Multi-Granularities Counting Bloom Filter [1]

ZHOU Mingzhong, GONG Jian, DING Wei, CHENG Guang

[1] Dept. of Computer Science and Technology, Southeast Univ.,
Jiangsu, Nanjing 210096 China
[2]Jiangsu Province Key Laboratory of Computer Networking Technology
{mzzhou, jgong, wding, gcheng}@njnet.edu.cn

Abstract. Counting Bloom Filter is an efficient multi-hash algorithm based on Bloom Filter. It uses a space-efficient randomized data structure to represent a set with certain allowable errors, and allows membership and multiplicity queries over the set. Aiming at the set whose items frequencies following heavy-tailed distribution, this paper presents a novel algorithm called Multi-Granularities Counting Bloom Filter (MGCBF) based on Counting Bloom Filter. This algorithm applies hierarchical data structures through several counting bloom filters to store the items frequencies information in the set. The time and space complexities analysis of this algorithm illustrates that it can reduce the space needed dramatically with the cost of little additional compute-time. And the following experiments indicate this algorithm is more efficient than other algorithms with same errors probability when the items frequencies of the target set follow heavy-tailed distribution.

1 Introduction

With the pervasion of the computer and also the network, it becomes more and more serious to deal with the expanding data. Extracting useful information from very huge dataset efficiently is one of most important research points. Bloom Filter is a space-efficient multi-hash algorithm created by B. Bloom in 1970's[1], and it is widely used in database and network applications [2]. But using original Bloom Filter to satisfy the needs of all kinds applications is becoming impossible because of the protean usages and quarries of data. Several algorithms based on original Bloom Filter come forth to fit the special needs of different applications [3][4][5].

In a good few of applications, some information of dataset need measuring is provided in advance (i.e. the size and the distribution). It can improve the performances of data disposal dramatically by taking advantage of those foregone information. Heavy-tailed distributions (also known as power-law distributions) have been observed in many natural phenomena including both physical and sociological phenom-

[1] This research is partially support by the National Basic Research Program (called 973 Program), No. 2003CB314803; Jiangsu Province Key Laboratory of Network and Information Security BM2003201 and the Key Project of Chinese Ministry of Education under Grant No.105084

ena, for example, the size distribution of files transferring in network [6], the length distribution of flows intercepted in some router for a period [7], et. al. Heavy-tailed distribution means the most items frequencies are very small, while a very few items frequencies is so large that the total number of their tuples takes a large proportion of all tuples in the set. If the asymptotic shape of the distribution is hyperbolic, it is heavy-tailed regardless of the distribution for small values of the random variable.

For the dataset whose items frequencies following heavy-tailed distribution, this paper represents a novel algorithm based on Bloom Filter, called Multi-Granularities Counting Bloom Filter (MGCBF), allowing membership and multiplicity queries for individual items with quite few errors. Because the MGCBF takes advantage of the characteristic of heavy-tailed distribution given by the dataset, it can reduce the storage space (vs. CBF [3]) and calculating time (vs. SBF[4]), and improve the statistical precision (vs. SCBF [5]). This algorithm can be used in all kinds of dataset whose individual items frequencies following heavy-tailed distribution.

2 Previous works

Bloom Filter are space efficient data structures whose size is m, allowing for membership queries over a given set $S=\{s_1,s_2,...,s_n\}$. The Bloom Filter uses k hash functions, $h_1, h_2,..., h_k$ to hash elements into an array V of size m. Initially, all positions of the array V are set to 0. For each element s, the bits at positions $h_1(s), h_2(s), ..,,h_k(s)$ in the array are set to 1. Given an item q, its membership in the dataset can be checked by querying the bits at positions $h_1(q), h_2(q), ..,,h_k(q)$. If and only if all those bits are set to 1, it is reported that $q \in S$. This algorithm can cause false positive error for it may be possible that not all bits at the positions h1(q), hz(q), ..,,hk(q) are set by the item q. But no false negative error can be produced.

Counting Bloom Filter (CBF) is one of Bloom Filter's extensions. And it is introduced by L.Fan, P. Cao et. al. in [3]. This algorithm changes the bits in the array to counters, allowing estimates of the multiplicities of individual keys with a small error probability. When an element s want to insert, the counters at positions h1(s), hz(s), ..,,hk(s) in the array add 1, And it means that CBF not only supports items insertions and queries like original Bloom Filter, but also can be used for items deletions. When an element sj want to be eliminated from the set, the counters at corresponding positions in the array subtract 1 if all these counters' values are bigger than or equal 1. Otherwise it is reported that $sj \notin S$. The CBF inherits false positive errors from Bloom Filter, and also false negative errors are introduced if the counters space is not large enough. The method of counters space estimation is illustrated in [3], and the errors probability is also given.

S.Cohen and Y.Matias indicate Spectral Bloom Filter (SBF) in [4], which is another extension of original Bloom Filter. This algorithm is based on CBF, which uses a compact data structure to represent the counters array and a optimal method called recurring minimum to reduce the errors probability. But the maintenance of this compact data structure needs additional calculating time. And so the items frequencies distribution of the dataset can influence the time of data structure maintenance. To the dataset with same tuples number, the asymmetrical distribution of items frequencies

need more compute time than symmetrical ones. When the frequencies follow heavy-tailed distribution, this situation is more deteriorated. The SBF does not give any optimizations for this situation for it does not fit the needs of those type datasets.

Space Code Bloom Filter (SCBF) is an algorithm for network flow length statistics proposed by S. Cohen and Y Matias in [5]. This algorithm can be used in other applications for the same requirements. The SCBF employs several Bloom Filters to estimate the packets number of individual flows. To make the data structure more efficiency, this algorithm is extended to a new algorithm called Multi-Resolution SCBF, applying sampling and multiplies resolution layers methods to express the flows whose packets number[1] exceed some thresholds. The maximum likelihood estimation (MLE) and mean value estimation (MVE) methods are adopted to gauge all flows length information. Because of the real time requirement of network flow disposal, this algorithm uses the method of sampling. And this method cannot measure the items frequencies and their distribution accurately. It cannot satisfy the precise measurement needs of some special applications.

3 The Multi-Granularities Counting Bloom Filter

The MGCBF employs a serial of CBFs (MGCBF=$\{cbf_0,cbf_1, \ldots, cbf_{h-1}\}$), which use a set of different counter granularities (C=$\{1,c_1,c_2, \ldots, c_{h-1}\}$) to count the frequency of individual items in the dataset. The time and space complexity are main problem wanted to solve because of the long length of the sequence [2][3][4][5]. This algorithm supports related items insertions, queries and deletions, for frequencies statistical dataset whose frequencies following heavy-tailed distribution. The prototype of this algorithm is introduced as following:

(1) When an item x wanted to add into MGCBF, the counters at positions $h_1^0(x)$, $h_2^0(x)$, …, $h_{k0}^0(x)$ in the array V_0 add 1. (V_0 is the array structure of MGCBF's first CBF, cbf_0. h_1^0, h_2^0, …, h_{k0}^0 are the hash functions in cbf_0. Without the loss of generality, we suppose $h_1^0(x) \leqslant h_2^0(x) \leqslant \ldots \leqslant h_{k0}^0(x)$);

(2) The second step is checking the value $h_1^0(x)$. If $h_1^0(x)=c_1$, the counters at positions $h_1^0(x)$, $h_2^0(x)$, …, $h_{k0}^0(x)$ decrease c_1, then the values in those counters are changed to 0, $h_2^0(x)-c_1$, …, $h_{k0}^0(x)-c_1$; At the same time, the counters add 1 at positions $h_1^1(x)$,$h_2^1(x)$, …, $h_{k1}^1(x)$ in V_1 which is the array of cbf_1, that means $h_1^1(x)+1$, $h_2^1(x)+1$, …, $h_{k1}^1(x)+1$;

(3) And then checking the value $h_1^1(x)$. If $h_1^1(x)=c_2$, we operate the same action as 2) in cbf_1 and cbf_2.. This action keeps doing until cbf_{h-1} is checked. When an item x multiplicity query is need, we need get a minimum values in every layer CBF which make up of a set: M(x)=$\{min_0(x),min_1(x), \ldots, min_{h-1}(x)\}$, and then the frequency of x in S is reported as following:

$$Counter(x) = min_0(x)+min_1(x)*c_1+ \ldots + min_{h-1}(x)* \prod_{i=1}^{h-1} c_i \qquad (1)$$

[1] Packets number is used to express the flow length in that paper.

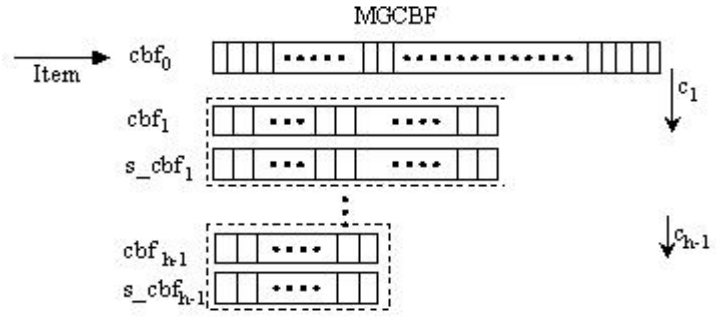

Fig.1. The MGCBF data structure

The MGCBF use different layers to statistic the items frequencies in the target dataset. The low frequent items only exist in low layers, while only very high frequent items can exhibit at every layer. When the items frequencies follow heavy-tailed distribution in the dataset, the arrays space reduces in power law as the layers increasing because only very small items are with very high frequencies. Because the counters in low layer CBFs' data structures should only maintain very small and controllable values, their space needed can reduce dramatically as the CBF. And multiply granularities makes counters values of every layer relative small, which can restrict the space needed in a small range. And so the storage resources needed in the MGCBF is relative smaller than those of the CBF, but this method will introduce additional time used for data structure maintenance and also the errors probability. Because the influences of errors caused by high-level CBFs are more important than those of low-level CBFs, the MGCBF applies an optimal method called recurring minimum introduced in [4] in high-level CBFs (cbf_1 and above) to reduce the errors probability dramatically by with the cost of a small storage resources and compute time. We will analyze the storage resources usages, calculating complexities and errors ratios in the following section.

insert (MGCBF,X)
  Initiate(MGCBF); //set every counter of each granularity CBF to zero;
  for i:=1 to N, do
    cbf_0.add(x_i, 1); //add the x_i into the granularity 0 CBF
    for j:=1 to h, do
      if($min_{j-1}(x_i)$= =c_j)
        cbf_{j-1}.remove(x_i, c_j);
        if cbf_j.isminimum
          s_cbfj.add(x_i, 1);
        else
          cbf_j.add(x_i, 1);
    return count(x_i):= $min_0(x_i)$+$min_1(x_i)$*(c_1 +)+ …+ $min_{h-1}(x_i)$* $\prod_{k=1}^{h-1} c_k$;
end_inserted

Fig.2. Insertion algorithm in MGCBF

The pseudo-code of insertion items action in the MGCBF is illustrated in Fig.2. The parameters used in that can be described as following: $\mathbf{x_1,\ldots,x_N}$**:** the incoming packets sequence;$\mathbf{cbf_0, \ldots, cbf_{h-1}}$**:** the CBFs which construct the MGCBF, h is number of stages; $\mathbf{s\_cbf_1, \ldots, s\_cbf_{h-1}}$**:** the second CBFs used in high-level of the MGCBF; $\mathbf{C=\{c_0,c_1,c_2, \ldots, c_{h-1}\}}$**:** the space of counting unit in every level; $\mathbf{M(x_i)=\{min_0(x_i),min_1(x_i), \ldots, min_{h-1}(x_i)\}}$**:** the serial made up of minimum count of the items x_i in every stage; $\mathbf{Count(x_i)}$**:** the tuples number of xi in the MGCBF.

4 Performances and errors analysis

Because the Pareto distribution is one of widely used heavy-tailed distributions, this paper employs a set whose items frequencies following a Pareto distribution to analyze the performances and errors probability. And this paper also compares these characteristics with those of the other two algorithms (CBF and SBF). Firstly, we suppose the items frequencies of the target dataset ***S*** following a Pareto distribution whose cumulative distribution function is $F(x)=1-1/x^{\alpha}[\alpha>1]$, and the individual items number in ***S*** is set to n. Using the properties of the Pareto distribution [8], we can get the expectation $E(x)=\alpha/(\alpha-1)[\alpha>1]$. And so we can infer the tuples number in ***S*** is about $n\times E(x)=n\alpha/(\alpha-1)[\alpha>1]$.

4.1 Performances analysis

In this section, we will analyze the performances in three different directions: the space complexity, the compute complexity and the error probability.

4.1.1 Space complexity analysis

When we suppose the most frequent item have P tuples in ***S***, **Equation** (2) should be guaranteed if using the MGCBF:

$$P \le C_1 + C_1 \cdot C_2 + \ldots + \prod_{i=1}^{h-1} C_i \qquad (2)$$

And then the counters number n_i in the array of cbf_i can be estimated as the below equation in the MGCBF data structure according to properties of the Patero distribution:

$$n_i = F[x > (1 \cdot C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i-1} C_i)] \cdot n = (C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i-1} C_i)^{-\alpha} \cdot n$$

Because we apply recurring minimum method to reduce the errors probabilities in high-level of the MGCBF, the space needed in level 2 and above is double of the original structure. With the supposition of $m/n=\theta$, the total space needed in level i can be inferred by the above equation which we call m_i:

$$m_i \log_2(C_i) = \theta \cdot n \cdot \log_2(2C_i) \cdot (C_1 + C_1 \cdot C_2 + \cdots + \prod_{j=1}^{i-1} C_j)^{-\alpha}$$

From this equation, we can calculate the all storage resource needed by the MGCBF to dispose the ***S***:

$$M_{MGCBF} = \sum_{i=1}^{h} m_i \log_2(C_i) \\ = \theta \cdot n[\log_2 C_1 + C_1^{-\alpha} \cdot \log_2(2C_2) + \cdots + (C_1 + C_1 \cdot C_2 + \cdots + \prod_{j=1}^{h-1} 2C_j)^{-\alpha} \cdot \log_2(2C_h)] \quad (3)$$

4.1.2 Compute complexity analysis

Firstly, we on the assumption that the mean time of one tuple's insertion, query or deletion in individual layers of the MGCBF is following: $T_0, T_1, \ldots, T_{h-1}$. According to the cumulated distribution function (CDF) of the Pareto distribution, we can calculate the ratio of items exist in cbf_i:

$$F[x > (1 \cdot C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i-1} C_i)] = (C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i-1} C_i)^{-\alpha}$$

For the item whose frequency is $f \in [C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i-1} C_j, C_1 + C_1 \cdot C_2 + \ldots + \prod_{j=1}^{i} C_j]$, the mean compute time of every item:

$$T(i) = T_1 + \frac{T_2}{C_1} + \cdots + \frac{T_i}{\prod_{j=1}^{i-1} C_j}$$

And then we can calculate the mean compute time of one tuple at following:

$$T = (1 - C_1^{-\alpha}) \cdot T_1 + [C_1^{-\alpha} - (C_1 + C_1 C_2)^{-\alpha}](T_1 + \frac{T_2}{C_1}) + \cdots + \\ [(C_1 + C_1 C_2 + \cdots + \prod_{j=1}^{h-2} C_j)^{-\alpha} - (C_1 + C_1 C_2 + \cdots + \prod_{j=1}^{h-1} C_j)^{-\alpha}](T_1 + \frac{T_2}{C_1} + \cdots + \frac{T_i}{\prod_{j=1}^{i-1} C_j}) \\ \Rightarrow T = T_1 + C_1^{-\alpha-1} T_2 + \cdots + (\prod_{j=1}^{h-1} C_j)^{-\alpha-1} T_h \quad (4)$$

4.1.3 Errors probability analysis

The errors caused by the MGCBF can be divided into two parts: (1) the inherent errors exist in the CBFs; (2) the errors caused by count numbers transferring in the layers.

About the errors of the Bloom Filter, B.Bloom analyzed it in detail when he introduced this algorithm [1]. When all n items of the target set $\boldsymbol{S}$ are inserted into the array V whose size is m, the error probability is $E = (1 - e^{-kn/m})^k$. And then we can calculate the minimal value: k=ln2*m/n, that means $(1/2)^k = (0.6185)^{m/n}$. This error ratio can be controlled by modifying the values of k and n through estimating the value m:

θ=m/n=6 k=4 E=0.0561 θ=m/n=8 k=6 E=0.0215
θ=m/n=12 k=8 E=0.00314 θ=m/n=16 k=11 E=0.000458.

The detail analysis about CBF errors probability in [4][5] indicated that the inherent errors of the CBF are equivalent with the original Bloom Filter. The experiments of [4] illustrated that the error probability can be reduce to 1/18 if using the method of recurring minimum. And so the inherent errors probability can be controlled in a very small region in the high-level of the MGCBF.

The transferring of count numbers in levels of the MGCBF makes it important to reduce the error probability caused by this method. We can suppose the inherent errors in cbf_0, cbf_1, …, cbf_{h-1} are E_0, E_1, …, E_{h-1}. And the total errors in cbf_i can be regard as the iterative of its lower levels, it can be express as **Equation** (5) :

$$E_{i-1}' = 1 - (1 - E_0)(1 - E_1)\cdots(1 - E_{i-1}) \tag{5}$$

And we can get the actual errors of every CBF of the MGCBF:

$$(E_0', E_1', \ldots, E_{h-1}') = (E_0, 1(1-E_0)(1-E_1), \ldots, 1-(1-E_0)(1-E_1)\ldots(1-E_{h-1}))$$

From the equation above, we can conclude that the errors probabilities of items frequencies estimations increase as the CBF levels increasing in the MGCBF. But the errors probabilities of high-level are controlled in a very small region, and so the errors probabilities of high frequent items does not increase remarkably.

4.2 Comparison with other algorithms

The MGCBF is designed for the dataset whose items frequencies follow heavy-tailed distribution, and it can compactly use storage and compute resources. While the traditional algorithms do not use the known information, it can induce costs of statistical information maintenance. In this section, the MGCBF will be compared with the other widely used two algorithms: the CBF and the SBF in performances and errors probabilities. Because the SCBF uses sampling and MLE estimations, the application scope and statistical errors are absolutely different with the MGCBF, and so it is not considered when we operate the comparison.

Using the CBF, the space needed is about $m\log_2 P = \theta \cdot n \cdot \log_2 P$ for the correctness assurance [3][4]. And the SBF initializes the data structure with very little space, and adjusts the space of data structure to satisfy the needs in real time [4]. The SBF use the storage spaces more efficiently with the cost of compute time for maintaining the data structure.

We set the parameters of the MGCBF as following: $\theta=m/n=12$， $k=8$， $P=65536$、$C_1=C_2=\ldots=C_h=4$、$T_1=T_2=\ldots=T_h$ and $\alpha=3$. And we also set the mean compute time for reassigning storage resources to T_a in the SBF, and the ratio of the counters whose space should be assigned is β.

According to the **Equation** (1), we can calculate the minimal value of h is $h_{min}=8$. And so we can get the storage resources used by these three algorithms as **Equation** (6) while the space complexity equation of the SBF is according to [4]. Because the SBF uses table indexes to maintain the storage data structure, it is the most space efficient algorithm of these three ones.

$$\begin{aligned} CBF &: C = 12n * log_2(65536) = 12n \cdot 16 = 192n \\ SBF &: C = \frac{\alpha \cdot n}{\alpha - 1} + o(\frac{\alpha \cdot n}{\alpha - 1}) + O(n) = \frac{3n}{2} + o(\frac{3n}{2}) + O(n) < 20n \\ MGCBF &: C = \sum_{i=1}^{h} m_i log_2(2C_i) = 12n \cdot log_2 8 \cdot [1 + 4^{-\alpha} + \cdots + 21844^{-\alpha}] \approx 34.61n \end{aligned} \tag{6}$$

The MGCBF uses a hierarchical data structure to maintain the items information, which means it needs more compute time than the CBF, while the SBF needs more

time to reassigned space for incoming items (that is to say $T_a >> T_l$). We compare the compute time of three algorithms in **Equation** (7). From these equations, we can indicate that compute time the MGCBF needs only a little more than that the CBF needed because of the heavy-tailed distribution of items frequencies in the target set. But compute time of the SBF is much more rely on the parameters T_a and β. We can infer that the parameter $\beta \in [0.5,1)$, and so its compute time needed is far more than the others.

$$\begin{aligned} CBF &: T = T_1 \\ SBF &: T = T_1 + \beta \cdot T_a \\ MGCBF &: T = [1 + C_1^{-\alpha-1} + \cdots + (\prod_{j=1}^{h-1} C_j)^{-\alpha-1}]T_1 = 1.004T_1 \end{aligned} \tag{7}$$

We check the errors probability of the cbf_i in the MGCBF, and compare it with those of the other two algorithms in **Equation** (8). The errors probabilities of these three algorithms are very small because all of them use more space and hash functions to guarantee the precision. And it is verified the inference in § 4.1.3 that the errors probabilities may increase with the levels augment of the MGCBF but this increase is controlled in a small scope. If we want to make the algorithms more efficient by reducing the storage resources and hash functions, it will introduce more errors for all these algorithms. But the precision of the MGCBF will not reduce more rapid than the other twos.

$$\begin{aligned} CBF &: E = E_0 = 0.003 \\ SBF &: E \approx E_0/18 = 0.0002 \\ MGCBF &: E'_{i-1} = 1 - \prod_{j=0}^{i-1}(1 - E_j) = 1 - (1 - 4.6 \times 10^{-4})(1 - \frac{4.6 \times 10^{-4}}{18})^{i-1} \\ &\quad = 1 - 0.997 \times 0.9998^{i-1} \end{aligned} \tag{8}$$

5 Experiments

This paper adopts five groups of datasets whose items frequencies following heavy-tailed distributions, and analyzes these datasets with three algorithms (the CBF, the SBF and the MGCBF) to compare and evaluate the performances and errors probabilities of these algorithms. The datasets are the packets coming from Abilene-III TRACE[9] and TRACE of the CERNET backbone. We apply the flow specification of 5-tuple to counting the flows by their lengths. **Fig** 3-a illustrates the distributions of five datasets: ***dif_num*** describes the individual items number in the datasets; ***mean*** indicates the mean value of individual items frequencies; ***P95*** depicts the frequency of item in 95 percentile; and ***P99*** depicts the frequency of item in 99 percentile; ***Max_val*** is used to characterize the tuples' number of the maximal frequent item in the dataset. And so from the **Fig**. 3-a , we can conclude that the individual items frequencies of all these five dataset follow heavy-tailed distribution though the items numbers of these datasets are far from the same ones.

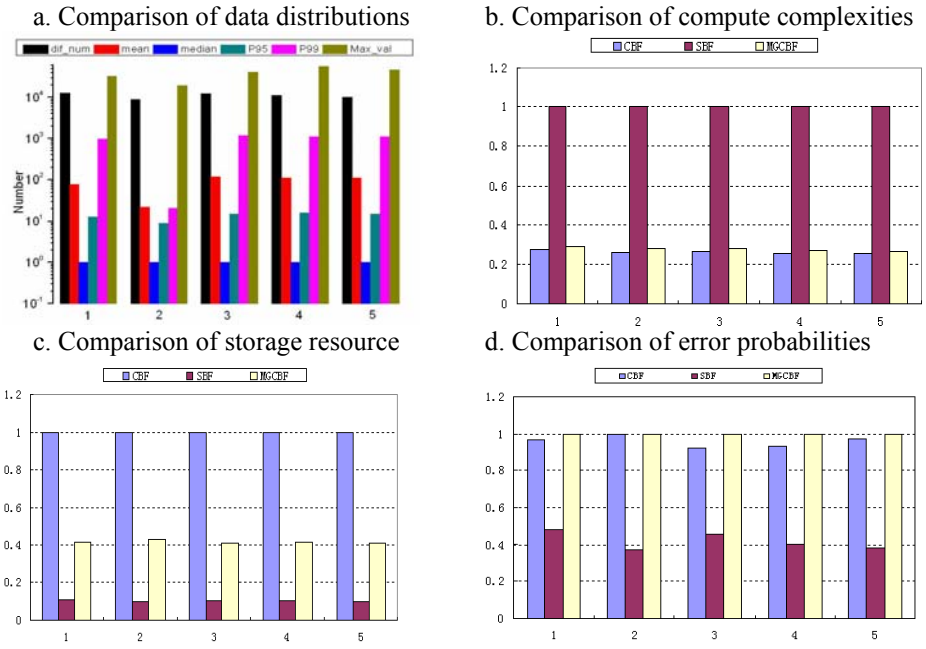

Fig.3. Performances Comparison of the CBF，SBF，MGCBF using different datasets

The data of the other figures of **Fig**.3 are normalized for the convenience of comparison. **Fig.**3-b describes comparison of three algorithms' compute performance. The performance of the MGCBF is more efficient than the SBF, and near the CBF, The reason is that the SBF need more time for data structure maintenance, which is depicted in §4.1.3 in detail. The **Fig**.3-c indicates the MGCBF saves 60% storage resources comparing with the CBF. Though the SBF is more space efficient than the MGCBF, it needs more time for calculation. The error probability of the MGCBF is almost same with that of the CBF in the **Fig.** 3-d, which indicates that the recurring minimum method reduces the errors probabilities in the high-level CBFs dramatically. The errors probabilities of the SBF are smaller than those of the other ones, but it is not prefect as described in [4]. The reason may be that the experiments applied different datasets and different hash functions. In generally, the experiments results verify the correctness of conclusion inferred in §4.

6. Conclusion

This paper introduces a novel algorithm to count the number and multiplicities of individual items in a set, which is called Multi-Granularities Counting Bloom Filter (MGCBF). This algorithm is based on the Bloom Filter, and takes advantage of the fact that the items frequencies follow heavy-tailed distribution in this target set. This

algorithm is more space efficient that traditional Counting Bloom Filter (CBF) with little compute time and few errors probabilities. And comparing to the Spectral Bloom Filter (SBF), this algorithm is more compute efficient with the cost of additional storage resources. The MGCBF not only support insertions and queries of items, but also support the items deletions, just like the SBF. And so the expansibility of the MGCBF is comparative fine. Further more, the more information about the dataset we can get, the subtler the parameters of the MGCBF we can set, and also the more precise results we can receive.

Referrence

[1] B. Bloom, Space/Time trade-offs in hash coding with allowable errors. *Commun.ACM.* Vol. 13, no.7, pp. 422-426, July 1970.

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing, 2002.

[3] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking, 8(3):281-293, 2000.

[4] S. Cohen, Y. Matias. Spectral Bloom Filters. In *Proceedings of the ACM SIGMOD 2003*. San Diego, California, USA. June, 2003.

[5] A. Kumar, J. Xu, et al. Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement. In *IEEE Infocom 2004*, Hongkong, China. March, 2004.

[6] Zhang Y, Breslau L., V. Paxson and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In Proceedings of ACM SIGCOMM, Aug. 2002, Pittsburgh, PA.

[7] A. Shaikh, J. Rexford and K. G. Shin. Load-Sensitive Routing of Long-Lived IP Flows. In *Proceedings of the ACM SIGCOMM 1999*.Cambridge, MA, USA. August , 1999.

[8] The Pareto Distribution, http://www.ds.unifi.it/VL/VL_EN/special/special12.html, Dec, 2005.

[9] The Abilene-III Trace Data – Illustrated, http://pma.nlanr.net/Special/ipls3.html, Dec, 2005.