Modifications using Circular Shift for a Better Bloom Filter

MyeongKyu Kim
Konkuk University
S. Korea
hehe6764@gmail.com

SungRyul Kim
Konkuk University
S. Korea
kimsr@konkuk.ac.kr

ABSTRACT

The Bloom filter[1] is a hash-based data structure that facilitates membership querying. Computation speed of Bloom filter is affected by hash functions that produce hash outputs. Basically, two operations: 'add' and 'query', consists of the Bloom filter. Previous researches have shown advanced computation speed of Bloom filter since the standard Bloom Filter is published. For example, Double Hash Bloom filter, Single Hash Bloom filter, etc.

We propose a system that uses less computation than previous works with similar false positive and which is implemented by using a circular bit shift. This method was implemented with faster calculation speed, compared with previous works. Furthermore, experiments which were compared with previous researches and standard Bloom filter. Therefore, we demonstrate that the proposed system computes faster than previous studies with similar false positive rate.

CCS CONCEPTS

• **Theory of computation → Bloom filters and hashing**;

• **Information systems → Query optimization**;

KEYWORDS

Bloom Filter, Hash Function, Circular shift

ACM Reference format:

G. Gubbiotti, P. Malagò, S. Fin, S. Tacchi, L. Giovannini, D. Bisero, M. Madami, and G. Carlotti. 1997. SIG Proceedings Paper in word Format. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK'97)*, 4 pages. https://doi.org/10.1145/123 4

1 INTRODUCTION

The Bloom filter is a hash-based data structure that facilitates membership querying [1]. k hash functions, independently random, encrypt each member in the input data. Then, each hashed output value denotes the address number in the Bloom filter. For the identification of an element which is included in a Bloom filter but not in included input data, k hash functions also encrypt the element and each output value are compared with the Bloom filter address. The above processes represent the benefits to discover a member, nevertheless, the Bloom filter has the error: false positive. [4] and [5] demonstrate more accurate false positive rate by the asymptotic bounds and constrained nonlinearization. Regardless of this deficiency, the Bloom filter has been used for various purposes: Squid web proxy cache, resource routing, packet routing [6] and even to identify malicious URLs in Google Chrome [7].

Previous researches have increased computation speed from the standard Bloom filter, although they calculate the similar false positive rate. The Double Hash [2] and, the Single Hash [3] have shown the method that decreases computation time from the standard Bloom Filter. Each of them has shown the improvement in calculation time from the standard Bloom Filter [1]. The Double Hashing Bloom filter uses two hash functions and simulates k (k > 2) hash functions. Single Hash Bloom filter uses just one hash function to simulates k hash functions.

We verify that the proposed system computes faster than previous researches with similar false positive rate. The proposed system is demonstrated through the experiments. Furthermore, a circular shift, an operation rearranging the final entry to the first position, is applied to implement the proposed system. We use one hash function to encrypt the data and calculate k simulation data from the encrypted data. We use the circular shift method to k simulation data. This method is used for faster computation. The following equation defines a circular bit shift:

$$\sigma(i) \equiv (i+1) mod\ n, \qquad \forall i = 1, \dots, n$$

Circular bit shift has been applied to various fields in computer science: a circular data structure etc. In this paper, we simulate hash functions by using a circular bit shift method. In the experiments, assembly instructions support the high-level programming language to decrease computation time.

Section 2 is separated into three parts as followed:
1) Description of Bloom filter and false positive.; 2) Previous improved Bloom filter.; 3) Regarding predictions of false positive in previous works. At first, we describe the Bloom filter theoretically by equations and through practical examples with figures. Furthermore, we introduce the previous researches as followed: Double Hash Bloom Filter and Single Hash Bloom Filter. Finally, we predict the improved Bloom filter from the previous researches. Section 3, we propose the system which has

RACS'20, October 13-16, 2020, Gwangju, Korea

ACM ISBN 978-1-4503-8025-6/20/10. . . $15.00
https://doi.org/10.1145/123_4

an advanced computation method compared to previous works. Section 4 presents experiments to compare with previous works and methods that this paper proposes. Finally, section 5 presents a conclusion and suggestions for future work.

2 PREVIOUS WORKS

2.1 Standard Bloom Filter

We introduce the following notation by convention:

n	The s ize of input element
m	The s ize of Bloom filter
k	The number of hash functions
$h_i(x)$	A hash function
bf	Abbreviation of the standard Bloom filter
dh	Abbreviation of the Double Hash Bloom filter
sh	Abbreviation of the Single Hash Bloom filter
21cX	X hash functions are selected in 21 hash functions

2.1.1. Description. The Bloom filter [1] consists of sequential bits array that is consisted of 0's and 1's. Let m be the size of a Bloom filter and n denote size of the input. Let set S = $\{x_1, \dots, x_n\}$ denote the input set. Initially, m bits are set to 0 in the Bloom filter. Let k hash functions, set H = $h_1(\ ), \dots, h_k(\ )$, encrypt n messages to produce n·k outputs. Each hash output indicates an address number in an m bits array Bloom filter and then the address is set to 1 in the Bloom filter. When an encrypted output indicates an address, which is previously set to 1 repeatedly, the address remains with 1. After all outputs are stored into the Bloom filter, a query can be checked whether y is in set S.

Fig. 1 describes the characteristics of a Bloom filter.

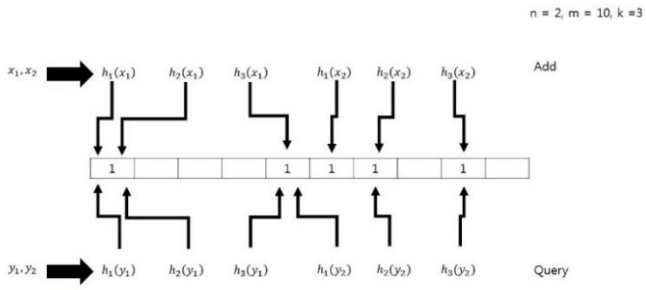

Figure 1: Description of standard Bloom filter

Fig. 1 represents three cases in the Bloom filter. First, when $h_1(x_1)$ and $h_2(x_1)$ represent the same address, it is set to 1. Second, Fig. 1 represents an element 'y_1' in the set S. Third as it is described in the next paragraph, the Bloom filter evaluates an element y_2 which is checked in the set S but element y_2 is not in the set S: false positive. As mentioned equation (2-7), we can calculate the approximate false positive in the Fig. 1:

$$\approx \left(1 - e^{-3*2/10}\right)^3 \approx 0.0918$$

2.1.2. False Positive. Fig. 1 describes a case of false positive (type 1 error) that an element is checked in the set S but the set S does not include the element. The below mathematical procedures are necessary to figure out a false positive.

At first, when a hashed element is stored into a Bloom filter of size m, the probability of a certain bit which is not set to 1:

$$1 - \frac{1}{m}. \tag{2-1}$$

because k hash functions produce k outputs,

$$\left(1 - \frac{1}{m}\right)^k. \tag{2-2}$$

(2-2) is the probability of not setting 1 by k times.

Therefore, when n inputs are stored in the Bloom filter,

$$\left(1 - \frac{1}{m}\right)^{nk}. \tag{2-3}$$

(2-3) is the probability of set to 0 when n inputs are stored in the Bloom filter by k hash functions.

Therefore,

$$1 - \left(1 - \frac{1}{m}\right)^{nk}. \tag{2-4}$$

is the probability of non-zero bit in the Bloom filter when n·k(inputs · the number of hash functions) outputs are stored in the Bloom filter.

K hash functions hash an element and then k hash values are inserted to a Bloom filter. Therefore,

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k. \tag{2-5}$$

is the theoretical false positive probability. Even in [4] represents that (2-5) is the incorrect equation to compute false positive rate. However, previous works consider (2-5) as the false positive of standard Bloom filter. Therefore, we consider (2-5) as an approximate false positive rate of the Bloom filter in this paper.

Furthermore, (2-6) can be calculated as the approximate false positive rate:

$$\approx \left(1 - e^{-kn/m}\right)^k. \tag{2-6}$$

The minimum value of false positive is when:

$$k = \frac{m}{n} \ln 2. \tag{2-7}$$

The number of optimal hash functions can be calculated approximately through the (2-7). Therefore, the false positive rate increases under the following circumstances: 1) More hash functions; 2) Input elements increase; 3) The size of the Bloom filter decreases, as shown in (2-5).

2.2. Improved Bloom filter

In 2019, [8] presents the more than 60 Bloom filter variants with diverse aspects. In this paper, we introduce two Bloom filters that improve the Bloom filter performance because we focus on an aspect: computation optimization. These improved Bloom filters simulate k hash functions to lessen the computation time. 2.2.1. Double Hash Bloom filter. In 2006, Adam Kirsch and

Michael Mitzenmacher introduced Double Hash Bloom filter [2]. Double Hash Bloom Filter simulates X hash functions by using two hash functions.

Double Hash Function $g_i(x)$ is defined as follows:

$$h_1(x) + i \cdot h_2(x). \quad (2\text{-}8)$$

$g_i(x)$ (i = 1…, X) produces X outputs in the X iterations that simulate each X hash functions. X outputs are inserted into the Bloom filter identical method as the standard Bloom Filter. Double Hash function reduces hash functions usage by simulation and [2] demonstrates identical asymptotic false positive rate by mathematical proofs. This was derived from double hashing in Knuth's discussion [9].

2.2.2. Single Hash Bloom filter. In [3], Single Hash Bloom Filter introduces simulation hash functions by one hash function. Simulating hash function, $r_i(x)$, is defined as follows:

$$(h_1(x) \gg 16) \otimes (h_1(x) \ll i\ ). \quad (2\text{-}9)$$

$r_i(x)$ (i = 1…, X) produces X outputs in the X iterations that simulate each X hash functions. X outputs are inserted into the Bloom filter identical method as the standard Bloom Filter and the Double Hash Bloom filter. The Single Hash Bloom carries out from the Double Hash Bloom filter by reduction hash functions.

2.2.3. Regarding predictions of false positives in previous works. Through the analysis of previous works, we could predict the false positive rate. Provided that the hash functions are perfectly random, previous researches prove the false positive rate. Although previous works proposed an improvement of computation speed in a perfectly random hash functions, the following experiments present hash functions that are not perfectly random in the Bloom filter.

Table 1: Comparison experiments between Bloom filter(bf), Double Hash(dh), Single Hash(sh)

21c7	bf	dh	sh
20k	0.0057	0.0069	0.0074
40k	0.0949	0.1331	0.1346
60k	0.2776	0.3956	0.395
80k	0.4771	0.6252	0.6406
100k	0.645	0.7439	0.8029

By using equation (2-7), we can calculate the ideal number of hash functions(k) as 7 when the size of Bloom filter(m) is 200k and the number of inputs(n) was 20k. An ideal case of input (20k), false positives were slightly different between the Bloom filters at most 0.0017. However, when we inserted inputs over 20k, the probability of false positive increased. Finally, the most gap of false positive reaches over 10%.

As shown in Table 1, the practical false positive rate of Double Hash and Single Hash was distinct from the standard Bloom filter. Therefore, we assumed that hash functions are not perfectly random unlike previous works did.

3. PROPOSED SYSTEM

Fig. 2 represents an intuitive description that a circular bit shift is applied to a hash function simulation. Furthermore, the below equation represents a description of the circular bit shift as a high-level programming language.

$$(h_1(x) \gg i) \mid (h_1(x) \ll 32 - i\ ). \quad (3\text{-}1)$$

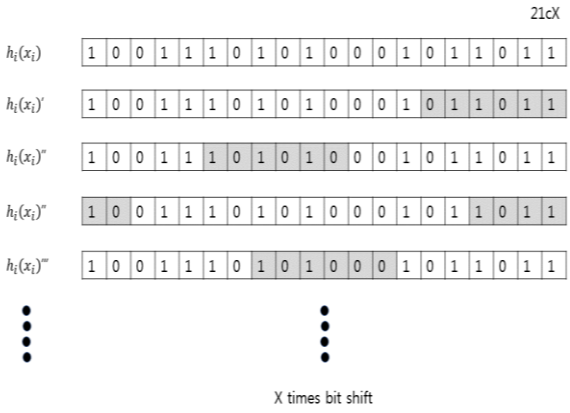

Figure 2: Simulated hash functions by the circular bit shift method

Equation (3-1) represents pseudo-code that the bit is shifted right by i-th, (i = 1…, X), addresses in every loop. Fig. 3 represents (3-1) more intuitively.

However, we could discover a problem when the equation (3-1) is applied to the implementation of the Bloom filter. As mentioned, the improvement of computation speed is the

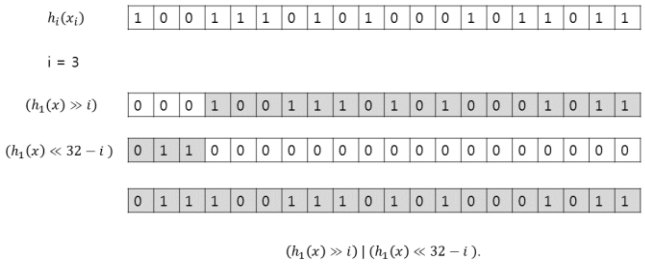

Figure 3: Intuitive description of equation (Fig. 2)

essentials in this paper. When we apply the equation (3-1) to the Bloom filter instead of (2-8) and (2-9), we should select the specific bits. (equation (2-9) is applied to the Single Hash Bloom filter no more any operations by itself.) Therefore, we use assembly language to reduce computation time. We use ROL or ROR (as known as the abbreviation of Rotate Right and Rotate Left) assembly instructions to implement the circular bit shift that uses the CF flag [11].

Table 2 represents the pseudocode of a circular bit shift in the Bloom filter. The high lightened line is including ROL/ ROR in the Table 2. As shown in (3-1), we diminish few operations to one

assembly instruction. Initial hash output d is circularly rotated every loop to simulate random hash values. To be compatible with c/c ++, there is the code '__asm' block in the middle of high-level programming language.

Table 2: Pseudo code of a circular bit shift

```
Data : k is the number of shift bits
Function: hash function h(x)
d ← h(x)
for i : 1 ... X do
    /* Loop for sequencial circular bit shift
    calculation */
    __asm
        /*As Known as assembly coding area*/
        ROR / ROL       d, k
        MOV             j, d
        SHR             j, 32-k
        end
    Bloom filter[j-1] ← 1
end
```

4. EXPERIMENTS

This section presents comparisons with the baseline of Bloom filter, standard Bloom filter, Double Hash, Single Hash, and Circular Shift Hash. Data samples consist of appended some classic books: The Bible, The Buddhist texts, etc. The total size of the appended sample is 200k lines. Input elements are inserted into the Bloom filter from 20k to 100k: 20k, 40k, 60k, 80k, 100k. X hash functions of 21 hash functions that are selected to produce hash values used in Bloom filter, baseline, Double Hash, Single Hash, circular shift hash. The list of 21 hash functions which are used in this paper is as followed:

Table 3: List of hash functions

PJW	DJB	CRC32	Ocaml
SML	STL	FNV32	BOB2
BOB3	BOB4	Hsieh	RSHash
SDBW	RS	JS	JSHash
BKDR	DJBHash	DEKHash	APHash
PJWHash			

Each of them is non-cryptographic hash functions and produces 32-bit outputs. Nonetheless, [12] recommend the cryptographic hash functions, we use only non-cryptographic hash functions for the straightforward comparisons. The hash functions selection in Table 3 is implemented by dynamic programming.

By (2-7), the number of optimized hash functions is (ln2) × (size of a Bloom filter) ÷ (input size). If calculation results are the real number, because the number of hash functions is always an integer number, they are rounded to the integer number in this paper experiments. We restricted the number of hash functions and the size of Bloom filter, because of the straightforward comparisons.

The top of each figure, there is the 21cX titles: the range of X is from 1 to 21. '21cX' states that X of 21 hash functions are used and let c denote 'combination'.

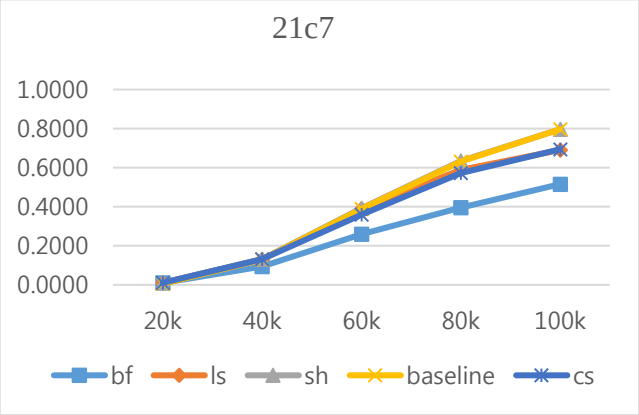

Figure 4: Comparison results in 21c7

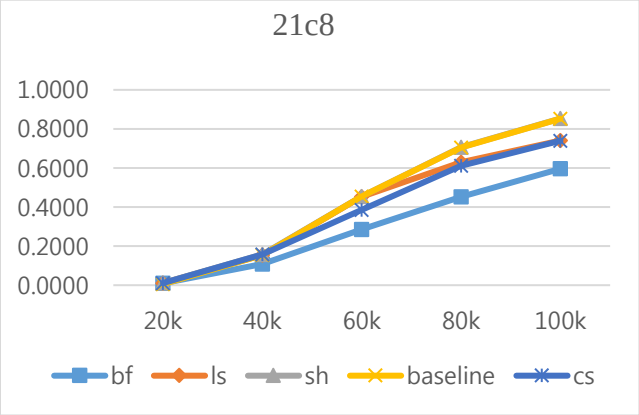

Figure 5: Comparison results in 21c8

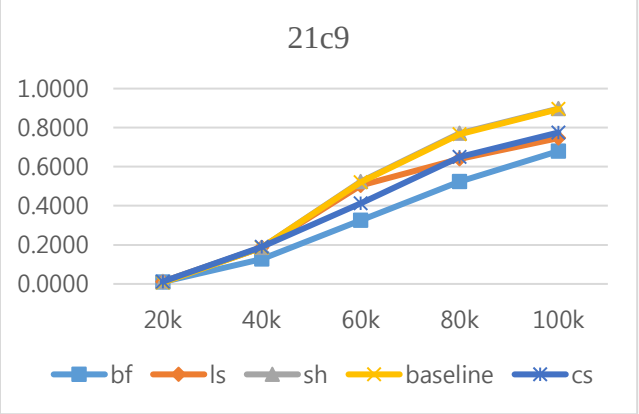

Figure 6: Comparison results in 21c9

For example, when the '21c7' is written in the top of like Fig. 4, seven hash functions are selected in Table 3 for the Bloom filter

implementation and 7 hash functions are simulated by two hash functions in a Double Hash Bloom filter or one hash function in a Single Hash Bloom filter. 'bf' denotes standard Bloom filter, 'dh' denotes Double Hash, 'sh' denotes Single Hash and 'cs' denotes circular bit shift Bloom filter.

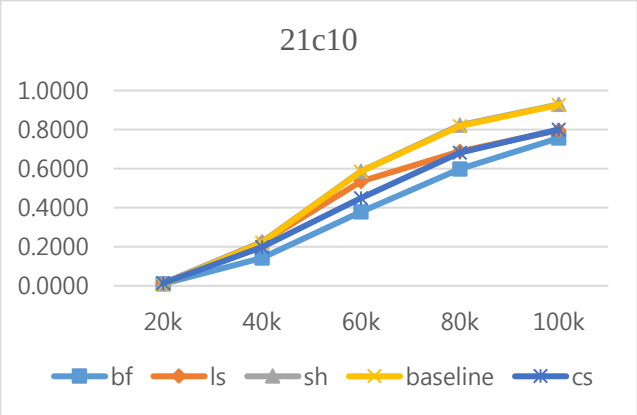

Figure 7: Comparison results in 21c10

Before we compare the experimental results with each other results, we show the description of the baseline in this paper. The definition of the baseline is referred to 'A baseline is a value or starting point on a scale with which other values can be compared' in Collins dictionary. Therefore, the baseline of the Bloom filter consisted of totally randomized inputs to 'compare' with other experiments. These pseudo hash outputs are created to insert into a Bloom filter. There is a random number range as follows: $0 \sim 2^{32} - 1$. Furthermore, we use the 'Chrono library' and 'mt19937' random number engine to create random numbers instead of the typical random creation function: rand ( ). Chrono library aids in measuring the time as nano milliseconds and mt19937 can create more distributed random numbers. However, this 'distribution' increases the probability of false positive because practical hash function outputs rarely indicate the same addresses in the Bloom filter. Therefore, it can be assumed that all false positive rates of baseline higher than any other Bloom filter.

As shown in Table 8, we present that there were more than 3000 addresses occupation on average. Therefore, the baseline of this paper represents the approximate upper limit of the false positive rate in the Bloom filter.

Table 4: False positive probability of 21c7

21c7	20k	40k	60k	80k	100k
bf	0.0098	0.0938	0.2601	0.3952	0.5157
dh	0.0102	0.1306	0.3891	0.5883	0.6904
sh	0.0108	0.1315	0.3918	0.6348	0.7967
baseline	0.0055	0.1300	0.3883	0.6313	0.7968
cs	0.0106	0.1319	0.3586	0.5716	0.6942

Table 5: False positive probability of 21c8

21c8	20k	40k	60k	80k	100k
bf	0.0098	0.1082	0.2851	0.4534	0.5960
dh	0.0106	0.1548	0.4532	0.6297	0.7406
sh	0.0116	0.1573	0.4547	0.7064	0.8534
baseline	0.0057	0.1543	0.4542	0.7042	0.8524
cs	0.0108	0.1571	0.3847	0.6117	0.7397

Table 6: False positive probability of 21c9

21c9	20k	40k	60k	80k	100k
bf	0.0100	0.1276	0.3262	0.5238	0.6799
dh	0.0117	0.1866	0.5035	0.6392	0.7464
sh	0.0120	0.1864	0.5239	0.7704	0.8975
baseline	0.0063	0.1859	0.5197	0.7668	0.8948
cs	0.0116	0.1890	0.4116	0.6493	0.7751

Table 7: False positive probability of 21c10

21c10	20k	40k	60k	80k	100k
bf	0.0101	0.1422	0.3789	0.5986	0.7566
dh	0.0123	0.2229	0.5348	0.6879	0.7970
sh	0.0132	0.2241	0.5876	0.8226	0.9288
baseline	0.0072	0.2211	0.5852	0.8179	0.9263
cs	0.0130	0.1992	0.4491	0.6815	0.8004

Table 8: counting address in a Bloom filter

21c7	20k	40k	60k	80k	100k
Practical Bloom filter					
avg	95786	144046	169596	183310	190706
min	65570	103127	128415	147003	160547
Baseline Bloom filter					
avg	100683	150681	175509	187838	193961
min	100151	150107	174948	187408	193641
n*k	140000	280000	420000	560000	700000

Now, we compare experimental results with a circular bit shift method and previous works. We showed the experimental results from Fig. 4 to Fig. 7 that standard Bloom filter almost always represented the lowest false positive rate through all the experiments. Furthermore, Table 4 represents that inserting 60k inputs resulted in at most about 10% gap between standard Bloom filter and others.

Double Hash Bloom filter represented a 10~20% higher false positive rate than standard Bloom filter. Single Hash Bloom filter represented the highest false positive rate overall experiments without 20k inputs which are the ideal situation. We applied a circular bit shift to Bloom filter represented the false positive rate lower than Single Hash Bloom filter and almost similar with Double Hash Bloom filter. Even when we inserted 60k inputs into the Bloom filter as a circular bit shift method, about 10% lower false positive rate are shown in 21c9 and 21c10. More detailed experimental results are shown in Table 3 to Table 5. However, we cannot consider experiments the conclusion that the circular

bit shift method has the lower false positive than Double Hash method or Single Hash method. Therefore, the experiments represented a false positive rate order as follows: Bloom filter < circular bit shift Bloom filter = Double Hash Bloom filter = Single Hash Bloom filter.

The following Tables present the run time measurement in the Bloom filter and the Double Hash Bloom filter, the Single Hash Bloom filter, the Circular bit shift Bloom filter. The host we implement experiments is a desktop that has an Intel Core i7-4770k and 8 GB memory. It runs Windows 7. For the accurate results, we repeat experiments 100 times and as shown 21c7. However, the standard Bloom filter has over 116280 cases in the 21c7. Compared to the Single Hash Bloom filter, the standard Bloom filter runs more than 5000 times. Therefore, we represent only three data sets in entirely run-time results. Each value is calculated by the following equation:

$$T=\text{start.clock}() - \text{finish.clock}() / \text{CLOCKS\_PER\_SEC}$$
$$\text{Run time speed} = \text{The number of input}/T*100 \quad (4\text{-}1)$$

As shown in four tables, the entire run-time speed was ordered following: Bloom filter < circular bit shift Bloom filter < Double Hash Bloom filter < Single Hash Bloom filter. A circular bit shift method is entirely faster in the maximum speed and the average speed, compared to the Single Hash Bloom filter.

Table 9: run-time speed in the 21c7

21c7		20k	40k	60k	80k	100k
bf	max	1.2500	0.9523	0.6521		
	avg	0.7331	0.6039	0.4108		
ls	max	1.1662	1.2966	1.1721	1.0710	1.0847
	avg	0.8970	1.0109	0.9018	0.8283	0.8292
sh	max	1.6543	1.7145	1.7016	1.5987	1.5716
	avg	1.3194	1.4367	1.3449	1.2570	1.2199
cs	max	1.7825	1.8744	1.7094	1.5689	1.6266
	avg	1.4139	1.5220	1.3536	1.2526	1.2561

5. CONCLUSION AND FUTURE WORKS

Hash-based structure Bloom filter is derived from facilitating membership query operations. As shown in chapter 2, previous works tried to improve the standard Bloom filter. Previous works use 1 or 2 hash functions to simulate hash functions by a simpler computation method.

Previous works represent that the computation speed of each Bloom filter is compared as follows: Bloom filter < Double Hash Bloom filter < Single Hash Bloom filter. As shown in [10], less usage of hash functions makes Bloom filter faster.

In this paper, we start by demonstrating the process of the proposed system through the experiments in Fig. 2. Even though this paper cannot proceed with the demonstration in the perfectly random condition, we propose a hypothesis that a faster computation level exists beyond previous works with similar false positive level. Furthermore, the demonstration is proceeded by experiments as shown in chapter 4 and the faster computation is proposed to implement the proposed system by using a circular bit shift. Assembly instructions are used in the high-level programming language to facilitate the implementation of a circular bit shift. As a result, computation time decreases as just one assembly instruction.

In chapter 4, we proceed with the experiments to compare with several Bloom filters. As a result, experimental results with a false positive rate in the Bloom filter are as follows: Bloom filter < circular bit shift Bloom filter = Double Hash Bloom filter =Single Hash Bloom filter. We demonstrate by these results that the proposed method computes faster with the similar false positive rate. Therefore, demonstrating the proposed system is summarized as follows:

1) Through the experiments, the false positive rate of standard Bloom filter is distinct from the previous works. Therefore, hash functions are not perfectly random

2) We compute the faster computation in the improved Bloom filter that has results similar level false positive rate;

3) We demonstrate the proposed system as experiments

In future work, perfect randomness will be focused on hash functions. A random deficiency concept will be used to demonstrate the perfect randomness of hash functions. Furthermore, analysis of binary sequences that decimal value of the hash output will be the key to mathematical proofs in future work.

References

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.
[2] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: building a better Bloom filter," in ESA'06: Proc. 14th Annual European Symposium on Algorithms. London, UK: Springer-Verlag, 2006, pp. 456–467.
[3] X. Gou, C. Zhao, T. Yang, L. Zou, Y. Zhou, Y. Yan, X. Li, B. Cui, "Single Hash: use one hash function to build faster hash based data structures," IEEE International Conference on Big Data and Smart Computing, 2018
[4] P. Bose, H. Guo, E. K. M. Smid, A. M. Y. Tang, P. Morin, "On The False Positive Rate Of Bloom Filters," Information Processing Letters, Volume 108, Issue 4, 31 October 2008, Pages 210-213.
[5] M. Zhong, P. Lu, K. Shen, J. Seiferas, "Optimizing Data Popularity Conscious Bloom Filters," PODC '08 Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing, Pages 355-364
[6] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey,"Internet Mathematics, vol. 1, no. 4, 2005.
[7] "Issue 10896048: Transition safe browsing from bloom filter to prefix set. - Code Review".
[8] L. Luo, D. Guo, R. T.B. M, O. Rottenstreich, X Luo, "Optimizing Bloom Filter: Challenges, Solutions, and Comparisons," IEEE Communications Surveys and Tutorials 21(2): 1912-1949 (2019)
[9] D. Knuth, The art of computer programming, vol. 3, sorting and searching, Addison-Wesley, Reading, MA, 1973.
[10] C. Henke, C. Schmoll, and T. Zseby, "Empirical evaluation of hash functions for multipoint measurements," SIGCOMM Computer Communications Review, vol. 38, no. 3, pp. 39–50, 2008.
[11] Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4.
[12] Q. Dang, "Recommendation for Applications Using Approved Hash Algorithms," NIST Special Publication 800-17, Revision 1, Aug 2012.