scaleBF: A High Scalable Membership Filter using 3D Bloom Filter

Ripon Patgiri[1], Sabuzima Nayak[2], Samir Kumar Borgohain[3]
Dept of Computer Science & Engineering
National Institute of Technology Silchar
Assam-788010, India

Abstract—**Bloom Filter is extensively deployed data structure in various applications and research domain since its inception. Bloom Filter is able to reduce the space consumption in an order of magnitude. Thus, Bloom Filter is used to keep information of a very large scale data. There are numerous variants of Bloom Filters available, however, scalability is a serious dilemma of Bloom Filter for years. To solve this dilemma, there are also diverse variants of Bloom Filter. However, the time complexity and space complexity become the key issue again. In this paper, we present a novel Bloom Filter to address the scalability issue without compromising the performance, called scaleBF. scaleBF deploys many 3D Bloom Filter to filter the set of items. In this paper, we theoretically compare the contemporary Bloom Filter for scalability and scaleBF outperforms in terms of time complexity.**

Keywords—*Bloom filter; membership filter; scalable bloom filter, duplicate key filter; hashing; data structure, membership query.*

I. Introduction

Burton Howard Bloom introduces a data structure on approximate membership query in 1970 [1], hence, it is named as Bloom Filter. Bloom Filter is an extensively experimented to enhance a system's performance since its inception. Moreover, Bloom Filter is also applied numerous areas, namely, Big Data, Cloud Computing, Networking, Security [2], Database, IoT, Bioinformatics, Biometrics, and Distributed system. However, Bloom Filter is inapplicable in hard real-time system, and password management system [3] due to accuracy issues. Applications of Bloom Filter take the lion's share in Computer Networking which includes Named Data Networking (NDN), Content-Centric Networking (CCN), Software-defined Network (SDN), Domain Name System (DNS), and Computer Security. The Bloom Filter reduces space consumption in an order of magnitude as compared to a conventional hash algorithm. However, Bloom Filter cannot stand itself. Bloom Filter is used as enhancer of a system. For example, BigTable uses Bloom Filter to reduce the number of disk accesses which improves the performance drastically [4]. Similarly, in Cassandra [5].

A. Motivation

Several variants of Bloom Filters have been developed to address some issues [6]. However, most of the Bloom Filters are developed to address scalability issue. Guanlin Lu et al. [7] proposes a Forest-structured Bloom Filter (FBF). The FBF is a combination of RAM and flash memory. Similarly, Debnath et al. [8] develops a very high scalable Bloom Filter

based on RAM and flash memory. BloomStore is also another highly scalable Bloom Filter [9]. However, these solutions are hierarchical, and thus, lookup and insertion cost is very high. It takes $O(logn)$ time complexity in insertion and lookup operations as demonstrated in Table I.

B. Contribution

To address scalability issues, we propose a novel scalable Bloom Filter, called scaleBF. scaleBF is a very simple data structure yet powerful. scaleBF increases its scalability without compromising the performance. scaleBF takes $O(1)$ time complexity in lookup and insertion operations, which is compared in Table I. scaleBF uses chaining hash data structure for implementing the scalability. Also, scaleBF deploys 3DBF [12] to inherit the performance and low memory consumption.

Table I depicts the most scalable Bloom Filters. Bloom-Flash [8], and FBF [7] uses hierarchical structures to indexed the Bloom Filters. BloomStore [9] uses linear chain data structure (not open hashing data structure) to store the Bloom Filters in Flash memory. Moreover, BloomStore is designed to perform parallel lookup operation. On the contrary, scaleBF uses chaining hash data structures to achieve higher scalability without compromising the performances. TB^{2}F [10] deploys tree-bitmaps and Bloom Filter, and used for name lookup in Content-Centric Network (CCN). The input is split into a T-segment of fixed size and a B-Segment of variable size. The T-segment key is inserted into bitmap-trie, and the B-segment is inserted into Bloom Filter. However, maintaining trie data structure is costly in terms of space as well as time. On the other hand, Bloofi [11] uses tree structured Bloom Filter which cuases costly in insertion and lookup. The scalability of BloomFlash [8], FBF [7], BloomStore [9], scaleBF is higher than TB^{2}F and Bloofi [11].

C. Organization

The article is organized as follows- Section II presents the proposed system, called scaleBF. The architecture of scaleBF is demonstrated in Section II. Section III presents a theoretical analysis on scaleBF. Also, every aspect of scaleBF is analyzed in Section III. Article discusses cons of scaleBF in Section IV. Finally, the article is concluded in Section V.

II. scaleBF: The Proposed System

A. 3D Bloom Filter (3DBF)

The 3-Dimensional Bloom Filter (3DBF) is similar to conventional Bloom Filter except array structure [12]. The

TABLE I.   COMPARISON OF VARIOUS SCALABLE BLOOM FILTER

Name	Types	Insertion	Lookup	Scalability	Platform	Algorithm
BloomFlash [8]	Hierarchical	Logarithmic	Logarithmic	High	RAM & Flash	Serial
FBF [7]	Hierarchical	Logarithmic	Logarithmic	High	RAM & Flash	Serial
BloomStore [9]	Linear Chaining	Constant	Constant	High	RAM & Flash	Parallel
TB^{2}F [10]	Hierarchical	Logarithmic	Logarithmic	Medium	RAM	Parallel
Bloofi [11]	Hierarchical	Logarithmic	Logarithmic	Medium	RAM	Serial
scaleBF	3D	Constant	Constant	High	RAM	Serial

3DBF uses 3D arrays and it is a static Bloom Filter in nature. The static Bloom Filter does not change the size at run time. Also, static Bloom Filter does not readjust with ever growing data. However, a new 3DBF is created to address the scalability issue.
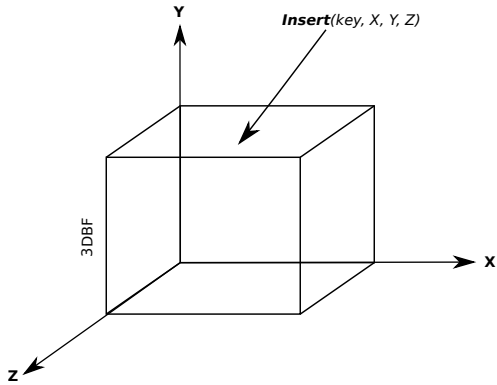

Fig. 1.   3DBF architecture

Figure 1 depicts the architecture of 3DBF. The 3D Bloom Filter uses four modulus operator using prime numbers instead of hashing a key into k different places. These modulus reduces the false positive probability. Thus, 3DBF is independent from number of hash functions k. Let, $\mathbb{B}_{X,Y,Z}$ bet the 3DBF where X, Y and Z be the dimension of the filter. The dimensions are prime numbers, otherwise, false positive increases. Let, $\mathbb{B}_{i,j,k}$ be a cell of the 3DBF. The cell stores **long int** which occupies $64 - bits$. Let us insert a key κ. The 3DBF uses Murmur hashing [13] to generate a hash-value of input item κ. Let, h be the generated hash-value by Murmur hashing. Now, $i = h\%X$, $j = h\%Y$, $k = h\%Z$, and $\rho = h\%63$, where ρ is the bit position of the cell $\mathbb{B}_{i,j,k}$. 3DBF sets a bit using Equation (1)-

$$\mathbb{B}_{i,j,k} \leftarrow \mathbb{B}_{i,j,k}\ OR\ (1 << \rho) \tag{1}$$

where OR is bitwise OR operator. Equation (1) is invoked to insert an input item into 3DBF. The lookup operation requires similar calculation. Equation (2) is invoked to perform the lookup operation in a 3DBF.

$$Flag \leftarrow (\mathbb{B}_{i,j,k} \oplus (1 << \rho))AND(1 << \rho) \tag{2}$$

If $Flag$ is assigned by '1', then 3DBF returns true, otherwise, it returns false. Each item requires a single bit in 3DBF as disclosed in Equation (1), and each cell has $63 - bits$. Therefore, 3DBF consumes the lowest memory as compared to other variants of Bloom Filter. Moreover, 3DBF features detection of the fullness of the filter. 3DBF defines the criticality factor to consider whether the filter is full or not [12].

B. Insertion operation in scaleBF

scaleBF deploys chaining mechanism of conventional hashing data structure for highly scalable. scaleBF deploys many 3DBFs.

1) Insertion of Bloom Filter: A Bloom Filter is formed by three 3DBFs. Each Bloom Filter is formed by three 3DBF. However, Bloom Filter can be formed by augmenting more 3DBF, but we have chosen three for simpler illustration. Each key is inserted into three 3DBF. Let, η be the chain size, and input item κ to be inserted. There are η chains in scaleBF. A new Bloom Filter (three 3DBF) is inserted into the chain if the Bloom Filter (three 3DBF) in particular chain is full.

2) Insertion of a Key: Insertion of the key is performed using Equation (1) and hashes the key into the particular chain. If a Bloom Filter (three 3DBF) size is full, then move to the last Bloom Filter (three 3DBF). Insert the key using Equation (1). A key is hashed into particular slot of the chain. There are many Bloom Filters in the particular slot linked with each other as shown in Figure 2. If first three Bloom Filter is full, then create and link three 3DBF as demonstrated in the figure.

C. Lookup operation in scaleBF

Figure 3 depicts the lookup operations of the scaleBF. A key is hashed into particular chain and lookup all Bloom Filters sequentially. As a comparison, three 3DBF is searched. If the first three Bloom Filter returns true, then the key is member of Bloom Filter. Otherwise, move forward to the next three 3DBF, and so on.

III.   ANALYSIS

There is no significant difference between 3DBF and conventional Bloom Filter analysis of number bits consumed, except $k = 1$ in 3DBF. Therefore, scaleBF is analyzed through the conventional Bloom Filter. Let, m be the size of Bloom Filter, n be the number of entries, and $k = 1$ be the number of hash function, then the probability of a bit to be '0' is

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

Therefore, probability of total bit to be '1' is

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

Since, scaleBF uses 3DBF, thus, $m = X \times Y \times Z \times 63$. F. Grandi [14] present a new way to calculate the false positive probability using $\delta - transformation$. Let, X be the random
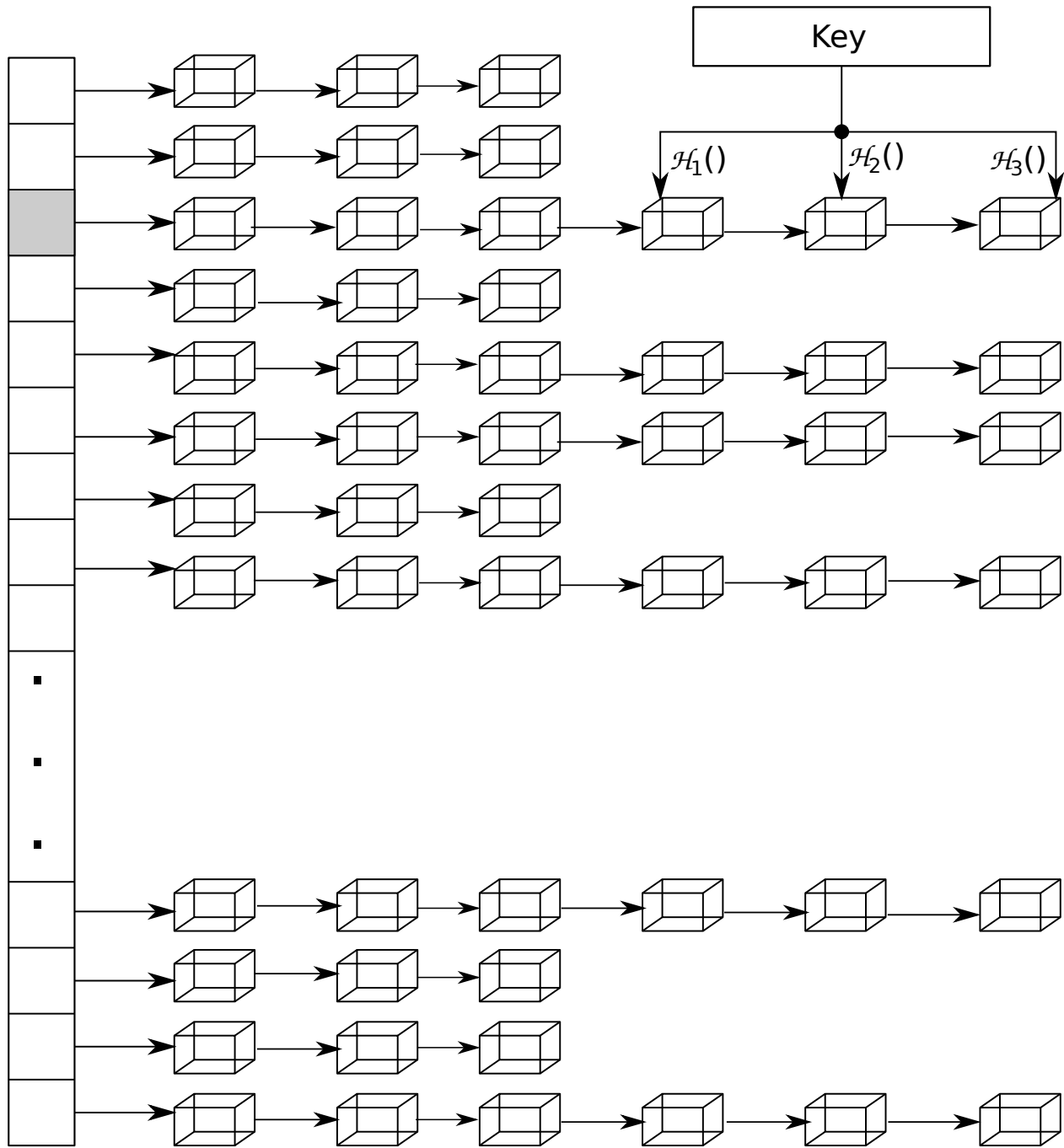
Fig. 2.   Insertion of an input item and increment of filter size using conventional chaining in scaleBF.

variable to represent the total number of '1' in the Bloom Filter, then

$$E[X] = m \left(1 - \left(1 - \frac{1}{m} \right)^n \right)$$

The probability of false positive is conditioned to a number by $X = x$, then

$$Pr(FP|X = x) = \left(\frac{x}{m} \right)$$

Therefore, false positive probability is

$$FPP = \sum_{x=0}^{m} Pr(FP|X = x)Pr(X = x)$$

$$FPP = \sum_{x=0}^{m} \left(\frac{x}{m} \right) f(x)$$

where $f(x)$ is probability mass function of X. F. Grandi [14] applies $\delta - transformation$ to calculate $f(x)$ and
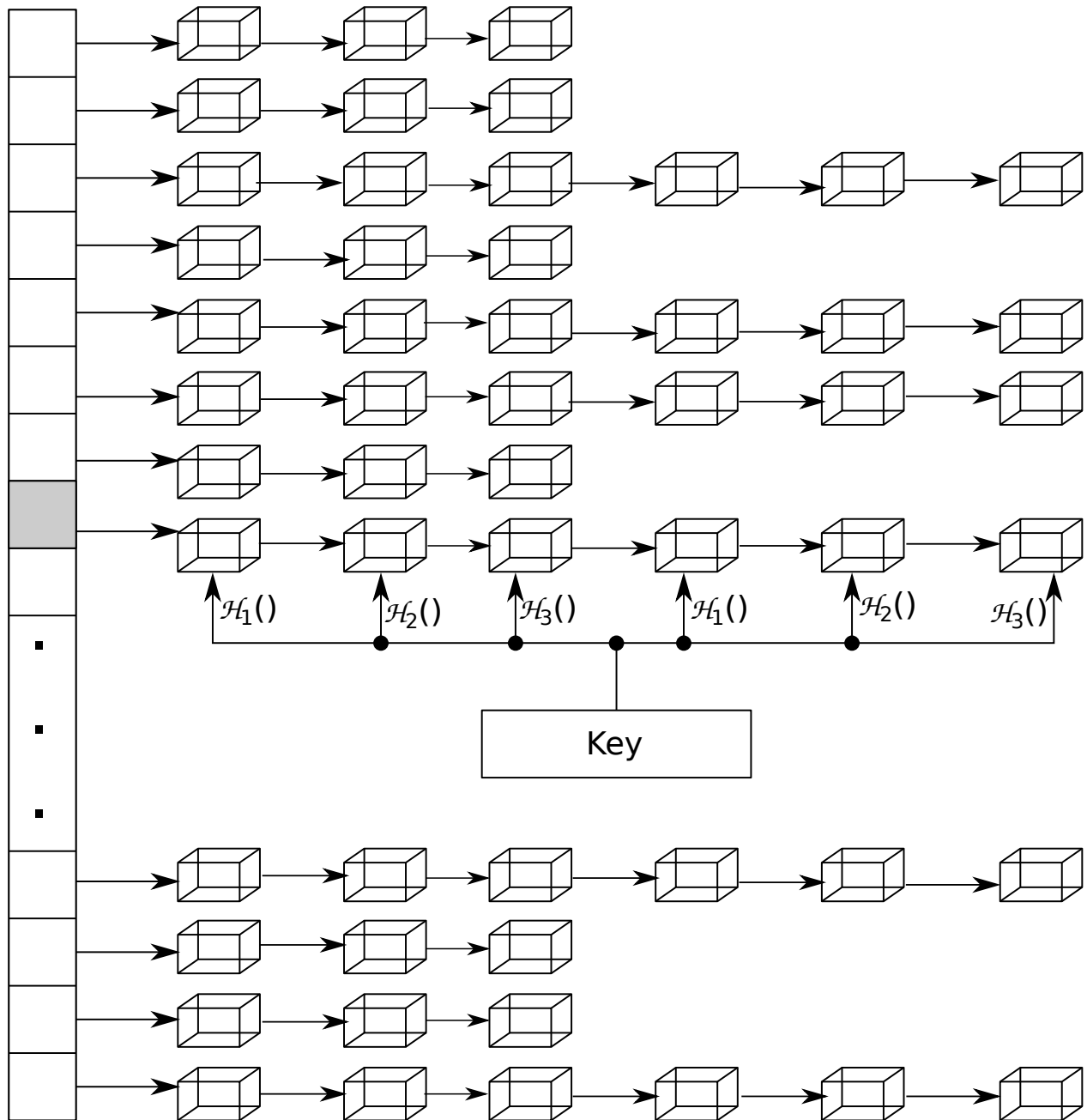
Fig. 3. Lookup of an input item in scaleBF.

presents FPP as follows-

$$FPP = \sum_{x=0}^{m} \left(\frac{x}{m}\right) \binom{m}{x} \sum_{j=0}^{x} (-1)^j \binom{x}{j} \left(\frac{x-j}{m}\right)^n \quad (3)$$

Equation 3 is calculation of false positive probability of a 3DBF. scaleBF deploys three 3DBF. Therefore, the false positive probability of a Bloom Filter having three 3DBF is

$$FPP = \prod_{p=1}^{3} \left(\sum_{x=0}^{m} \left(\frac{x}{m}\right) \binom{m}{x} \sum_{j=0}^{x} (-1)^j \binom{x}{j} \left(\frac{x-j}{m}\right)^n \right) \quad (4)$$

Let, there are n Bloom Filter (three 3DBF each), and their false positive probabilities are FPP_i, where $i = 1, 2, 3, \ldots, n$. From Equation (4), the average false positive probability of scaleBF is

$$FPP_{avg} = \frac{\sum_{i=1}^{m} FPP_i}{n} \quad (5)$$

Equation (5) presents the false positive probability of scaleBF.

A. Scalability

Scalability is the key barrier to the modern Bloom Filter. There are numerous Bloom Filter that addresses the scalability

issue. However, scalable Bloom Filters are developed based on reordering, hierarchical and forest structure. scaleBF uses simple hashing scheme to enhance the scalability of Bloom Filter. The chaining is the most used hashing data structure. However, chaining has linear search in the worst case, i.e., $O(n)$ time complexity. In other words, all keys are hashed into single chain location. However, it is once in a blue moon in real-world. Besides, most of the chain remains unused. Therefore, the chaining size must be a prime number to avoid the above situation.

Undoubtedly, the scalability is achieved using chaining data structure in scaleBF. The RAM size of the system also plays a vital role in scaleBF. 3DBF allocates memory dynamically which requires few memory blocks be contiguous to satisfy the request by the most modern programming language. Therefore, there is less worry about the unavailability of memory blocks. However, scaleBF does not guarantee the availability of the memory.

Let, P be the slot size and Q be the number of chains to be stored in chaining. The load factor $\alpha = \frac{Q}{P}$, where P is a prime number, and Q is the total Bloom Filter to be inserted. Therefore,

$$Q = \sum_{i=1}^{T} \frac{m_i}{3} \qquad (6)$$

where m_i is the size of ith 3DBF. Then, the load factor becomes

$$\alpha = \frac{\sum_{i=1}^{T} \frac{m_i}{3}}{P} \qquad (7)$$

The total available bits in scaleBF are

$$\frac{\tau \times X \times Y \times Z \times \left(\sum_{i=1}^{T} \frac{m_i}{3} \right)}{P} \qquad (8)$$

where τ is the threshold that depends on the requirements, X, Y, and Z are the dimensions. The τ is calculated by $1, 2, 3, \ldots, \beta$ and β be the number of bits per cell in a 3DBF [12]. For high accuracy, τ is set to 1. However, $\tau = \beta$ defines that false positive is insignificance.

B. Time and Space Complexity

The time complexity is also a key barrier in the scalable Bloom Filter. Hierarchical Bloom Filter or Forest Structured Bloom Filter takes $O(logn)$ time complexity in lookup and insertion operation. Other variants of scalable Bloom Filters also decrease the performance. scaleBF uses $O(1)$ time complexity to lookup and insertion operation on an average case. However, the worst case time complexity is $O(n)$ and it is impractical.

Let, a key κ to be inserted into scaleBF. The κ is hashed into a particular slot of chain and insert into the key κ in desired Bloom Filter (three 3DBF). If the first Bloom Filter is full, then move to the next and so on. Let, the maximum, the size of a particular chain is $\mathcal{C}$. scaleBF uses prime number P to evenly distribute the keys as disclosed in Equation (7). Thus, the size of $\mathcal{C}$ is small. Let us, there are 70% slots empty even if prime number P. That is, 30% slots are filled. Then, each slot has at least 30% of Q which is also very small.

However, the P is a prime number, and thus, the distribution is fair enough to fill each slot. Thus, $\mathcal{C}$ is very small and the total time complexity is $O(1)$ on an average. Similarly, lookup cost also $O(1)$ on an average case.

C. Performance

scaleBF also inherits the performance of 3DBF [12]. The insertion and lookup cost depends on the cost of Equation (1) and (2). Equation (1) and (2) uses Murmur hashing [13], which is known as a very fast string hashing. The computational complexity of Murmur hashing is $O(1)$, since, the length of a string is constant and small. Therefore, the Equation (1) and (2) also cost $O(1)$ time complexity. 3DBF enhances the performance by reducing the total number of complex arithmetic operations. Thus, scaleBF increase its scalability without compromising the performance.

IV. DISCUSSION

scaleBF provides impressively very high scalability. However, the initial cost of memory consumption can be high. For instance, insert a key which mapped to the slot 3 of chaining, and creates new three 3DBF. Another insertion key also triggers creation of new three 3DBF which is mapped into a slot, say 2. Thus, the initial cost of memory is high. However, scaleBF is ideal for very large scale membership filtering. Moreover, scaleBF also ideal solution of large memory allocation due to dynamic memory allocation system. scaleBF also depends on the size of 3DBF.

V. CONCLUSION

Deduplication requires very high scalable Bloom Filter, since, deduplication processes trillions of keys. Moreover, there are diverse applications of high scalable Bloom Filter, for instance, DNA Assembly. In this paper, we have presented a very high scalable Bloom Filter without comprising the performances. In addition, scaleBF also provides insertion and lookup cost of $O(1)$. scaleBF outperforms Bloofi [11], Bloom-Flash [8], FBF [7], and TB2F [10] in terms of computational time complexity while maintaining higher scalability. However, the scaleBF does not support deletion of an item. Thus, there is no false negative. Interestingly, scaleBF can be applied many research areas to boost up the performance and scalability, and its applicability not limited to NDN, but also Big Data, Cloud Computing, Database, Distriubuted System, IoT, and Computer Networking.

REFERENCES

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[2] R. Patgiri, S. Nayak, and S. K. Borgohain, "Preventing ddos using bloom filter: A survey," *EAI Endorsed Transactions on Scalable Information Systems*, 2018.

[3] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, 2008.

[5]   A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.

[6]   R. Patgiri, S. Nayak, and S. K. Borgohain, "Shed more light on bloom filter's variants," in *Proceedings of the 2018 International Conference on Information and Knowledge Engineering*.   CSREA Press, 2018, pp. 14–21.

[7]   G. Lu, B. Debnath, and D. H. C. Du, "A forest-structured bloom filter with flash memory," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, 2011, pp. 1–6.

[8]   B. Debnath, S. Sengupta, J. Li, D. J. Lilja, and D. H. C. Du, "Bloom-flash: Bloom filter on flash-based storage," in *2011 31st International Conference on Distributed Computing Systems*, 2011, pp. 635–644.

[9]   G. Lu, Y. J. Nam, and D. H. C. Du, "Bloomstore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash," in *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 2012, pp. 1–11.

[10] W. Quan, C. Xu, A. V. Vasilakos, J. Guan, H. Zhang, and L. A. Grieco, "Tb2f: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking," in *2014 IFIP Networking Conference(IFIP NETWORKING)*, vol. 00, 2014, pp. 1–9.

[11] A. Crainiceanu and D. Lemire, "Bloofi: Multidimensional bloom filters," *Information Systems*, vol. 54, pp. 311 – 324, 2015.

[12] R. Patgiri, S. Nayak, and S. K. Borgohain, "rDBF: A r-dimensional bloom filter for massive scale membership query," *Journal of Network and Computer Applications*, vol. Personal communication.

[13] A. Appleby, "Murmurhash," Retrieved on August 2018 from https://sites.google.com/site/murmurhash/, 2018.

[14] F. Grandi, "On the analysis of bloom filters," *Information Processing Letters*, vol. 129, pp. 35 – 39, 2018.