data set. Our goal is

to store an approximate summary of each data item using a hash-based index, popularly known as the Bloom Filter (BF). However, since the volume of data insertions is high the traditional Bloom Filter would quickly "fill up", and lead to a large number of false positives (FPs). Recent work on Stable Bloom Filters (SBF) [16] limits the false positives by clearing cells of a bloom filter over time. Although, SBF tries to limit the false positive rate, it is agnostic to data importance— both during insert and delete operations. As we show our in our evaluations, exploiting the data-importance semantics during indexing, and deletion of items provides better query accuracy, one that is correlated to importance of data items.

Our approach, Importance-aware Bloom Filter (IBF), is a modification of the traditional Bloom Filter which incorporates importance-aware semantics for insertion and deletion, and operates on an unbounded data set. Apart from basic modifications to the Bloom Filter operations, we also examine schemes that balance the false-positive and false-negative rates. The key idea (similar to Stable Bloom Filter [17]) is to periodically decrement values of a subset of cells to make room for new items in the data stream. However, in IBF, the insertion as well deletion operation captures the semantics of the data, i.e., data importance. Specifically,

in this work, we make the following contributions.

- We propose Importance-aware Bloom filter (which extends SBF [16]), a data structure for indexing and querying data items on an unbounded set based on their importance.
- Via analysis we show the IBF (similar to SBF) is stable, i.e., the bloom filter has a constant upper-bound on the false-positive and fale-negative rates irrespective of the stream size. The time complexity of IBF to process a data time is constant irrespective of the stream size.
- Given a fixed amount of space, we evaluate and compare performance of IBF to BF and SBF on real as well as synthetic data. Our results show that IBF behaves well in capturing the data importance, and it has lower false positives and false negatives for data items with higher importance, in comparison to BF and SBF.

The rest of the paper is organised as follows. In next section, Sec. II, we formally describe our problem, and give a brief sketch of IBF. In Sec. III, we describe IBF in detail and analyse its properties theoretically. Next, in Sec. IV, we show effectiveness of IBF by comparing it with prior work. In Sec. V, we compare IBF with prior work in indexing and querying streaming data. Finally, in Sec. VI, we discuss future work, and conclude the paper.

II. Background and problem statement

Our work is partially inspired by the work in [16] which proposes a data structure to approximately answer membership queries on a stream of data items. In this section, first, we briefly describe work in [16]. Then, we formally define our problem formulation, and explain why work in [16] is not suitable to our problem. Subsequently, we give a brief overview of our solution– Information-aware Bloom Filter (IBF). We discuss IBF in detail in next section (Sec. III).

A. Stable bloom filter

Traditionally, the Bloom filter [10] (BF) data structure has been used to store data itmes to approximately answer the membership query. A membership query asks whether *a data item d_i is present in the memory* or not. Note that, a Bloom filter only stores a hash for each data item, whereas the data items are physically stored in a seperate memory (typically an external disk-memory), distinct from the Bloom filter. To answer membership query for an item, a Bloom filter takes the hash of the data item, and returs the answer as 'Yes' or 'No' in constant time, but with some probability. Thus, a Bloom filter is cost efficient than indexing data structures like B-Tree or B^+-Tree, which require several memory accesses to search for an item. The entire Bloom filter can reside in main memory, and it takes only one memory access to know whether an item is present or not. However, the constant time search of Bloom filter comes at the cost of the probabilistic behavior in answering the membership query.

A Bloom filter is essentially an array of m bits, initially set to 0. When a new item d_j is to be stored in the memory, BF uses K uniformly distributed and independent hash functions, and computs K bit locations: $\{h_1(d_j), h_2(d_j), \ldots, h_k(d_j)\}$, to be set to 1. To answer a membership query, BF again applies K hash functions, and checks whether all of the corresponding K bit locations are set to 1 or not. If so, BF replies with a duplicate. However, the item may not be a duplicate, and the K bit locations may be set for some other data items previously stored in the memory. This gives rise to false positive; the case where BF can incorrectly answer a given query. To explain false positives formally, consider a query $Q(d)$, which tests for the membership of item d. A false positive occurs when d is not present in BF, but $Q(d)$ returns true (d is present in BF). A false negative occurs when $Q(d)$ returns false, although d is present in BF.

The work in [16] extends Bloom Filter data structure, and proposes Stable Bloom Filter (SBF), which (1) adds "eviction" operation to BF, and (2) supports querying on a stream of data items. The naive BF, with a fixed number of bits, may not be applicable to store the data items in a stream. This is because, as more and more elements arrive, the fraction of zeros in the BF will decrease continuously, and the false positive rate will increase accordingly. Eventually, the false positive rate will reach the limit, 1, where every distinct item will be reported as a duplicate. Stable bloom filter avoids this problem by deleting some items before the error rate reaches a predefined threshold. To do this, stable bloom filter adds a random deletion operation into the Bloom filter so that false positive rate exceed a threshold.

We now give a brief description of SBF. A SBF is defined as an array of integers SBF[1],...,SBF[m]. The size of the array is m elements, and each element can take a minimum value of 0, and a maximum value M. Each element of the SBF is called as a cell. Thus, each cell of the array is allocated d bits; the relation between M and d is $M = 2^d - 1$. The initial value of the cells is zero. Each newly arrived item in the stream is mapped to K cells by some uniform and independent K hash functions. As in a regular Bloom filter, SBF checks if a new item is duplicate or not by checking whether all the cells the

item is hashed to are non-zero. This is the duplicate detection process. After the duplicate detection process, before inserting new item into the SBF, it randomly decrements P cells by 1 so as to make room for fresh items, and then sets the same K cells used during detection process to M.

In [16], the authors prove that after a number of iterations, the fraction of zeros in the SBF will be non-zero irrespective of the values of M, K or P. This is termed as the stable property, and hence the name Stable Bloom Filter. It also proves that, if the stream elements are uniformly distributed, the false positive rate can be upper bounded.

B. Problem statement

We now define our problem formally. Let $I : d_i \rightarrow \mathbb{IMP}$ be an importance-function that maps some attribute of the data item d_i to a value imp_i, which represents the importance of data item d_i. We assume that items arrive in a stream denoted by $D = d_1, d_2, \ldots, d_n$, where n is the number of items arrived so far. The value of n can be infinite, which means stream is not bounded. We assume that, using a fixed hash function or fingerprinting, each data item can be coverted into a number, and henceforth, when we refer to an item d_i, it is the hash or fingerprint of the original data item.

Now, given a set of data items $D = d_1, d_2, \ldots, d_n$, an importance function I and a finite amount of memory S, our **overall objective** is to store items in D so that membership of an item $d_i \in D$ can be determined in constant time. This objective is similar to the problem in [16], which determines membership of an item with a definite false positive and false negative probability. However, note that, our data items are "tagged" with importance values, and our **requirement** is to have good confidence in answers when data items with high importance are queried. That is, our **goal** is to have a low false positive rate, and a low false negative rate while answering queries for data items with high importance values. *How should we design an efficient data structure to achieve this goal?*

To answer this question, we design IBF, an Importance-aware Bloom Filter which takes data semantics, and hence different importance levels into account, while indexing a stream of data items. Note that, the work on SBF in [16] is not suitable to our goal because SBF does not differentiate between items based on the importance values while indexing the data items and answering the membership queries.

C. Information-aware Bloom Filter (IBF) overview

We now give a brief overview of IBF. Essentially for IBF, we modify the insert operation in SBF [16] to make the data structure importance-aware. The overall result of such modification is that, now the items having high importance are stored for longer time. Similarly, we evict elements from IBF such that, the items having high importance have less probability of eviction. The importance-aware insert and delete operation thus result in lower false positives (and lower false negatives) for items with high importance. At the same time, as a desirable effect, (which we confirm through the our experiments) the false positive rate and false negative rate for items with lower importance remains reasonable low in most of the cases.

III. IBF: Algorithm and Analysis

As we mentioned in last section, we insert or delete elements in IBF such that, important data items have lower probability of false positives or false negatives. In this section, we first describe a simple extension of SBF, IBF-2C [1]. In comparison to SBF, which sets the values of cells to M while inserting a new item, in IBF-2C, we either set the cell values to $\frac{M}{2}$ or to M, depending on importance of the data item. In a generic case, we assume that a function f is given to us, which maps the importance value to a number between 1 and M, i.e., $f : imp_i \rightarrow (1, M)$. In case of IBF-2C, this function maps the importance value to either $\frac{M}{2}$ or M. Next, we describe a generalization of IBF, IBF-MC, where while inserting an item, we set the cell values to a number $\in (1, M)$ based on the importance value of the data item. For IBF-2C as well as IBF-MC, we present analysis of false positive and false negative rates, and prove the stability property.

A. IBF-2 Class (2C)

We now introduce IBF-2C, an Information-aware Bloom filter which has importance-aware insert operation, and random delete operation.

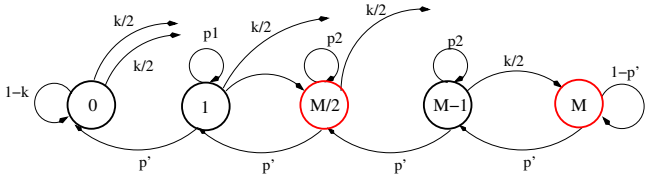

Fig. 1. IBF-2C: Important Insert Random Delete

Information-aware Bloom Filter-Two Class (IBF-2C) An IBF-2C is defined as an array of integers $IBF[1], \ldots, IBF[m]$. The value of each cell of the array is between 0 (minimum) or M (maximum). The update process follows Algorithm 1. Each cell of the array is allocated d bits; the relation between M and d is $M = 2^d - 1$.

We now explain Algorithm in 1. In Alg. 1, each incoming element is mapped to K cells by K uniform and independent hash functions. We then check, by probing the K cells, whether all the cells are non-zero, and see if the element is a duplicate. After this step, we update IBF-2C as follows. We first pick P cells randomly, and decrement the cell value by 1 (unless it is already zero). This steps is to make room for future elements in the stream. We then set the same K cells to either $\frac{M}{2}$, if the data item is not important or M, if the data item is important.

Analysis: We now analyze IBF-2C, and prove the stable property, which states that when the number of items in stream are sufficiently large, the fraction of zeros in IBF-2C will converge to a fixed value irrespective of the values of M, K, and P. This is an important property from both theory as well as practical point of view as it bounds the false positive rate.

To analyze IBF-2C, we assume that the data items as well as the importance values are uniformly distributed. We then model the cell value of IBF-2C as a discrete time Markov

[1] 2C stands for two classes.

chain as shown in fig. 1. Each iteration of the algorithm is a time step in the Markov chain. In each time step, the cell value is either decremented by 1 or is set to $\frac{M}{2}$ or M or remains the same. These conditions are captured in transition probabilities in the Markov chain shown in fig. 1. If a cell value is T, after one iteration it goes to state $T-1$ with probability $p' = p \times (1-k)$, where p is the probability of decrementing a cell, and k is the probability setting the cell. If the cell is set, it goes to state $\frac{M}{2}$, with probability $\frac{k}{2}$ or to state M with probability $\frac{k}{2}$. Otherwise, the cell remains in the same state. It is easy to see that this Markov chain is indeed a valid discrete time Markov chian with irreducibility (every state is reachable from every other state) and aperiodicity (period of recurrence for each state is 1). For any irreducible, aperiodic Markov chain, the limiting probabilities V_i for each state exist and are unique [31]. Let $P_{i,j}$ denote the probability of transition from state i to state j.

Input: A sequence of numbers
Output: A sequence yes or no corresponding to each input number

Initialize IBF[1],...,IBF[m] to 0.
foreach *number* x_i **do**
  Probe K cells IBF[$h_1(x_i)$],...,IBF[$h_K(x_i)$].
  if *none of the above K cells is 0* **then**
    DupFlag = yes
  end
  DupFlag = no
  Select P cells uniformly at random.
  foreach *each cell IBF[j]* $\in$ { *IBF[j_1],..., IBF[j_p]* } **do**
    if *IBF[j]* $\geq$ *1* **then**
      IBF[j] = IBF[j] - 1
    end
  end
  foreach *each cell* $\in$ *IBF[$h_1(x_i)$],...,IBF[$h_K(x_i)$]* } **do**
    if $f(imp(x_i)) < \frac{M}{2}$ *and IBF[$h(x_i)$]* $< \frac{M}{2}$ **then**
      IBF[$h(x_i)$] = $\frac{M}{2}$
    end
    IBF[$h(x_i)$] = M
  end
  Output DupFlag.
end

Algorithm 1: Approximately Detect Duplicates using IBF

Theorem III.1. *Given an IBF-2C of m cells, if in each iteration, a cell is decremented by 1 with a probability p and set to importance value* $imp \in \{\frac{M}{2}, M\}$ *with a probability k, the limiting probability of a cell becoming zero exists.*

Proof:

The existence of probability is straightforward from the Markov model of IBF-2C. Since the Markov chain is irreducible, aperiodic and stable, each state i will have a non-zero limiting probability V_i [31]. Thus, there is non-zero probability that a cell will have its value zero in limiting cases. We now compute the limiting probability of a cell becoming zero. When limiting probabilities exist, the probability of the system entering a state is same as the probability of leaving the state, and the sum over probabilities of all states is 1 [31]. That is, for every state i, $\sum_j P_{j,i}V_j = (\sum_k P_{i,k})V_i$ (Eq.(1)), and $\sum V_i = 1$ (Eq.(2)). We now compute steady state probabilities for each state.

For state M,
Using Eq.(1), $\frac{k}{2}\sum_{i=0}^{M-1} V_i = p'V_M$.
But $\sum_{i=0}^{M} V_i = 1 \rightarrow \sum_{i=0}^{M-1} V_i = (1 - V_M)$.
Thus, $V_M(p' + \frac{k}{2}) = \frac{k}{2}$, which implies, $V_M = \frac{\frac{k}{2}}{p'+\frac{k}{2}}$.

For state $(M-1)$,
Using Eq.(1), $(\frac{k}{2} + p')V_{M-1} = p'V_M$. $\rightarrow V_{M-1} = \frac{p'}{p'+\frac{k}{2}}V_M$,
which gives $V_{M-1} = \frac{p'}{p'+\frac{k}{2}}\frac{\frac{k}{2}}{p'+\frac{k}{2}}$.

Similarly, **for state** i, $\frac{M}{2} < i < M$,
$V_i = \left(\frac{p'}{p'+\frac{k}{2}}\right)^{(M-i)} \frac{\frac{k}{2}}{p'+\frac{k}{2}}$. (Eq.(3))

For state $\frac{M}{2}$,
Using Eq.(1), $\frac{k}{2}\sum_{i=0}^{\frac{M}{2}-1} V_i + p'V_{(\frac{M}{2}+1)} = \left(p' + \frac{k}{2}\right)V_{\frac{M}{2}}$.
Replacing $V_{(\frac{M}{2}+1)}$ using Eq.(3)
and $\sum_{i=0}^{\frac{M}{2}-1} V_i$ by $\left(1 - V_{\frac{M}{2}} - \sum_{i=\frac{M}{2}+1}^{M} V_i\right)$, we get
$\frac{k}{2}\left(1 - V_{\frac{M}{2}} - \sum_{i=(\frac{M}{2}+1)}^{M} V_i + \left(\frac{p'}{p'+\frac{k}{2}}\right)^{\frac{M}{2}}\right) = \left(p' + \frac{k}{2}\right)V_{\frac{M}{2}}$.
If $A = \sum_{i=(\frac{M}{2}+1)}^{M} V_i$, and $B = \left(\frac{p'}{p'+\frac{k}{2}}\right)^{\frac{M}{2}}$,
we get $V_{\frac{M}{2}} = \left(\frac{\frac{k}{2}}{p'+k}\right)(1 - A - B)$.

For state $\left(\frac{M}{2} - 1\right)$,
$p'V_{\frac{M}{2}} = (p' + k)V_{\frac{M}{2}-1} \rightarrow V_{\frac{M}{2}-1} = \left(\frac{p'}{p'+k}\right)V_{\frac{M}{2}}$

Similarly, **for state** i, $\frac{M}{2} > i > 0$, $V_i = \frac{p'}{p'+k}V_{\frac{M}{2}}$

Now, using Eq.(2), $V_0 = (1 - \sum_{i=1}^{M} V_i)$.

■

$p_1 = 1 - (p' + k)$
$p_2 = 1 - (p' + \frac{k}{2})$
$V_M = \frac{(\frac{k}{2})}{(1-p_2)}$
$V_i = (\frac{p'}{(1-p_2)})^{M-i} \times V_M$ for $i > \frac{M}{2}$ and $i < M$
$V_{M/2} = \frac{\frac{k}{2}}{p'+k} \times (1 - A + B)$ where $A = \sum_{i=\frac{M}{2}+1}^{M} V_i$
and $B = \frac{p'}{(1-p_2)})^{\frac{M}{2}}$
$V_i = (\frac{p'}{(1-p_1)})^{\frac{M}{2}-i} V_{\frac{M}{2}}$ for $i > 0$ and $i < \frac{M}{2}$
$V_0 = 1 - \sum_{i=1}^{M} V_i$

Fig. 2. Limiting probabilities of states of Markov chain shown in Fig. 1.

Corollary III.2. (Stable Property) *After large iterations, the expected fraction of zeros in IBF-2C is a constant.*

Proof: In each iteration, each cell of the SBF has a certain probability of being set to M or to M/2 by the item hashed to that cell. For a fixed distribution of input data, the probability that a particular cell is processed in each iteration is fixed. Therefore, the probability of each cell being set is fixed. Also, the probability that an arbitrary cell is decremented by 1 is also a constant. By theorem III.1, the probabilities of all cells in the SBF becoming 0 after N iterations are constants, for sufficiently large value of n. Therefore, the expected fraction of 0 in an SBF after n iterations is a constant. ■

Theorem III.3. (Stable Point (Average Case)) *When IBF is stable, the expected fraction of 0s in the IBF is $m \times V_0$, where $p = \frac{P}{M}$ and $k = \frac{K}{M}$.*

Proof: Since we assume uniform distribution for data and importance values, each cell has the same limiting probability of being set zero. Thus, on an average, when IBF-2C has m cells, the fraction of cells having zero values is $\sum_{i=0}^{m} Pr(\text{cell i is zero})$, which is $m \times V_0$. ■

In IBF, there could be two kinds of errors: false positives (FP) and false negatives (FN). A false positive happens when a distinct element is wrongly reported as duplicate. A false negative happens when a duplicate element is wrongly reported as distinct. We call the respective probabilities as the false positive rate (FPS) and false negative rate (FNS).

Corollary III.4. (FP Bound When Stable) *When IBF is stable, the FP rate is constant and no greater than FPS, $FPS = (1 - V_0)^K$*

Proof: If V_{j0} denotes the probability that the cell $IBF[j] = 0$ when IBF is stable, the upper bound on FP rate is $FPS = (\frac{1}{M}(1 - V_{10}) + ... + \frac{1}{M}(1 - V_{M0}))^K$, i.e., $FPS = (1 - \frac{1}{M}(V_{10} + ... + V_{M0}))^K$ Note that $\frac{1}{M}(V_{10} + ... + V_{M0})$ is the expected fraction of 0s which is V_0 in average case. ■

False negative rate: False negative rate is related to input data distribution. As in [17], we define a gap to be the number of elements between a duplicate and its nearest predecessor. Suppose a duplicate element x_i whose nearest predecessor is $x_{i-\delta_i}$ is hashed to K cells. A FN happens if any of those K cells is decremented to 0 duing the δ_i iterations when x_i arrives. From the Markov chain, we can calculate the probability, $Pr_0(\delta_i)$ that a cell is decremented to 0 from $M/2$ or M within the δ_i iterations. With this, the probability that FN occurs is given by $Pr(FN_i) = 1 - \prod_{j=1}^{K}(1 - Pr_0(\delta_i))$.

B. IBF-Multi Class (MC)

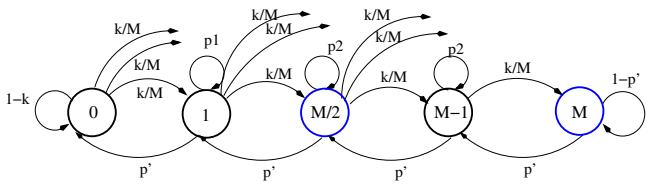

Fig. 3. IBF-MC: Importance-aware Insert, Random Delete

We now briefly describe generalization of IBF-2C, IBF-MC. In IBF-MC, as per the importance of the data item, we set the value of a cell to a value between 1 and M. For items with highest importance, the cell is set to M, whereas for items with lowest importance, the value will be set to 1.

Information-aware Bloom Filter-M-Class (IBF -MC) An IBF-MC is defined as an array of integers $IBF[1], \ldots, IBF[m]$. The minimum value of each cell is 0, and the maximum value is M. The update process follows Algorithm 2. Each cell of the array is allocated d bits; the relation between M and d is $M = 2^d - 1$.

The steps in Algorithm 2 are similar to Algorithm 1 except when we set the cells. Depending on the importance of the item, the K cells are set to value between 1 and M.

Input: A sequence of numbers
Output: A sequence yes or no corresponding to each input number
Initialize IBF[1],...,IBF[m] to 0.
foreach *number x_i* **do**
  Probe K cells IBF[$h_1(x_i)$],...,IBF[$h_K(x_i)$].
  if *none of the above K cells is 0* **then**
    DupFlag = yes
  end
  DupFlag = no
  Select P cells uniformly at random.
  foreach *each cell IBF[j] $\in$ { IBF[j_1],..., IBF[j_p] }* **do**
    if *IBF[j] $\geq$ 1* **then**
      IBF[j] = IBF[j] - 1
    end
  end
  foreach *each cell $\in$ IBF[$h_1(x_i)$],...,IBF[$h_K(x_i)$] }* **do**
    if *IBF[$h(x_i)$] $<$ $f(imp(x_i))$* **then**
      IBF[$h(x_i)$] = $f(imp(x_i))$
    end
  end
  Output DupFlag.
end

Algorithm 2: Approximately Detect Duplicates using IBF

Analysis: Similar to analysis of IBF-2C, we now analyze IBF-MC. We assume that the data items as well as the importance values are uniformly distributed. We then model the cell value of IBF-MC as a discrete time Markov chain as shown in fig. 3. In each iteration or time step, the cell value is either decremented by 1, or set to a value $\in (1, M)$, or it remains the same. These conditions are captured in transition probabilities in the Markov chain shown in fig. 1. This Markov chain is indeed a valid discrete time Markov chian with irreducibility (every state is reachable from every other state) and aperiodicity (period of recurrence for each state is 1). If the cell is set, it goes to state i with probability k/M. Otherwise, the cell remains in the same state.

Theorem III.5. *Given an IBF-MC of m cells, if in each iteration, a cell is decremented by 1 with a probability p and set to importance value $imp \in \{1, M\}$ with a probability k, the limiting probability of a cell becoming zero exists.*

Proof:

The existence of probability is straightforward from the Markov model of IBF-MC. Since Markov chain is stable, each state will have a non-zero limiting probability. Thus, there is non-zero probability that a cell will have its value zero in limiting cases. We omit the details of the probability calculation due to lack of space. The probabilities can be calculated in same fashion as described in proof of Theorem III.5. In Fig. 4, we show how to compute the limiting probabilities V_i for each state i of Markov chain in Fig. 3.

$p' = (1-K) \times P$, $U_1 = \frac{K}{p'}$, $A_1 = 1 + U_1$
$C_2 = (\left(\frac{M-1}{M} \times k\right) + p') \times U_1$, $B_2 = \frac{k}{M}$
$U_2 = \frac{(C-B)}{p'}$, $A_2 = var_1 + U_2$
$C_i = (\left(\frac{M-i+1}{M} \times k\right) + p') \times U_{i-1}$, $B_i = B_{i-1} + \frac{k}{M} \times U_{i-1}$
$U_i = \frac{(C_i - B_i)}{p'}$, $A_i += U_i$
$V_0 = \frac{1}{A_M}$, $V_i = U_i \times V_0$

Fig. 4. Limiting probabilities of states of Markov chain shown in Fig. 3 ■

Corollary III.6. (Stable Property) *After sufficiently large iterations, the expected fraction of zeros in IBF-MC is a constant.*

Proof: Similar to argument in corollary III.2. ■

Theorem III.7. (Stable Point (Average Case)) *When IBF is stable, the expected fraction of 0s in the IBF is $m \times V_0$, where $p = \frac{P}{M}$ and $k = \frac{K}{M}$.*

Proof: Similar to argument in Theorem III.3. ■

Corollary III.8. (FP Bound When Stable) *When IBF is stable, the FP rate is constant and no greater than FPS, $FPS = (1 - V_0)^K$*

Proof: Similar to argument in III.4. ■

The false negative rate can be calculated in similar way as in IBF-2C.

C. Time complexity

Theorem III.9. Time complexity *Given that K, P, and M are constants (set in IBF apriori), the processing of each data element in the input stream take $O(1)$ time, independent of the size of the size of IBF and the length of the stream.*

Proof: The time cost of our algorithm for handling each element is dominated by K, P and M. Within each iteration, we firstly probe K cells to detect duplicates, and then pick P cells to decrement by 1. Then we set K cells to a value between 1 and M. Since K, P, and M are constants, the time complexity is $O(1)$. ■

D. From theory to practice: parameter setting

Setting max and n Given fixed storage space SS, we have $SS = max \times n$. How should we select max and n? Higher the value of n, lower will be the false positive and false negative rate for all-importance items. Higher the value of max, lower will be the false negative rate for more important items. Further, max is also function of the range of $F : d_i \rightarrow \mathbb{I}$. However, higher value of max entails higher values of K and P to maintain the same false positive and false negative rates, which may increase the computational cost per iteration. In our experiments on synthetic data sets, which we will describe in next section, we generate importance values on a scale of 50, i.e., importance values between 1 and 50. For such streams, we set a bench mark of expected false positives and false negatives. For instance, we consider an upper bound of 20% FP for IBF varaints. Considering such upper bound, we run the experiments by varying values of max. We observe that $max = 7$ works reasonably well. With $max = 7$, the importance values are logically partitioned into 8 classes, and an importance value requires only 2-bits per cell. Further, setting $max > 7$ result in only marginal gains even when we take different permutations of the input data streams. Thus, we choose $max = 8$ throughput our experiments. However, we wish to note that $max = 8$ may not result in best performance for IBF for different data streams. Our primary objective, in setting $max = 8$ which gives $\frac{SS}{max}$ cells, is to compare IBF with SBF and BF for a fixed storage space SS.

Setting K and P The theoretical analysis gives a handle to observe the behavior of IBF-2C and IBF-MC by different values of input parameters. For instance, we varied the values of K and P, and observed the false positive and false negative rates. For our experiments, we set bench marks for false positive and false negative rates. For example, we consider an upper bound of 20% on false positive rate, thus select $k = 5$, and $P = 10$ for our experiments.

IV. Experiments

In this section, we compare IBF-2C, IBF-MC with IBF and BF. Our overall goal is to see how these variants behave when different data items have different importance. Thus, for importance aware comparison, we define two new metrics– weighted false positive and weighted false negatives. We also track FP and FN rates for a fixed importance value, across all importance values. For evaluation, we consider two different data sets, one generated synthetically, and other taken from a real-world trace. To make the results independent of the sequence of data items, we repeat the experiments by taking different permutations of the data streams.

A. Experiment parameters

We consider total space available (SS) to be 16KB. Thus, with 8 bits per cell, we have 16384 cells in IBF. We repeated our experiments for different SS values and observed similar trend. Tab. II summarises other experimental parameters.

B. Metrics

Weighted False Positives and Weighted False Negatives: The "impact" of false positives and false negatives for important items is more than that of a less important item. Thus, if a false positive or false negative occurs for an *important* element, a greater penalty is paid by the application. This is well

TABLE I
AGGREGATE BASED EXPERIMENTS: YOUTUBE DATA

	Dataset 1				Dataset 2			
Algo	FP	FN	WFP	WFN	FP	FN	WFP	WFN
BF	34.81	0	35.29	0	34.81	0	34.17	0
SBF	25.83	0.84	26.23	0.87	26	0.9	25.55	0.81
IBF_{2C}	16.14	2.49	16.34	1.99	10.16	3.97	9.95	2.91
IBF_{MC}	5.19	8.91	5.23	4.76	0.84	14.01	0.87	7.52
IBF_{H}	9.08	13.25	9.39	10.43	0.12	16.07	0.07	10.34

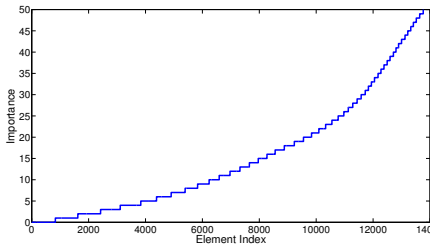

Fig. 5. Data Distribution: Youtube Data 1

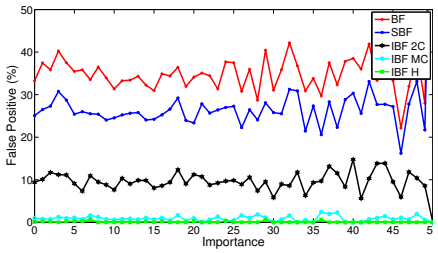

Fig. 6. False Positives: Youtube Data 1

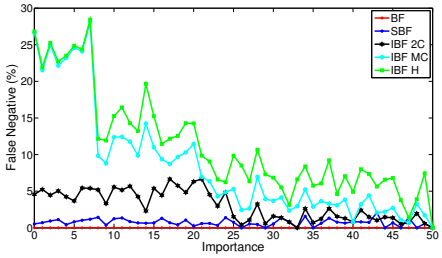

Fig. 7. False Negatives: Youtube Data 1

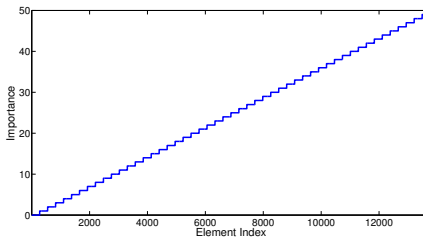

Fig. 8. Data Distribution: Youtube Data 2

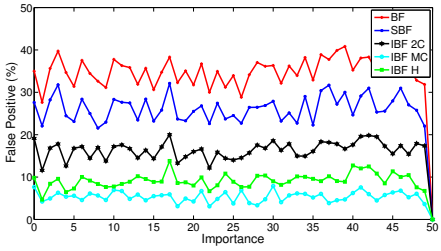

Fig. 9. False Positives: Youtube Data 2

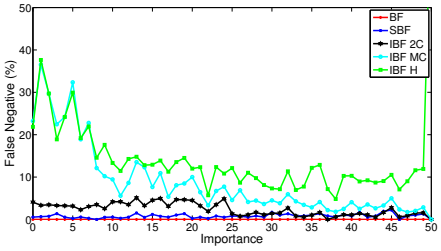

Fig. 10. False Negatives: Youtube Data 2

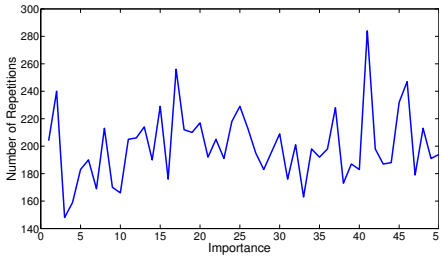

Fig. 11. Data Distribution: Synthetic Data 1

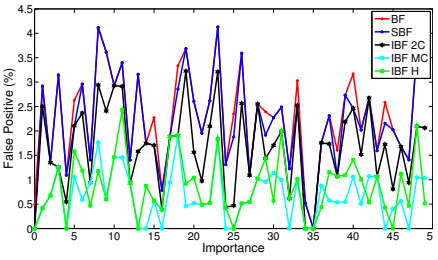

Fig. 12. False Positives: Synthetic Data 1

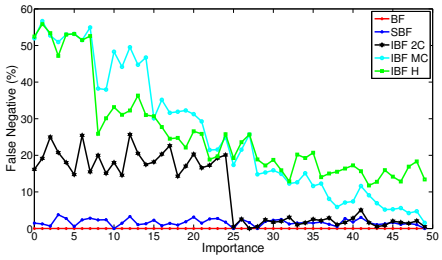

Fig. 13. False Negatives: Synthetic Data 1

TABLE II
EXPERIMENTAL PARAMETERS

Storage space SS	16KB
Maximum value of a cell	7
No. of hash functions K	5
No. of cells to delete P	10

captured if we *weight* the occurence of the false positives and false negatives with the importance of the element for which the FP/FN occurs. Thus, given n queries, we define weighted-false positive rate (wFP) and weighted-false negative rate (wFN) as shown below: $wFP = \sum_{k=1}^{n} \beta \times imp_k$ where β=1 if Q_k generates a FP, otherwise β=0. $wFN = \sum_{k=1}^{n} \beta \times imp_k$ where β=1 if Q_k generates a FN, otherwise β=0.

FP/FN against importance: While wFP and wFN capture the overall effectiveness of a probabilistic data structure, they do not explain the effect of the parameters on individual data elements. We wish to observe the performance of the IBF against the importance of the data elements. Suppose the

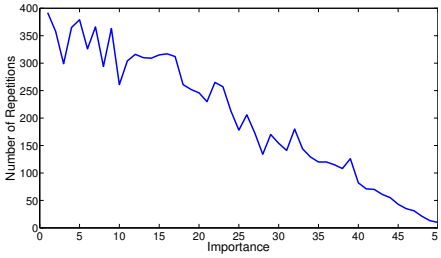

Fig. 14. Data Distribution: Synthetic Data 3

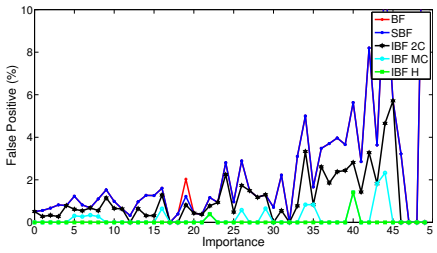

Fig. 15. False Positives: Synthetic Data 3

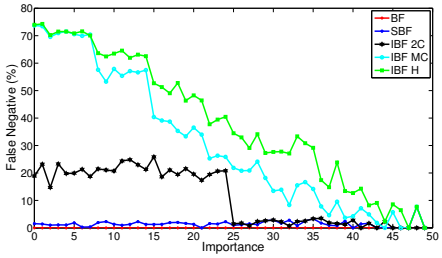

Fig. 16. False Negatives: Synthetic Data 3

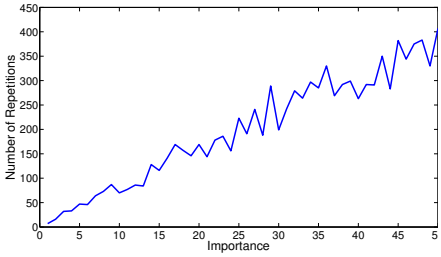

Fig. 17. Data Distribution: Synthetic Data 2

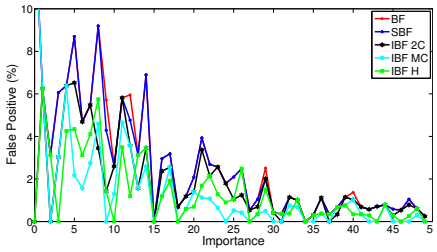

Fig. 18. False Positives: Synthetic Data 2

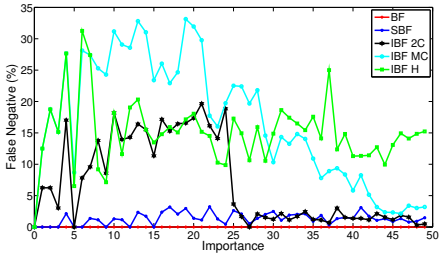

Fig. 19. False Negatives: Synthetic Data 2

importace values of the elements are integers in the range 1.....Z. Then we define *Importance rate arrays(I)* for FP (I_{fp}) and FN (I_{fn}). $I_{fp}[i] = \frac{\sum_{imp=1}^{Z} \beta}{\sum_{imp=i}^{Z} 1}$ where β=1 if Q_k generates a FP, otherwise β=0. $I_{fn}[i] = \frac{\sum_{imp=1}^{Z} \beta}{\sum_{imp=i}^{Z} 1}$ where β=1 if Q_k generates a FN, otherwise β=0. The i^{th} entry of the array I_{fp} thus gives the false positive rate among elements having importance i. Similarly, the i^{th} entry of the array I_{fn} gives the false negative rate among elements having importance i. This is our primary metric of comparison.

C. Comparison among Bloom filters

	Combination
BF	*M-insert with M=1*
SBF	*M-insert + Random-delete*
IBF_{2C}	*M/2-insert + Random-delete*
IBF_{MC}	*Imp-insert + Random-delete*
IBF_H	*Imp-insert + Probablistic-delete-val*

TABLE III
COMBINATIONS OF INERT AND DELETE PROCEDURES FOR THE IMPORTANCE-AWARE BLOOM FILTER.

Tab. IV-C shows different Bloom filter data structures we consider for comparison. IBF_H is another variant of IBF, where insertion is done similar to IBF_{MC}, but the deletion is performed with respect importance values mapped to cell, instead of random delete. We call the deletion scheme as *Probabilistic Delete Val* where the probability of choosing a cell is inversely proportional to the *value of the cell itself*, except cells having 0 value. We use this algorithm, so that elements with high importance are not evicted, thus improving the false negative rates.

D. Data sets:

Synthetically generated data set: We generated synthetic data, with {10%,30%,50%} of the elements repeating, and the importance of the elements randomly generated between 1 and 50. In order to determine what kinds of data distributions IBF will be most useful on, we generated three kinds of data distributions. In the first distribution, each element occurs in the data stream with equal probability, irrespective of the importance (See Fig. 11). In the second and third distribution, the probability of occurence of an element is directly (Fig. 17) and inversely (Fig. 14) proportional to its importance respectively. Thus, in the second dataset, important elements occur more frequently, and in the third one they occur less frequently.

Real-world data set: The real world dataset that we considered to perform our experiments was taken from [24]. It contains the traces for the video accessed from YouTube, a popular video-sharing website, for a campus network over several days. We used the IDs of the videos as our data elements, and a function of the video length as the importance. Such a scenario might be conceivable for a proxy server that buffers videos. As the overhead for fetching a longer video from the network is higher, the penalty for false negatives is high. Hence, we assign higher importance to longer length videos.

E. Results

We now explain the behavior of Bloom filters in Tab. IV-C by giving stream of elements from synthetic data set, as well as youtube vidoe data set as input. We calculate weighted false positive and weighted false negative, along with the FP and

FN for elements with a particular importance values, for all importance values.

Youtube data set: In our first set of experiments, we assigned the importance value for videos directly proportional to its length, scaling the importance values to the range 0-49. However, due to the presence of a few outlier of extremely long length, this distribution is skewed. Thus we assigned the importance value proportional to importance for the bottom 90 percentile in the range 0-49, and capped the importance value of the very long videos at an importance value of 50. This leads to the distribution of importance amongst elements as shown in Figure 5. Figures 6 and 7 show the false positive and false negatives for this dataset. Note that, IBF has 15% less false positives for almost all data items in comparison to SBF. At the same time, IBF has the same or lower false negatives for items with higher importance. That is IBF has lower false positives and lower false negatives for items with higher importance. Notice that in both Imp-Insert and $\frac{M}{2}$-Insert, a trend of decreasing FN rates with increasing importance can be observed. This is consistent with what we expect from the theory developed in Section III. This result on real data set confirms our belief that IBF is indeed a practical solution to handle indexing and membership queries while taking into account importance values of the data items.

In our second set of experiments, the importance value was assigned based on the percentile of the video length. Thus, there are an equal number of videos for all importance values. The distribution, false positives and false negatives are shown in Figures 8, 9 and 10 respectively. The results are similar to those above, with both M/2 Insert and Imp-Insert distinctly performing better than SBF to index data items with higher importance.

In Tab. I, we show wFP and wFN values for both sets of experiments. For dataset 1, IBF-MC (for example) has several fold lower FP and wFP values than SBF. SBF has wFP of 26% whereas IBF has wFP of only 5%. At the same time, the increase in false negatives marginal, SBF has wFN of 1% whereas IBF has wFN of 5%. Similar behavior can be observed for dataset 2. This shows that IBF captures data semantics well, and results in low wFP and wFN.

In all experiments $IBF-H$, which is a variant of $IBF-MC$ behaves almost similar to $IBF-MC$. Our objective was to evaluate a heuristic IBF deletetion scheme where he deletion which is proportional to importance values mapped to cells. However, this deletion scheme does not seem to be effective in comparison to random delete operation. We wish to investigate this behavior in future.

Synthetic data set: Fig. 12 and Fig. 13 show false postive and false negative for different importance values with respect to distribution of items in Fig. 11. We can observe that with respect SBF, IBF has lower false positives for almost all the importance values, and at the same time, it has about the same or lower false negatives for items with more importance. This matches with the overall goal we had set initially. When important elements occur less frequently, however, as we observe in Figures 14, 15 and 16, BF and SBF are not suitable to keep low FP and low FN for more important elements in the data stream. As seen in Fig. 15 and Fig. 16, FP rates go up as importance increases. This is undesirable behaviour of elements with higher importance. Notice, however, that IBF-MC, results in low false positives, and low false negatives for elements with high importance. IBF-H behaves similarly, but with higher false negatives than IBF-MC. IBF-2C shows clear distinction between 2 classes - those of higher and lower importance in false negatives. For importance values greater than 35, IBF has about 4% lower false negatives, and about 8% lower false positives than SBF.

Now, as observe in real data sets, it is expected that there are comparatively less number of items with higher importance. However, it is interesting to see how IBF behaves when there are more number of important items with higher importance. With resepct to Fig 17, Fig. 18 and Fig. 19, when important elements occur more frequently, IBF performs as good as SBF in maintaining false positives and false negatives for higher importance values. Thus, IBF can handle different data distributions, and still achieve our goal of capturing data semantics.

F. Theory verification: expected number of '0' cells

To verify the results of our experiments with that of theory, in Fig. 20, we plot the number of zeros observed in each variant of Bloom filter with respect to number of iterations of inserting data items. We feed the same parameters (for example, in this case $max = 7$, $K = 5$, $P = 10$) to our theoretical models which then give the expected fraction of empty cells analytically, and then confirm that indeed the two values are similar. Further, from Fig. 20, it can be seen that BF eventually gets 'full' with increasing number of iterations, whereas SBF and IBF stabilize to a particular value. This confirms the stable property of IBF through experiments.

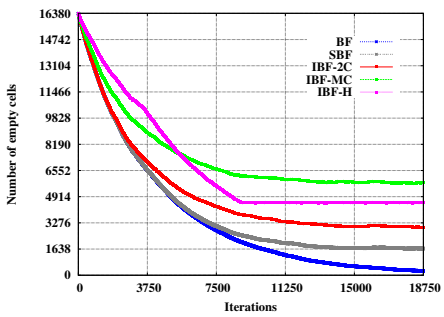

Fig. 20. Number of cells with value zero

V. Related Work

Load shedding [29] technique is usually performed when there is limited memory to store the streaming data. In such techniques [30], [27], [28], older tuples are evicted based on some load shedding scheme to make way for new data items, and then the entire data item is stored in memory. Our work is different because we do not store the entire data in memory, instead we use hash functions to represent it in a Bloom filter. It is easy to see that large data items will quickly fill up the memory, hence a load shedding scheme will not be useful, especially when window sizes are also large. An advantage, however, over our approach is that load shedding does not introduce false negatives.

Data-importance is studied for join queries in [21], but tuple eviction is based on load shedding. The Stable Bloom filter (SBF) discussed in [16] indexes streaming data, but the scheme does not consider data-importance. Moreover, SBF deletes cells at random, which we believe is too simplistic. Our work provides a more complete solution, motivated by the fact that streaming data items have importance semantics.

Counting Bloom filters [20] and Spectral Bloom filters [15] are not directly applicable to our problem, because they assume that the set cardinality is known (and hence the FP an FN rates can be optimized by tuning other parameters). We make no such assumption, and provide a solution that can deal with unbounded sets, and fluctuating data distributions. Aging Bloom filter [32] is an interesting idea to remove stale data when indexing a static set in the filter, however, this work also does not address data-importance, and assumes that older data ages (or becomes less important over time). We do not make any such assumption; in fact, all data items in the landmark window are relevant, and the data-importance is the factor that guides the insertion and deletion of items in the filter. Time-decaying Bloom Filter [14] is another work that maintains the frequency count for each item in a data stream, and the value of each counter decays with time. Again, this work also assumes that items in a data stream maybe be time-sensitive, and does not address data-importance while maintaing the time-decaying counters. It is also worth noting that, as compared to all previous work, we measure a importance-centric metric: the weighted FP and FN rates, which is a more realistic way of assessing quality of the query result.

VI. Conclusion

In this work, we presented IBF, a probabilistic data structure for indexing and querying a stream of data items with different importance values. IBF has several desirable properties, for instance, the number of empty cells in IBF remain constant, there exists an upper bound on the false positive rate irrespctive of the size of the stream, and it has $O(1)$ time complexity to index an item and answer a query. Importantly, IBF produces lower false positives and lower false negatives for important data items. We evaluated IBF on synthetic as well as a real-world data set. Our results show that, indeed IBF is effective in taking data semantics into account. Thus, we believe IBF is an efficient and promising solution to index a stream of data items. As part of future work, it is interesting to capture the relationship between max, K and P analytically. Also, it would be interesting to deploy IBF in an online environment (for example, as a fast cache in routers) and analyse its behavior.

References

[1] Coral8, inc. homepage, http://www.coral8.com, 2007.

[2] Etrade financial, http://etrade.com, 2007.

[3] Nyse euronext, http://www.nyse.com/, 2007.

[4] Pacific tsunami warning center, http://www.prh.noaa.gov/pr/ptwc/, 2007.

[5] Scottrade homepage, http://scottrade.com, 2007.

[6] Tao project, http://www.pmel.noaa.gov/tao/, 2007.

[7] Yahoo news, http://news.yahoo.com/rss, 2007.

[8] M. M. Astrahan and et al. System r: Relational approach to database management. *TODS*, 1(2):97–137, 1976.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of Principles of database systems*, 2002.

[10] B. H. Bloom. Space/time Trade-offs in Hash Coding With Allowable Errors. *Commun. ACM*, 13, July 1970.

[11] D. Carney and et al. Monitoring streams - A new class of data management applications. Technical Report CS-02-04, Department of Computer Science, Brown University, Feb. 2002.

[12] S. Chandrasekaran and et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proceedings of CIDR*, 2003.

[13] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of ACM SIGMOD*, pages 379–390, 2000.

[14] K. Cheng, L. Xiang, M. Iwaihara, H. Xu, and M. M. Mohania. Time-decaying bloom filters for data streams with skewed distributions. *Research Issues in Data Engineering, International Workshop on*, 0:63–69, 2005.

[15] S. Cohen and Y. Matias. Spectral bloom filters. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 241–252, New York, NY, USA, 2003. ACM.

[16] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *ACM SIGMOD'06*.

[17] F. Deng and D. Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, 2006.

[18] eBay. ebay homepage, http://pages.ebay.com, 2007.

[19] S. S. P. Engine. Streambase systems inc. homepage, http://www.streambase.com/, 2007.

[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *IEEE/ACM Transactions on Networking*, pages 254–265, 1998.

[21] D. Kulkarni and C. V. Ravishankar. ijoin: Importance-aware join approximation over data streams. In *SSDBM '08: Proceedings of the 20th international conference on Scientific and Statistical Database Management*, pages 541–548, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of ICDE*, 2002.

[23] R. Motwani and et al. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of CIDR*, 2003.

[24] U. T. Repository. http://skuld.cs.umass.edu/traces/network/readme-youtube.

[25] P. Seshadri, M. Livny, and R. Ramakrishnan. Sequence query processing. In *Proceedings of ACM SIGMOD*, pages 430–441, 1994.

[26] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. In *Proceedings of USENIX Annual Technical Conference*, pages 13–24, 1998.

[27] N. Tatbul. Qos-driven load shedding on data streams. In *EDBT '02: Proceedings of the Worshops XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers*, 2002.

[28] N. Tatbul, U. Çetintemel, and S. Zdonik. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In *International Conference on Very Large Data Bases (VLDB'07)*, Vienna, Austria, September 2007.

[29] N. Tatbul, U. Cetintemel, S. Zdonik, M. Chemiack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of VLDB*, 2003.

[30] N. Tatbul and S. Zdonik. Window-aware Load Shedding for Aggregation Queries over Data Streams. In *International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, September 2006.

[31] K. Trivedi. Probability and Statistics with Reliability, Queuing, and Computer Science Applications. *John Wiley and Sons, New York, 2001*, ISBN number 0-471-33341-7.

[32] M. Yoon. Aging bloom filter with two active buffers for dynamic sets. *IEEE Transactions on Knowledge and Data Engineering*, 22:134–138, 2010.