Enhancement of Precision on Cloud Data using Multi-Layered Bloom Filter

Saptarsi Saha and DVN Siva Kumar

Computer Science & Information Systems,
Birla Institute of Technology and Sciences, Hyderabad

***Abstract*—For economic savings and management flexibility, cloud computing has been widely utilised to maintain personal and management data. In recent years, it's a common practice to outsource data to the various cloud servers. To ensure relevant searches over the outsourced data, old traditional data utilization functions, such as plain text keyword searching, is impossible. We are using multi-keyword searching schemes over cloud data to solve this problem ~~by using effective and different operations.~~ The Traditional methods of solutions have high computational costs. Our scheme introduces a Multi-Layered Bloom filter approach which adds precision and improves the accuracy than the existing approaches. The proposed system supports insertion of keywords and keyword search operations in practical and proper ways by using a layered Bloom filter approach[1]. The false positive count of the proposed approach is lower when compared with other traditional techniques[2]. Experimental trails on real-world data sets reveal that the proposed approach performs satisfactorily in the terms of false positive count and precision.**

***Keywords*— Cloud, Computing, ~~Search~~, Keywords, Multi, BloomFilter, MultiLayer, Precision, Performance, Data**

I. Introduction

Cloud Computing has been booming a lot in recent years and it will keep doing so in the future too. Many people have realized the benefits that can be reaped from it. In this era of information explosion and high volumes of data, people are forced to manage a massive amount of data, increasing the management costs and losing efficiency. People, enterprises, and organizations may simply use cloud computing to solve this problem and increase productivity. This will offer on-demand network access to a shared pool of adjustable computing resources that is convenient, flexible, and ubiquitous[3]. To be specific, data owners and big businesses can outsource their data to cloud servers and their respective data centers to get access to these data as they want and when they want. Cloud computing provides a big breakthrough for people in terms of cost savings and administration flexibility.We also need to provide users with a fast and efficient way to search for their uploaded documents and datasets.With this in mind, we would like to propose a unique new data structure to solve our above-mentioned problem and improve the ~~efficiency and~~ precision of the system as a whole. If we use hashmaps or sets though it is time-efficient, we have a significant concern in the space complexity department as it will require O(n) space. In bloom filters, we make use of the bit array and more than one hash function. To determine whether a value x has already been visited, the bloom filter examines whether the value of the locations of bit arrays generated by all hash functions is true or false. Bloom filters do not save the objects themselves and consume less space than the theoretical minimum required to store the data accurately, hence they have an error rate. ~~To increase search efficiency and improve upon existing ones~~ Our novel strategy leverages the creation of an efficient approach based on Layered Bloom filters.The proposed method is capable of achieving a sub-linear search time.[4-6] and is also able to improve the precision and accuracy of the existing bloom filter schemes in use by reducing the count of false positives.

We present a number of keyword search techniques that can improve precision while also~~ reducing search time.~~ Our contributions are summarised here as the following:

(1) To boost search performance, a Layered Bloom filter technique is developed. This approach can achieve a search time that is sub-linear.In addition, our system outperforms other analogous schemes in terms of accuracy and precision.

(2) The results of the experiments conducted on a real-world data set reveal that our proposed strategies function well.

II. Related work

A. SEARCHABLE ENCRYPTION

A common method to protect user's data in the cloud is to perform encryption on the data and then upload it into the cloud server. The most popular of these, Searchable Encryption by which the client can store encrypted data in the cloud using encryption methods. Further, a client can also execute keyword searches over the cipher text domain. Many works have been proposed in this regard under various ~~threat models~~ different approaches to accomplish the needed search capability, such as multi-keyword Boolean search, ranked search, single keyword search, similarity search, and multi-keyword tagged search. The drawbacks of these approaches are massive cost in terms of data usability. Consider the fact that several of the existing techniques for keyword-based information retrieval and plaintext data cannot be easily applied to encrypted data. It's impractical to download large amounts of files and data from the cloud and decrypt them locally.Because of the large processing burden for both the cloud server and the user, existing System approaches are not practicable.There had been various ranked search schemes that implemented single keyword search. This changed when Cao et al.[7] proposed the first multi-keyword ranked search scheme. This was a unique breakthrough in the industry that utilized the similarity of the inner products to compute the relevant

similarity score. The main disadvantage of this method is that the search time is practically proportional to the number of files in the dataset since a score must be generated for each file even if it does not include the sought term.Sun et al. [8] tried to improve the efficiency of the search and proposed a novel approach based on a index tree based data structure for the full dataset that used the Vector Space model to evaluate similarity.This model was used in conjunction to the cosine measure.This increased the efficiency of the search but at the same time, it decreased the search precision to some extent. Fu et al. [9] worked on a synonym-based approach that can be used to achieve more accurate search results along with supporting synonym queries. This scheme had the issue of update cost. The cost for updating was very huge. If a data owner wants to add a new file to their dataset, the index tree has to be reconstructed along with the vector index which is also encrypted to guarantee that the functioning is not hampered. Another new scheme was proposed by Sun et al. [10] which supported dynamic operations as well. ~~Because the data owner would have to encrypt the interior nodes of the tree as leaf nodes, this approach has a significant computational cost.~~ However this approach had a significant computation cost.

B. BLOOM FILTERS

This also led to the inefficiency of updating. Another new scheme was proposed by Cheng et al [11] which introduces the concept of a bloom filter on top of the existing search index tree scheme previously defined. This made an improvement on the efficiency of the search along with a reduction in the cost of dynamic operations mainly due to the properties owned by a bloom filter. A layered bloom-filter approach is not a very new concept. It has been done previously also for various different kinds of use-cases. Bloofi [12], one of those schemes has been established as a successful scheme owing to the works of Crainiceanu et al[12]. First mentions of a layered approach can be found owing to the works of Ficara et al [13] which propose a unique multilayered compressed scheme on bloom filters. The scheme clearly shows a remarkable improvement over the traditional bloom filter approach both in terms of space and lookup time. Another unique take on the multilayered approach is mentioned in the scheme designed by Cen et al[14] which proposes using the layered approach to detect duplicate URLs to improve the speed of data collection by web crawlers. The proposed scheme showed that we can reduce the total amount of the false positives to a greater extent in a layered approach compared to the traditional approach at the expense of memory which is far less than approaches used by hashmaps and sets. The improved efficiency and precision in a multilayer bloom filter make it a compelling factor in our proposed scheme and is very valuable in practice. Chum et al. [18] in recent years, proposed a scheme which introduces two bloom filters instead of one to reduce the count of false positives. The proposed scheme gains considerable performance improvements by reducing the false positive elements over a traditional single bloom filtered approach.*Rottenstreich* et al.[19] also suggested a memory efficient approach to completely eradicate the false positives count. Peng et al.[20] in his scheme also proposed a persistent bloom filter approach to test the set-membership operations.Their approach works on the membership testing for temporal queries and by using the properties of a bloom filter the space complexity is substantially reduced.Graf et al[21] proposed a new scheme using a XOR filter which is also an implementation of a multilayer bloom filter approach which reduces the space as well as the rate of false positives while achieving higher efficiency and precision.

III. PROPOSED APPROACH

A. SYSTEM MODEL

As shown in Fig.1, there are three data entities to consider in the system model of our suggested schemes: the owner, the user, and the cloud server.A data owner is a group of people, businesses, and individuals whose major objective is to move a document collection DC to the cloud in order to save money and increase management flexibility. Next the data owner will upload the encrypted document set C along with the keywords to the required cloud server.Next when an authorised user with the required permissions searches the cloud server for the documents relevant to them using the query words,the server hosted on the cloud will then go through the layered bloom filter to find the relevant documents.This is the process's last phase, in which the cloud server delivers to the user the most relevant documents.

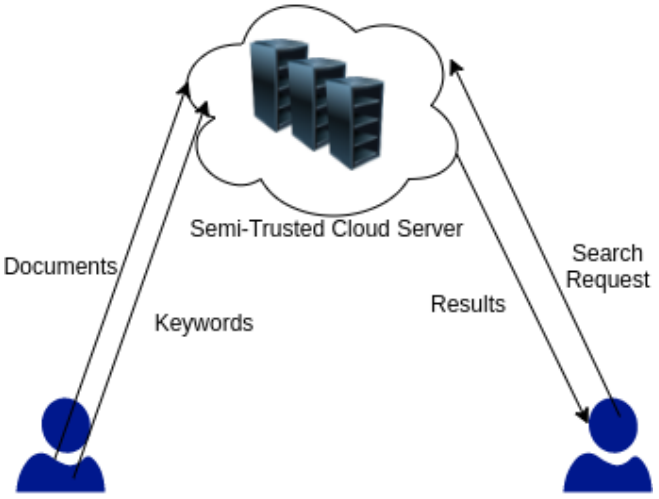

Fig 1.0 Design Model

B. DESIGN GOALS

~~As previously indicated, our novel scheme achieves more precise multi-keyword ranked search than the aforementioned systems.~~

~~Search efficiency: In our schemes, the search time should be sublinear to the size of the set of documents.~~

Precision: Our scheme increases the precision when compared to the traditional single-layer approach.

Analysis: Our scheme involves rigorous testing of the approaches for comparison of the false positives of a layered bloom filter versus a traditional single-layered approach.

IV. PRELIMINARIES

A. BLOOM FILTER & ANALYSIS

If we use hashmaps or sets though it is time-efficient, we have a significant concern over the space complexity as it will require O(n) space. In bloom filters, we make use of the bit array and more than one hash function. If we use more than one hash function to indicate that particular item x is in

the dataset, we reduce such false-positive cases. We use three hash functions to store the value of a key x, as shown in Fig 1.1.

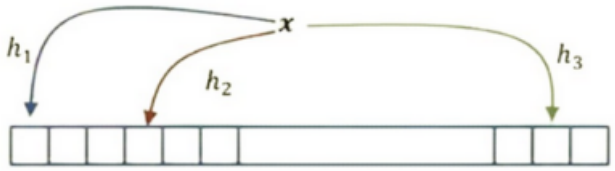

Fig 1.1: Description of Bloom filter data structure

To determine whether a value x has already been traversed in the bitarray, the bloom filter examines whether the indices of the locations of bit arrays generated by all hash functions is true or false. If this is the case, the value x may have already been traversed by the bitarray. Otherwise, it is not visited. Upon querying the bloom filters, we can say whether the value is not visited/present, but we can probably tell if the item is present/seen.

Suppose there are k hash functions, and the input is the given range of keywords. Then algorithm 1 and 2 represents the insertion and lookup operations in bloom filters.

Let "l" be the number of keywords ~~in the bloom filter~~. Let "k" be the number of hash functions. Let "m" be the number of bits in a bit array. Here "B" represents the intended bloom filter and "S" is the searchable keyword for the bloom filter. Each of the keywords is inserted sequentially into the bloom filter after passing it through "K" hash functions. While searching the same procedure is followed until and unless a false value is found which confirms that the element is not present in the bloom filter. For true values we need to check if the value pointed out by the bloom filter is a false positive value or not.

```
Initialize(B) i.e. for i∈{1..m},
B[i] = 0
for i =1 to l:
    for j= 1 to k
       Addr=Hj[ai]
       B[Addr]=1
    end for;
end for;
```

Algorithm 1: Insertion in bloom filters

```
LOOKUP(B,S):
  for i = 1 to l:
     for j = 1 to k
        Addr=Hj[ai]
        if B[Addr]!=1
              return false
        end if
     end for;
  end for;
```

Algorithm 2: Lookup in bloom filters

Disadvantages of bloom filter: The issue with bloom filter is in LOOKUP operation (Algorithm 2) sometimes still returns true even when an element is not present in the data-set.It is called false-positives values. The challenge is to reduce the false positives to get more accurate results from the Algorithm 2~~algorithm~~.

Analysis of Bloom filter: The increasing value of k (number of hash functions) possibly makes it harder for false positives to happen but it also increases the number of the filled up positions in the bloom filter array. Our goal is to find the optimal value of k.

False-positive analysis:

Let m = |B| and n elements are inserted. If S has not been inserted, what is the probability that Lookup(B, S) will return true? Assume hash functions $\{h_1, h_2, ..., h_k\}$ are independent and for element S $P_r[h_i(S) = j] = 1/m$ for all positions of j in a Bloom filter array B. Suppose we have inserted n elements and we have k hash functions, then

$$Pr[h_i(S) = 0] = (1 - 1/m)^{kn} \approx e^{-kn/m}$$

The expected number of zero bits = $me^{-kn/m}$ with high probability. Now the likelihood that LOOKUP will return present/true is that.

$$Pr[LOOKUP(B,S) = PRESENT] \approx (1 - e^{-kn/m})^k$$

We can easily observe from the above equation that when the value of m increases, the false positive value reduces. Now we need to choose k to minimize false positives, Let p = $e^{-kn/m}$, then.

$$Log(FalsePositive)$$
$$= log(1-p)^k = klog(1-p)$$
$$= -(m/n)log(p)log(1-p)$$

We can observe that we will get the optimal results at p = 1/2. Thus from the above equation, optimal results for the value of k can be evaluated as below.

$$k = mlog(2)/n$$

The significant advantage of bloom filters over a traditional time efficient hashmap or set-based approach is improved space efficiency and ease of query. Even though

hashmaps and set-based approaches can provide O(1) time complexity, the bloom filter is perfect for scenarios involving a large dataset with tight memory requirements since bloom filters require approximately 10 bits to store a character compared to a byte in traditional set based approaches.Insertion operations are always O(k) in time, and the query is also O(k) in time. The space complexity of bloom filters is O(m) because they require a bit-array of size m. A typical bloom filter, on the other hand, has its own set of drawbacks. Furthermore, because a bloom filter has no recollection of which bits were changed by which objects, we can only obtain a yes or no answer, and even a yes answer is not always right. As a result, we implemented a layered approach to the bloom filter, as discussed in the following sections.

B. MULTILAYER BLOOM FILTER

Bloom filters do not save the objects themselves and consume less space than the theoretical minimum of one byte per character required to store the data accurately, hence they have an error rate. They have false positives.[15] So another version of the bloom filter was introduced; it is known as a multi-layer bloom filter in which the number of false positives is decreased more than bloom filters.
In the instance of our multi-keyword search problem, we may partition the full set of keywords into layers and store them in the multi-layer bloom filter.

Each keyword of the strings separated by delimiter is present at different layers, and each layer has a corresponding classic bloom filter associated with them. We denote bloom filters as $BF^{k,m}$ = traditional bloom filter of k hash functions and m bits array each standard bloom filter is used to record the data at each layer. So the overall result will be obtained by using the output of all the layers up to which our keyword is present. Let L be the number of layers, the result of multilayer bloom filter will be like

$$MLBF^{L,k,m} = BF_1^{k,m}, BF_2^{k,m}, ..BF_L^{k,m}$$

While inserting the new keyword, suppose insert a_i in the ith layer of bloom filter. If any particular a_i is not in the ith layer, then we assume S is not present else present.

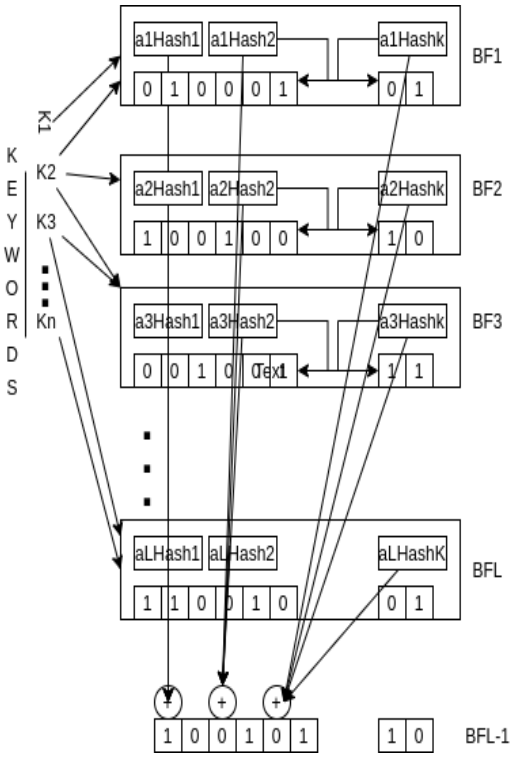

Fig 1.2 The principle of MLBF

When a new keyword is inserted, the way to calculate the bit-position in $BF_{L+1}^{k,m}$ is to perform an XOR operation among corresponding bit-positions of the 1-layer segment to the L-Layer segment, as shown in Fig 1.2 above. Algorithms 3 and 4 describe the steps to Insert and Lookup in a MultiLayer Bloom Filter.

Let "L" be the number of layers in the multilayer bloom filter. Let "k" be the number of hash functions.

```
for i=1 to L:
    for j=1 to k
        Addr=H_j[a_i]
        BF_i[Addr]=1
        LAddr[j]=LAddr[j]⊕Addr
    end for;
end for;
for j=1 to k
    BF_{L+1}[LAddr[j]]=1
end for;
```

Algorithm 3: Insert into MultiLayer Bloom Filter

```
//first, check BF_1 to BF_L
  for i=1 to L:
     for j=1 to k:
        Addr=H_j[a_i]
        if BF_i[Addr]!=1
              return false
        LAddr[j]=LAddr[j]⊕Addr
     end for;
  end for;
  //second, check BF_{L+1}
  for j=1 to k
     if BF_{L+1}[LAddr[j]]!=1
       return false;
     end if
  end for;
  return true;
```

Algorithm 4: Lookup in MultiLayer Bloom Filter

V. Proposed scheme

The user uploads a document along with the given range of keywords. The keywords can be automatically extracted from the document also using some background preprocessing function. The dataset now consists of a unique range of keywords extracted from a set of documents. The objective is to index the document using a multi layered bloom filter

~~Although the Bloom filter outperforms the traditional approaches, it does produce some false-positive results. So another version of bloom filter was introduced; it is known as a multi-layer bloom filter in which the number of false positives is decreased more than bloom filters in case of our problem based on multi-keyword ranked search.~~

The following steps are followed to perform the operations in our proposed scheme.

1. Get the entire keywords dataset from the user or by extraction from the document.
2. Each string of the input keywords is separated by a delimiter which will be split accordingly to a list of words.
3. The words from the list will be inserted into a multilayered bloom filter in a round-robin fashion.
4. Each word from the list is present at different layers, and each layer has a corresponding classic bloom filter associated with them.
5. We denote bloom filters as $BF^{k,m}$ = traditional bloom filter of k hash functions and m bits array.
6. Each standard bloom filter is used to record the data at each layer.
7. So the overall result will be obtained by using the output of all the layers up to which our keyword is present as depicted in algorithms 3 and 4.
8. Let L be the number of layers, the result of multilayer bloom filter will be like $MLBF^{L,k,m} = BF_1^{k,m}, BF_2^{k,m}, ..BF_L^{k,m}$
9. The multilayer bloom filter is indexed according to the document and ready to be searched upon.

VI. Experimental Setup and initial results

Setup: We built a small testbed consisting of an Intel I5-8300H Processor, 16GB Ram, and 512GB SSD. We are mainly testing the PRECISION & PRIVACY part of the experiment and trying to modify the results. The initial dataset used from Kaggle[16][17] comprised a set of keywords of some news articles and contained 1195191 and 3168800 unique values respectively.The size of the above mentioned datasets are 62.7MB and 247MB respectively. To determine the false positives in an ideal scenario, we run the following steps through a traditional bloom filter approach with a 0.05 error rate (the ideal lowest error rate for a bloom filter) and a multilayered bloom filter with n traditional bloom layers where n is determined uniquely by dividing the number of keywords with the hash count dynamically.

Algorithm 7: False positives detection

1. Get the unique dataset(A) and split it into equal numbers of rows. In our case, the parts will comprise 597595 appx rows. Let's call them B and C.
2. Make a new dataset consisting of 597595 rows which comprise shuffled rows from the original dataset. Let's call this T.
3. Insert the values from the dataset B&C into two different bloom filters BF_B & BF_C.
4. Insert the values from the dataset B&C into two different multi layer bloom filters & MBF_B & MBF_C .
5. For every E ∈ T
   if E ∈ BF_B and E ∈ BF_C
   false_postiive_bloom++
   if E ∈ MBF_B and E ∈ MBF_C
   false_postive_multi_bloom++
6. Return false_positive_bloom, false_positive_multi_bloom

From Algorithm 5 we can clearly see that since all the elements are unique, if a value is present in B and C simultaneously, it is evident that it's a false positive.

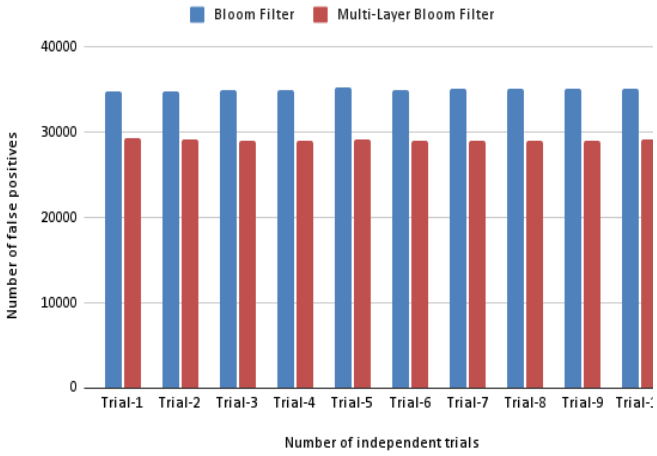

Fig 1.3 False Positive Rate in First Dataset

We ran our script 10 separate times to check for the accuracy of the underlying experiment. The results obtained from 10 separate experimental trials as defined in x-axes of Figs 1.3 & Figs 1.4 show that the amount of false positives in a bloom filter is in the 35000 range. In contrast, the number of false positives in the multilayered approach lies in the field of 30000, which indicates that there can be a 15-20% improvement in the newer multilayered approach. For the second dataset consisting of 316880 unique rows the results look similar as depicted in figure 1.4 which shows a comparison in the total number of false positives in the two different approaches. Initial results indicate that there can be a theoretical improvement in the range of 15-30% by including the MultiLayer approach over the traditional bloom filter.

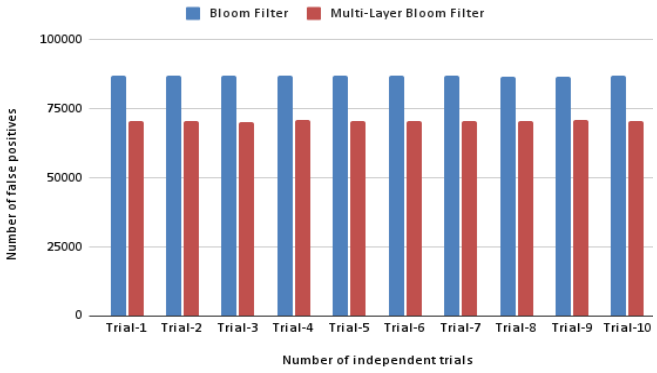

Fig 1.4 False positive rate in second dataset

The number of hash functions defined play a very important role in determining the precision of the system in whole. In our experiments we set the number of hash functions, K = 2,3,4,5,6,7 and calculated the false positive probability and the precision performance for the respective datasets as defined in Table 1.Table 2 denotes the precision percentage of a multilayer bloom filter "mbf" compared to a traditional bloom filter "bf". We can clearly see in figure 1.5 and figure 1.6 that there is a considerable improvement in the performance using the multilayered approach over a traditional bloom filter. On further exploration we can notice that for the dataset we are using with the range of keywords ranging from 5-20 word maximum performance is achieved when the number of hash-functions is between 4-5. Our scheme is optimized in the way that it can dynamically update the number of layers in the bloom filter to generate the optimal amount of layers to match with the optimal value of the hash function. That way we can guarantee that optimal precision is always achieved for any number of keywords.

TABLE 1. The probability of false positive in bloom filter

k	p
2	0.07529613
3	0.05570797
4	0.05026948
5	0.05102869
6	0.05569736
7	0.06357069

TABLE 2. The precision percentages for k hash functions

k	mbf	bf
2	99.966510	99.957916
3	99.973804	99.967851
4	99.975770	99.970712
5	99.975895	99.970680
6	99.974507	99.968552
7	99.972379	99.964975

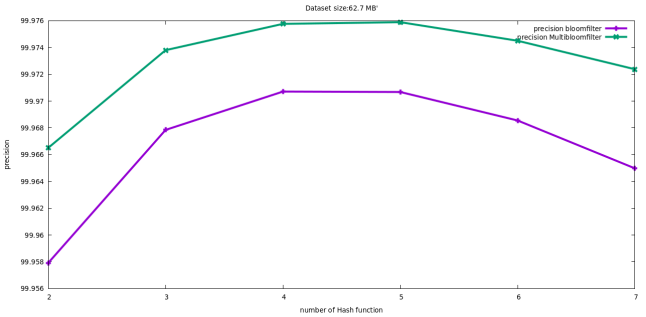

Fig 1.5 Precision performance for first dataset

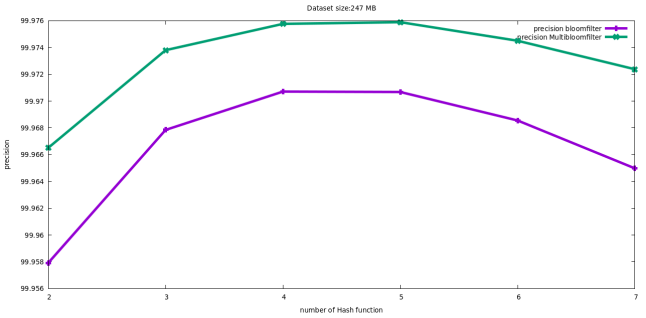

Fig 1.6 Precision performance for second dataset

VII. CONCLUSION

Throughout this work, we have presented a layered bloom filter approach to store keywords over cloud data. Furthermore, our proposed plan has the potential to improve the precision of the existing scheme by a factor of 15-30%.

We are implementing a dynamic layered approach of a Bloom filter to determine the relevant documents ~~to improve search efficiency~~. Because of the Bloom filter's properties, this new scheme can be implemented in very less memory space compared to traditional approaches ~~and is also dynamic in its approach to maximise the precision of the queried documents.~~ Moreover the layered approach helps to reduce the false positives count to a greater extent than any traditional approach. The memory usage in our approach is far less compared to any traditional approaches often ranging from 450 KB for a range of 1195191 unique values and 1.27 MB only for a range of 316880 unique values. Finally, we would like to state that the experimental results obtained using a real-world dataset demonstrate that our scheme is capable of achieving the design objectives efficiently and effectively.

VIII. Acknowledgment

I would like to express my gratitude to my supervisor, Professor Dr. DVN Siva Kumar, for making this work possible. His friendly guidance and expert advice were invaluable at all stages of the project.

References

[1] Cheng Guo, Ruhan Zhuang, Chin-Chen Chang, Qiongqiong Yuan. "Dynamic Multi-Keyword Ranked Search Based on Bloom Filter Over Encrypted Cloud Data," IEEE Access, 2019

[2] Cen Zhiwang, Xu Jungang, Sun Jian. "A multilayer bloom filter for duplicated URL detection," 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010

[3] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," IEEE Trans. the Parallel Distrib. Syst., vol. 27, no. 2, pp. 340–352, Jan. 2016.

[4] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS), 2006, vol. 19, no. 5, pp. 79–88.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in Advances in Cryptology-EUROCRYPT. Berlin, Germany: Springer, 2004, pp. 506–522.

[6] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," IEICE Trans. Commun., vol. E98-B, no. 1, pp. 190–200, 2015

[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in Proc. IEEE INFO-COM, Apr. 2011, pp. 829–837.

[8] W. Sun et al., "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur. New York, NY, USA: ACM, 2013, pp. 71–82.

[9] Z. Fu, X. Sun, N. Linge, and L. Zhou, "Achieving effective cloud search services: Multi-keyword ranked search over encrypted cloud data supporting synonym query," IEEE Trans. Consum. Electron., vol. 60, no. 1, pp. 164–172, Feb. 2014.

[10] X. Sun, X. Wang, Z. Xia, Z. Fu, and T. Li, "Dynamic multi-keyword top-k ranked search over encrypted cloud data," J. Int. J. Secur. Appl., vol. 8, no. 1, pp. 319–332, 2014.

[11] C. Guo, R. Zhuang, C. Chang and Q. Yuan, "Dynamic Multi-Keyword Ranked Search Based on Bloom Filter Over Encrypted Cloud Data," in IEEE Access, vol. 7, pp. 35826-35837, 2019, doi: 10.1109/ACCESS.2019.2904763.

[12] Adina Crainiceanu and Daniel Lemire. Bloofi: Multidimensional Bloom Filters. Information Systems,Volume 54, December 2015, pp.311-324 http://arxiv.org/abs/1501.01941 http://www.sciencedirect.com/science/article/pii/S0306437915000125

[13] D. Ficara, S. Giordano, G. Procissi and F. Vitucci, "MultiLayer Compressed Counting Bloom Filters," IEEE INFOCOM 2008 - The 27th Conference on Computer Communications, 2008, pp. 311-315, doi: 10.1109/INFOCOM.2008.71.

[14] Cen Zhiwang, Xu Jungang and Sun Jian, "A multi-layer bloom filter for duplicated URL detection," 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010, pp. V1-586-V1-591, doi: 10.1109/ICACTE.2010.5578947.

[15] J. Kim, "On the False Positive Rate of the Bloom Filter in Case of Using Multiple Hash Functions," 2014 Ninth Asia Joint Conference on Information Security, 2014, pp. 26-30, DOI: 10.1109/AsiaJCIS.2014.32.

[16] https://www.kaggle.com/therohk/million-headlines

[17] https://www.kaggle.com/therohk/india-headlines-news-dataset

[18] *Chi Sing Chum and Xiaowen Zhang. 2017. A New Bloom Filter Structure for Searchable Encryption Schemes. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17). Association for Computing Machinery, New York, NY, USA, 143–145. DOI:https://doi.org/10.1145/3029806.3029839*

[19] Ori Rottenstreich, Pedro Reviriego, Ely Porat, and S. Muthukrishnan. 2020. Constructions and Applications for Accurate Counting of the Bloom Filter False Positive Free Zone. In Proceedings of the Symposium on SDN Research (SOSR '20). Association for Computing Machinery, New York, NY, USA, 135–145. DOI:https://doi.org/10.1145/3373360.3380845

[20] Yanqing Peng, Jinwei Guo, Feifei Li, Weining Qian, and Aoying Zhou. 2018. Persistent Bloom Filter: Membership Testing for the Entire History. In *Proceedings of the 2018 International Conference on Management of Data* (*SIGMOD '18*). Association for Computing Machinery, New York, NY, USA, 1037–1052. DOI:https://doi.org/10.1145/3183713.3183737

[21] *Thomas Mueller Graf and Daniel Lemire. 2020. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. ACM J. Exp. Algorithmics 25, Article 1.5 (2020), 16 pages. DOI:https://doi.org/10.1145/3376122*

[22]

[23]