Dynamically Adaptive Count Bloom Filter for Handling Duplicates in Data Stream

Senthamilarasu.S [#1] and M.Hemalatha[#2]
[#1, 2] Department of Computer Science, Karpagam University
Coimbatore, TamilNadu, India.
[1] s.a6asu@gmail.com
[2] csresearchhema@gmail.com

***Abstract*— Identifying and removing duplicates in Data Stream applications is one of the primary challenges in traditional duplicate elimination techniques. It is not feasible in many streaming scenarios to eliminate precisely the occurrence of duplicates in an unbounded data stream. However, existing variants of the Bloom filter cannot support dynamic in both filter and counter together. In this paper we focus on eliminating the duplicates by introducing the dynamic approach on both the size of the counter and the bloom filter. The basic idea is instead of keeping either the size of counter or filter static in this paper we improvised the performance of by considering both the counter and the filter size as dynamic. In addition necessary algorithms for new item insertion, querying on the membership and deleting the duplicates are also proposed. we showed that a the proposed approach guarantees the superiority in terms of accuracy and time efficiency and reduces the considerable amount of false possible rate than the existing approaches.**

Keywords- Bloom Filter, Data stream, Set membership Query and Hash Functions.

I. INTRODUCTION

Due to High Evolution of information from different fields like business, internet, e-commerce, etc. It is more difficult to handle huge amount of data. To reduce this dependency, data mining and Machine learning algorithm has been used to model the streaming data. A data stream is a massive uncontrolled and an unstructured sequence of data elements. These elements are generated continuously at a rapid rate. In real time, all data elements cannot be processed by the main-processing operation because the data are generated anywhere and received by end node for evaluating (processing and storing) with uninterrupted transmission. It is usually desirable for decision makers to find out the valuable information from the continuous stream such as stock market, health care, traffic analysis etc. Recently, the data generation rates increase and faster than ever before. This rapid generation of relentless streams of data has challenged our resource capacity, computation and communication capabilities of the computing systems.

The importance of data stream proposes new models, systems and techniques to reduce the latency, memory, error rates and increasing accuracy, computing power and more to identify patterns, query result, hidden information from continuous stream. The Data stream management System (DSMS) maintains huge volume of data that provides the query services to user with effective and reliable manner. The data handling is the core activity where the real computation is handled. It is frequently analyses the data from various sources. DSMS contains an unimaginably vast amount of digital information which is getting ever vaster ever more rapidly.

The problem involved handling volume of data is, Reliability and manageability might be in tension with each other - reliable findings might suggest the need for many data sources, with the problem of data overload. The data stream contains noisy, replicability and generalisability is problematical to get query results with more accuracy and data should not be compact. Even small parts of the data are complex. Handling data is a conflict and an inherently inexact task and also loses a variety of different types of data. However, the growth and advantage of the technology cause a new problem. For example Since an RFID tag is detected without contact, if an RFID tag is within a proper range of an RFID reader, the RFID tag will be detected whether we want to or not. Therefore, if RFID tags stay or move slowly in the detection region, much unnecessary data (i.e., Duplicate RFID data) will be generated [1].

Data duplication is a well-known portion of data stream processing, is a sequence of bytes across data comparisons. Deduplication is ideal for highly redundant operations like backup, which requires repeatedly copying and storing the same data set multiple times for processed within the time period. Eliminating redundant data can significantly shrink storage requirements and improve efficiency of bandwidth. Because, primary storage has cheaper over time. Detecting and removing duplicate is important techniques in data monitoring and analyzing. Information representation and processing of queries are associated problems that encompass the core issues in many computer applications. Representation suggests organizing information primarily based on a given format and mechanism such that information is operable by a corresponding

technique. The processing of queries involves diminishing decisions based on whether an item with a specific attribute value belongs to a given set [11].

A standard Bloom filter (SBF) is a space-efficient data structure for representing a set and answering queries within a constant delay [12]. The SBF has been modified and improved from different aspects for a variety of specific problems. The SBF has been modified and improved from different aspects for a variety of specific problems. The most important variations include compressed Bloom filters [2], , distance-sensitive Bloom filters [3], Bloom filters with two hash functions [4], space code Bloom filters [5], spectral Bloom filters [6], generalized Bloom filters [7], Bloomier filters [8], Bloom filters based on partitioned hashing [9] and counting Bloom filters [10]. These implementations have extended the basic approach of bloom filter (BF). The BF used in the many applications and it can achieve high efficiency space and query accuracy. A Static Bloom filters performance only in static set. While dealing dynamic set then span the dataset by sliding window. Performing Sliding windows operation decayed element is removed from a dynamic set while detecting or identifying duplicates based on the distinct elements. Each element is stored and identified the streaming data, which anyone is matched with a particular element, the corresponding bits in the associated BF, as they can be shared by other elements and reset operations can introduce false negatives. Several membership queries are represented in a sliding window and also several BF-based schemes have been proposed for reducing the deduction in monitoring the elements to improve the scalability. The disadvantage is that dynamic Bloom filters do not outperform Bloom filters in terms of false match probability when dealing with a static set with the same size memory. The undetectable incorrect deletion of false positive items and detectable incorrect deletion of multi-address items are two general causes of false negative in a Bloom filter.

II. Related Work

Duplicate detection was considered by, Bloom filter supports the membership queries with static set and decrease the false positive probability to a sufficiently low level. In dynamic Bloom filter supports static as well as dynamic dataset and it performs the insert, membership queries, delete and union operations. It could control the false positive probability at a low level by expanding its capacity as the set cardinality increases without the upper bound. [11]. Most of the Bloom filter returns only false positive but never returned false negative for reducing redundancy. Inspiration of false negative was fully exposed with the bloom filter, which increases the ratio of bits set to a value larger than one without decreasing the ratio of bits set to zero. CBF decreases the number of exposed false negative items without increasing the probability of false positive [13]. Unidentifying and removing duplicates might affect the accuracy and the prediction results might wrong while taking decision based on duplicates. Removing duplicates using decaying bloom filter with the extension of counting bloom filter that effectively remove elements and continuously place arrived data over sliding windows and it produce the false positive errors ,to reduce the time complexity and increase the accuracy[14]. Remove redundancy or duplication in the data stream presented novel reservoir sampling based Bloom filter (RSBF) technique, and combined with basic concepts of reservoir sampling and Bloom filters for approximate detection of duplicates in data streams [15]. Traditional duplicates elimination techniques are not applicable to many data stream applications. So we finding some properties of Stable Bloom filter analytically and tight with upper bound false positive rates to solve this problem [16]. solves the detecting duplicates in data stream as to finding a positive frequency element in a stream given in frequency and or frequency updates where the sum of all frequencies is positive rates[17]. Deals with the finding and removing duplicate records in data warehouse as using divide and conquer method for matching records within the cluster for improving efficiency [18].

III. Problem Statement

Bloom Filters provide space-efficient storage of sets at the cost of a probability of false positives on membership queries. The size of the filter must be defined a priori based on the number of elements to store and the desired false positive probability, being impossible to store extra elements without increasing the false positive probability. This leads typically to a conservative assumption regarding maximum set size, possibly by orders of magnitude, and a consequent space waste.

IV. Proposed Methodology

The Data Stream Management is a very challenging field of handling the dynamic dataset. In our previous work the problem of load shedding (i.e.) Overflow of incoming data stream is overcome by proposing the window based aggregate function. While collecting dataset from various sources may leads to duplication of data entries. This paper focuses on identifying the duplicate entries and removing them. In this proposed approach we formulated the dynamically adaptive count bloom filter which maintains both the counter and the filter size to be handled dynamically. In the earlier approaches either counter or the filter is kept dynamic but in our case we considered both of their size to be dynamic. Because while the data stream is unbounded just keeping the filter or counter to be dynamic still leads to the problem of overflow so depending on the size of the dataset the size of the counter should also be updated.

A. *Hash Algorithms*

In this work, to address problem of handling duplicate elements over data stream using filter with extension of Bloom Filter and It is also discover , what are all the similar elements? The following encrypted hash functions are used to discovering duplicate elements from the stream such as MD5, SH1, and RIPEMD. These Hash functions are accessed 128-bit and 512-bit version and which had been found to have questionable security and also used for data integrity. May Higher bit (256/512) versions diminish only chances of accidental collision.

The **MD5 Message-Digest Algorithm** is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. It has been utilized in wide variety of applications such as security, data integrity and so on. The following steps are processed in the MD5.

- Suppose if the input message has m bits,
- The input message has to be broken into chunks of 512 bit blocks
- To convert the size of the message padding is performed by first a single bit is appended at the end of the message followed by as many zeros as are required to make the length of the message divisible by 512.
- The remaining bits are filled up with a 64-bit integer representing the length of the original message, in bits.

The 4 state variables are used in md5 algorithm. These variables are sliced and diced and are (eventually) the message digest the main part of the algorithm uses four functions to thoroughly goober the above state variables. Those functions are as follows:

F(X,Y,Z) = (X & Y) | (~(X) & Z)

G(X,Y,Z) = (X & Z) | (Y & ~(Z))

H(X,Y,Z) = X ^ Y ^ Z

I(X,Y,Z) = Y ^ (X | ~(Z))

Where &, |, ^, and ~ are the bit-wise AND, OR, XOR, and NOT operators. These functions, using the state variables and the message as input, are used to transform the state variables from their initial state into what will become the message digest.

SHA-1

SHA stands for Secure Hash Algorithm.SHA-1 produce a 160-bit message based on principles similar to MD4 and MD5. The following steps are describing SHA1 Algorithms.

- Padding
  - Pad the message with a single one followed by zeroes until the final block has 448 bits.
  - Append the size of the original message as an unsigned 64 bit integer.
- Initialize the 5 hash blocks (h0,h1,h2,h3,h4) to the specific constants defined in the SHA1 standard.
- Hash (for each 512bit Block)
  - Allocate an 80 word array for the message schedule
    - Set the first 16 words to be the 512bit block split into 16 words.
    - The rest of the words are generated using the following algorithm
- word[i3]
  - XOR word[i8]
  - XOR word[i14]
  - XOR word[i16]
- then
- Rotated 1 bit to the left.
  - Loop 80 times doing the following. (Shown in Image1)
    - Calculate SHA function () and the constant K (these are based on the current round number.
    - e=d
    - d=c
    - c=b (rotated left 30)
    - b=a
    - a = a (rotated left 5) + SHA function() + e + k + word[i]
  - Add a, b, c, d and e to the hash output.
- Output the concatenation (h0, h1, h2, h3, h4) which is the message digest.

RIPEMD

RIPEMD-160 was developed in Europe as part of RIPE project in1996. It uses 2 parallel lines of 5 rounds of 16 steps. It creates a 160-bit hash value. It is slower, but probably more secure, than SHA.

- pad message so its length is 448 mod 512

- append a 64-bit length value to message
- initialize 5-word (160-bit) buffer (A,B,C,D,E) to
- (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
- process message in 16-word (512-bit) chunks:
  - use 10 rounds of 16 bit operations on message
- block & buffer – in 2 parallel lines of 5
  - add output to input to form new buffer value
- Output hash value is the final buffer value.

B. *Scalable Bloom Filter*

In this proposed work we have proposed a new variant approach on the usual counter filter which is known as scalable counter filter (SCF). In the traditional Counter bloom filer (CBF) the data structure was usually static. The size of the CBF was fixed over time which actually consists of a static dataset.

The existing approaches suffers from two major limitation

- Counter overflow during the insertion of elements
- Allocation same bit length for each counter may result in memory waste

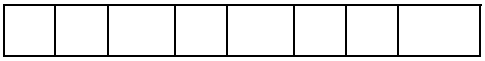
The proposed method overcomes these two problems by adapting the concept of scalable bloom filter technique as the variant of counter bloom filter. In this paper the filter used for finding the duplicate elements in the data stream is dynamic bloom filter. Once the current active standard bloom filter was filled a new bloom filter will be created with varying size. To cope up with such dynamic filter the counter was also made dynamic by the concept of scalable growth.

The estimation of the set size that is to be stored in a filter may be wrong, possibly by several orders of magnitude. We may also want to use not much more memory than needed at a given time, and start a filter with a small size. Therefore, a SBF should be able to adapt to variations in size of several orders of magnitude in an efficient way Proposed Adaptive Dynamic Scalable counter bloom filter. Setting the size of the bloom filter priori is a major Caveat in managing the data stream, because by applying an upper bound on the expected false positive rate and estimating the maximum capacity (n) is required. In most of the real time cases the over dimensioning may result due to the unknown number of elements to be stored. Particularly in bloom filters the elements are added and queried independently based on the time factor. This would cause wastage in memory space. The construction of our approach is mainly based on a dynamic matrix with s x m bits. The number of bits to be stored is m. We used three different hash functions to index the position of the elements. The hashing functions used in our approach are MD5, SHA 1 and RIPEMD.

The filer and counter are treated as slices to overcome the problem to collision in the extreme cases of voluminous data stream.

$$h_1(x) = h_2(x) = h_3(x)$$

This will result in error prone rate of false positive to be high.



1 m

The system parameters used in this proposed work filter size is represented by m. The number of hash function used are k in our case k = 3. The number of slices is represented by s. The prediction of filter size may be wrong possible due to different reasons. At the same time using more memory than needed at a given time should be avoided. Therefore, the filter should be able to adapt to variations in size efficiently.

The estimation of the set size that is to be stored in a filter may be wrong, possibly by several orders of magnitude. We may also want to use not much more memory than needed at a given time, and start a filter. Suppose if the filter was m bits in length then the counter should also be created with m bits length. If the filter size varies the counter size should also vary. The filter maintains the scalable growth exponentially. Initially the filters are initialized to zero. When the new elements arrive for insertion the k hash functions are applied on it and its corresponding position is mapped in the filter and the values are set to 1. At the same time the counter is incremented by one. The expected growth rate is represented by s. error probability tightening ration is represented by r. The false positive probability is represented by

$$p = p^k$$

$$\text{Where } p = \frac{\text{no. of bits set}}{\text{Slice size}}$$

if an n element has been inserted then fill ratio $P = \frac{n}{M}$

if n= 3 and m = 5 then error probability will be .216%

The probability of given set to be zero after inserted n elements are

$$\left(1-\frac{1}{m}\right)^{n}$$

To set a specific bit in a slice is set after n elements are

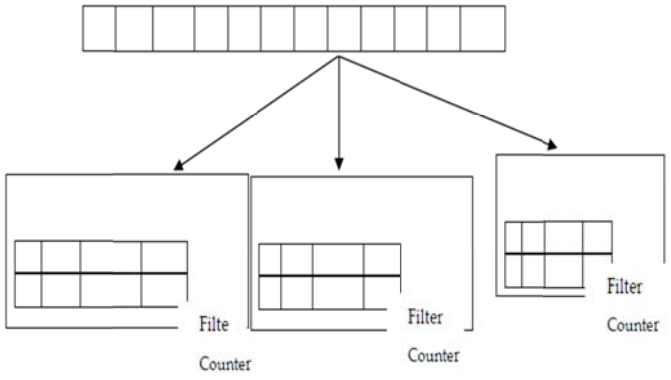
$$p = 1 - \left(1-\frac{1}{m}\right)^{n}$$

A variant of Bloom filters [2], which we adopt in this paper, consists of partitioning the *M* bits among the *k* hash functions, thus creating *k* slices of *m = M/k* bits. In this variant, each hash function *hi (x)*, with $1 \leq i \leq k$, produces an index over *m* for its respective slice. Therefore, each element is always described by exactly *k* bits, which results in a more robust filter, with no element especially sensitive to false positives. The no of counters for each filter depends on the M.

If M = 12, k = 3 then

m = M/k = 12/3 = 4 bits

No.of counters to be used for one slice is= M bits = 12.



For *M* = 12 and *k* = 3 a filter would have 3 slices with 4 bits in each. Insertion of one element, the resulting configuration would have exactly one bit set in each slice. Each slice is depicted here in a row For each slice with 4 bits each a separate array of counter is used with same 4 bits for each row. Each slice consists of both filer and the counter. When a new filter is added to a ADSCBF, its size can be chosen orthogonally to the required false positive probability. A flexible growth can be obtained by making the filter sizes grow exponentially. We can have a ADSCBF made up of a series of filters with slices having sizes m_0, m_0s, m_0s^2,.., m_0s^{l-1}.

Given that filters stop being used when the fill ratio reaches 1/2, This geometric progression allows a fast adaptation to set sizes of different orders of magnitude. A practical choice will be *s* = 2, which preserves *mi* as a power of 2, if *m*0 already starts as such; this is useful as the range of a hash function is typically a power of 2.

C. *Representation and Membership Queries of Proposed Work*

In the propose approach multiple filters are used. But only one filter is active at a time and others are inactive. During insertion operation the amount of elements inserted into each filter is tracked. Initially a single filter is created and represented as slices. The number of slice to be partitioned is depending on the number of hash function used. In our case we use three hash functions so the slices are represented based on it. In the initialization process the filter values are set to zero and the filter is denoted as active filter. When the new element arrives it was hashed with MD5, SHA1 and RIPEMD and they are mapped into the corresponding index of the slice. When a filter gets full due to the limit on the fill ratio, a new filter is added with varying size.

The ADSCBF starts with one filter with k_0 slices and error probability P0. When this filter gets full, a new one is added with k_1 slices and $P_1 = P_0r$ error probability, where r is the tightening ratio with $0 < r < 1$. At a given moment we will have l filters with error probabilities P_0, P_0r, P_0r^2,..., P_0r^{l-1}.

The compounded error probability for the ADSCBF will be:

During insertion, the first BF that has its element counter less than the given threshold is selected as the active BF. If such an active BF cannot be found, a new BF is created and designated as the active BF. The element is then inserted into the active BF.

Duplicate element detection is an important problem, especially pertaining to data stream processing. In the general case, duplicate detection in an unbounded data stream is not practical in many cases due to memory and processing constraints. To check the existence of an element in the filter the input element is hashed with three different hash functions and check the counter value of each of it. If the counter value is greater than zero for all the k bits then it indicates the duplication entries and it eliminates the new element to avoid the duplication. The duplication identification is continued till it reaches the end of all the filters. If it doesn't find such an entry then it considers that the element is not present in the available filter.

To find the existence of element x in order to avoid the duplicate entry the following algorithm is used

Query (*x*)
Require: *x* is not null
For m: 1 to n do
    c = 0
For i *:* 1 to *s* do
    j**=1**
If $ScalableBF_{M.}\text{Slice}_i[hash_j(x)] = 0$ **then**
    Break
Else
    counter <- *counter* + 1
If *c* = *k* **then**
    Return true
else
    Return false

In this query algorithm to check whether the element x is already existed in the filter the three hashing algorithm is applied on the element x on the different slices of the filter and if all the hashing output is zero then the counter value c will be zero which will not be equivalent to k (i.e) no of hash function. So the output returns false to indicate that the element is not present in the filter. If the element in present in the filter all the three hashing index position will hold the value of 1 each time a set bit is found means the counter c will be incremented. If all the three hash position consists of set bits the result returns the output of true when the c is equal to the k value. n represents number of filters and s represents no of slices for each filter and the looping is performed in each filter slice to check the availability of query element .

V. EXPERIMENTAL RESULTS

The experimental result shows that the performance of the proposed approach has significantly increases the accuracy performance and decreases the false positive rate by maintaining both the size of the filter as well as the counter to be dynamic by adapting the concept of the scalable growth. In this size of the filter slice are varied exponentially. It overcomes the problem of memory usage and avoids the duplication entries in data stream.

The accuracy of the filter is measured by the accurate representation of a data element. This can be defined as a data element x has an accurate representation if the minimum value stored in the k counters matches the real number of times x has been inserted in the data set.

The Table 1 shows that the accuracy rate, false positive rate and time taken by four different approaches namely bloom filter, window based approach, decaying bloom filter Proposed Scalable bloom filter. The result shows that our proposed approach has the highest accuracy rate of 96.7% , false positive rate has been considerably reduced to 7.3% and the time taken for the process 12 ms. By maintaining the size of both filter and counter dynamic it will overcome the problem of memory usage and handling continuos voluminous dataset.

TABLE I

COMPARISON RESULT OF EXISTING AND PROPOSED APPROACH OF SCALABLE BLOOM FILTER

Technique	Accuracy	False Positive Rate	Time (ms)
Bloom filter	91.6	12	20
Window Based Approach	89.76	19	25
Decaying Bloom Filter	93.5	10.8	17
Proposed Scalable Bloom Filter	96.7	7.3	12

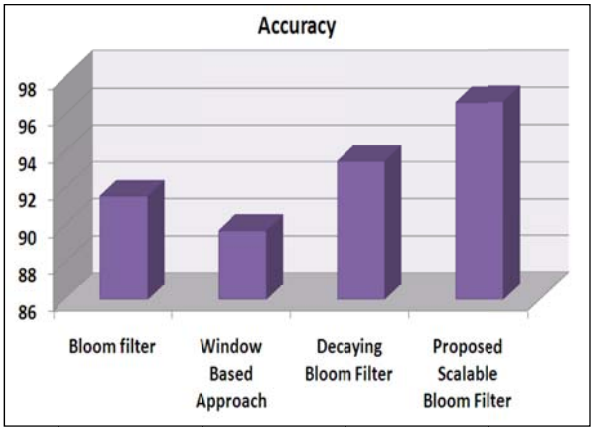


Fig. 1 Performance result of various approaches based on Accuracy

As shown in Fig. 1, our proposed dynamic adaptive scalable Bloom filter increases the accuracy rate rather than our existing filters, which is showed in the above table. Our proposed executes faster than our existing approach.

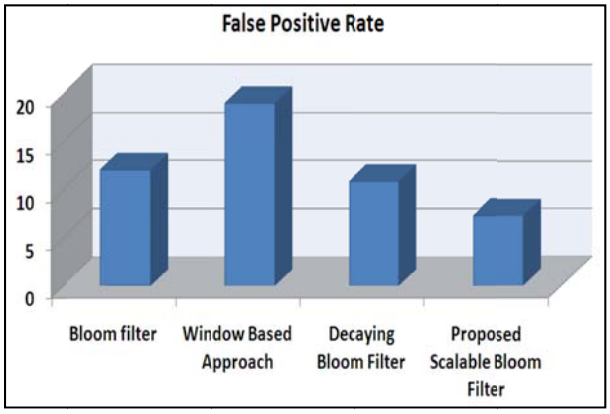


Fig. 2 Performance result of various approaches based on false positive Rate

As shown in Fig. 2, for each existing Filter approach i.e. Bloom filter, Windows based approach and Decaying Bloom filter, the ratio increases as the false match probability increases; but our proposed Scalable Bloom filter reduced the false positive rate for increasing efficiency. The existing filters never consumes more memory than Scalable bloom filter and save more memory as the false match probability decreases.

In figure 3 represents the time taken by each techniques and our proposed approach reduces the time complexity by introducing the dynamicity in filter size as well as counter size also.

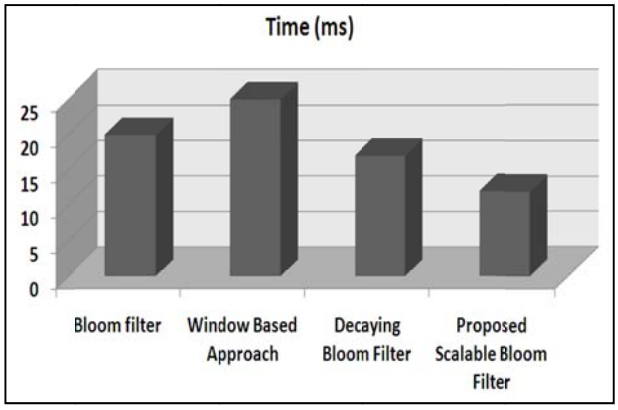


Fig. 3 Performance result of various approaches based on Time

VI. CONCLUSION

For static dataset bloom filter is an excellent data structure to handle the membership query with fixed cardinality. But it does not support the dynamic dataset. In real cases data stream are only the dynamic dataset. So to deal with such real time cases bloom filter is not recommended. The deletion process is also not supported in the bloom filter and false positive rate is also high. So the necessity arises to deal with the dynamic dataset with varying size both in filter and counter in order to identify the duplicate entries in the data stream. The proposed method not only inherit the advantage of bloom filter but it have enhanced features than Bloom filter by adapting the scalability based counter and the designing the varying size filter instead of fixed size filters. The false positive rate of the proposed approach has been decreased noticeably while comparing the existing ones.

ACKNOWLEDGMENT

We thank Karpagam University for the motivation and encouragement for giving me the opportunity to do this research work as successful one.

REFERENCES

[1] Bai,Y., F. Wang, P. Liu,“ Efficiently filtering RFID data streams”, *VLDB Workshop on Clean Databases*. 2006.
[2] Mitzenmacher, “Compressed Bloom Filters,” IEEE/ACM Trans.Networking, vol. 10(5), 2002, pp. 604-612.
[3] Kirsch,A., and M. Mitzenmacher, “Distance-Sensitive Bloom Filters,” Proc. Eighth Workshop Algorithm Eng. and Experiments (ALENEX ’06), Jan. 2006.
[4] Kirsch,A. and M. Mitzenmacher, “Building a Better Bloom Filter,”Technical Report tr-02-05.pdf, Dept. of Computer Science,Harvard Univ., Jan. 2006.
[5] Kumar,A., J. Xu, J. Wang, O. Spatschek, and L. Li, “Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement,” Proc.23rd IEEE INFOCOM, Mar. 2004,.pp. 1762-1773.
[6] Cohen,S. and Y. Matias, “Spectral Bloom Filters,” Proc. 22nd ACM SIGMOD, June 2003,pp. 241-252,
[7] Laufer, R.P. , P.B. Velloso, and O.C.M.B. Duarte, “GeneralizedBloom Filters,” Technical Report Research Report GTA-05-43,Univ. of California, Los Angeles (UCLA), Sept. 2005.
[8] Chazelle,B., J. Kilian, R. Rubinfeld, and A. Tal, “The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables,” Proc. Fifth Ann. ACM-SIAM Symp. Discrete Algorithms(SODA), Jan. 2004.pp. 30-39.
[9] Hao,F., M. Kodialam, and T.V. Lakshman, “Building High Accuracy Bloom Filters Using Partitioned Hashing,” Proc. SIGMETRICS/Performance, June 2007,pp. 277-287,
[10] Fan,L., P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide Area Web Cache Sharing Protocol,” IEEE/ACM Trans. Networking, vol. 8(3), June 2000.pp. 281-293,
[11] Deke Guo, Jie Wu, Honghui Chen, Ye Yuan, Xueshan Luo,” The Dynamic Bloom Filters”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. , NO. , 2009.
[12] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. Commun. ACM, 13(7),1970,PP.422–426,
[13] Deke Guo, Yunhao Liu, Xiangyang Li, and Panlong Yang, “False Negative Problemof Counting Bloom Filter”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 22,(5),PP.651-664.
[14] Hong Shen and Yu Zhang,” Improved Approximate Detection of Duplicates for Data Streams Over Sliding Windows”, JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY,Vol. 23(6),pp.1-15.
[15] Sourav Dutta,, Souvik Bhattacherjee, Ankur Narang,” Towards “Intelligent Compression” in Streams: A BiasedReservoir Sampling based Bloom Filter Approach” In ACM, EDBT 2012, March 26–30, 2012.
[16] Fan Deng,Davood Rafiei,” Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters”in ACM, *SIGMOD 2006*, June 27–29.
[17] Parikshit Gopalan and Jaikumar Radhakrishnan,” Finding duplicates in a data stream”.
[18] Bilal Khan,Azhar rauf,.sajid H.Shah and shah khusro,”Identification and removal of duplicate records”. In world applied Science journal Vol.5 (13) pages 1178-1184.
[19] MyungKeun Yoon,” Aging Bloom Filter with Two Active Buffersfor Dynamic Sets”,IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,Vol.22(1),pp.34-38.
[20] Christian Esteve Rothenberg, Carlos A. B. Macapuna, F´abio L. Verdi, and Maur´ıcio F. Magalh˜aes,” The Deletable Bloom Filter: A New Member of the Bloom Family “, IEEE COMMUNICATIONS LETTERS, VOL. 14,(6),pp.557-559.

[21] Yi-Hsuan Feng, Nen-Fu Huang and Yen-Min Wu,"Efficient and adaptive statful replication for stream processing engines in high-quallity clusters ",IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 22 (11),pp.1788-1796.
[22] Hairulnizam Mahdinand Jemal Abawajy ,"An approach for removing redundant data from RFID data streams ", *Sensors* 2011, *11*, 9863-9877.
[23] Shen H, Zhang Y. ,"Improved approximate detection of duplicates for data streams over sliding windows", JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 23(6),pp.1-15.
[24] Mohamed A. Sharaf, Alexandros Labrinidis and Panos K. Chrysanthis,," Scheduling Continuous Queries in Data Stream Management Systems ", PVLDB '08, August 23-28, 2008, Auckland, New Zealand,pp.1526-1527.
[25] Michael Cammert,Ju¨rgen Kra¨mer, Bernhard Seeger, and Sonny Vaupel," A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 20(2),pp.230-245.

AUTHORS PROFILE

Dr. M. Hemalatha completed M.Sc., M.C.A., M. Phil., Ph.D (Ph.D, Mother Terasa women's University, Kodaikanal). She is Professor & Head and guiding Ph.D Scholars in Department of Computer Science at Karpagam University, Coimbatore. Twelve years of experience in teaching and published more than hundred papers in International Journals and also presented more than eighty papers in various national and international conferences. Area of research is Data Mining, Software Engineering, Bioinformatics, and Neural Network. She is a Reviewer in several National and International Journals.

S. Senthamilarasu, completed MCA, Pursuing Ph.D Research in Computer Science, under the guidance of Dr. M. Hemalatha, Professor and Dept. Computer Science at Karpagam University, Coimbatore, Tamilnadu. Presented two papers in national conferences and two International conference. Area of my Research is Data Stream in Data mining.