Hotcount: A High-Precision Traffic Statistics for Multi-Tenants

1st Guanjie Qiao
School of Computer Science
National University of Defense Technology
Changsha, China
2536786901@qq.com

2nd Gaofeng Lv
School of Computer Science
National University of Defense Technology
Changsha, China

3rd Jing Tan
School of Computer Science
National University of Defense Technology
Changsha, China

4th Lusha Mo
School of Computer Science
National University of Defense Technology
Changsha, China

***Abstract*—Traffic statistics in large-scale data streams play an important role in the network community, and can be used for congestion control, anomaly detection, heavy hitter detection, etc. However, in the context of multi-tenancy, accomplishing the traffic statistics task of multiple tenants with limited resources (CPU, memory, etc) has become one of the major challenges.**

To solve the challenges above, this paper proposes a multi-tenant-oriented traffic statistics structure, named Hotcount, which can grantee the dynamic allocation of resources in multi-tenancy scenarios. At the same time, Hotcount is also able to realize multi-tenant traffic statistics task, and further improve the accuracy of the traffic statistics task. Hotcount separates the cold and hot flows based on statistics. It uses the hot/cold part to record the hot flow size with high precision and cold flow size with lower precision.

With extensive experiment, we proved that the processing speed of Hotcount is similar to that of the original classic algorithm. Meanwhile, it greatly improves the accuracy of traffic statistics tasks. In the per-flow size statistics task, the accuracy is improved by 6.7 to 49.5 times than the original algorithm. In the heavy hitter detection task, the accuracy is improved by 11.3 times to 2065.6 times than the original algorithm even with memory-size constraints.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

A. Background and Motivation

Traffic statistics in large-scale data flows play an important role in the network and can be applied to congestion control [1], anomaly detection [2], and so on. Due to the unevenness of the data flows in the network,it is known that most flows are small, referred to as cold flows,while a very few flows are large, referred to as hot flows. Hot flows have a greater impact on the network. Therefore, the high-precision statistics of the hot flows size become very important.

In order to deal with large-scale traffic statistics, various statistical structures have emerged. Sketch is a typical network traffic statistics structure based on hash calculations, which reduces the memory space usage of traffic. The most typical structure, Countmin Sketch [3], is a commonly used traffic statistics structure due to its simple structure and high accuracy. However, Countmin Sketch [3] cannot record the flow key, resulting in irreversibility, so it is difficult to distinguish between cold flows and hot flows. Later proposed statistical structures based on Sketch, such as Countmin Sketch [3]+Heap, Count Sketch [4]+Heap, Elastic Sketch [5] use a heap or filter method to store the hot flows separately, and record the hot flow size and flow key at the same time. If cold flows and hot flows are stored together, the size of the counter seriously affects the accuracy of statistics. Using a larger counter under the same memory, a smaller number of counters will cause severe hash collision. Using a smaller counter will cause the hot flows to overflow and also affect the statistical accuracy. Therefore, in order to statistic the size of the hot flows and the cold flows with high accuracy, many statistical methods for separating the cold and hot flows have been proposed. The hot flow is recorded with high precision and the cold flow is recorded roughly. However, these structures do not consider the characteristics of the flow, and use a large counter to record the cold flow, resulting in serious hash conflicts. Due to the larger counter, the statistical accuracy of both the size of per-flow and the statistical accuracy of the hot flows are seriously reduced in the case of small memory space. According to experimental statistics, about 95% of the flows are cold flows, and accurate counting can be achieved with an 8-bit counter. Therefore, compared with the previous algorithm, an 8-bit counter is used to record the cold flows, so that more counters can be used to reduce the hash collision of the cold flows. Therefore, how to achieve high-precision traffic statistics in a limited memory space is the focus of research.

With the development of multi-tenant technology [6], traffic statistics are no longer for a single tenant, and there is no research on traffic statistics for multi-tenants. Why do we need to dynamically allocate memory space for tenants? The first is that each tenant in the network has a different amount of traffic. Some tenants have large traffic, and some

ISBN 978-3-903176-39-3

have small traffic. If the same memory space is allocated to each tenant, there will be serious hash conflicts for tenants with larger traffic, resulting in decreased statistical accuracy. For tenants with small traffic, memory space will be wasted. The second is that for different tenants, the traffic characteristics are different. For example, the traffic of tenant 1 is large, and the traffic of tenant 2 is small. The hot flow of tenant 2 may not be a hot flow for tenant 1. If all tenants count the traffic together, the hot flows of tenant 2 will be judged as cold flows, which affects the accuracy of statistics. With the development of multi-tenant technology, how to complete the traffic statistics task of multiple users with limited resources, and how to allocate memory and resources for different tenants are the focus of research.

B. Our Proposed Approach

Based on the requirements mentioned above, this paper proposes a multi-tenant-oriented statistical structure Hotcount. This paper proposes a statistical resource allocation scheme in the case of multi-tenant, which can dynamically divide the memory space for each tenant. The statistical structure Hotcount of cold and hot separation is proposed. The hot part uses a two-dimensional array, and each bucket stores the key value and count of the hot flow. The hot part records the hot flows size with high precision, and the cold part also records the cold flows size more accurately.Optimized version of hot flows second hash is proposed to solve the hot flow conflict, and the high precision of the hot flow can be guaranteed even when a smaller counter is used in the cold part. A conservatively updated cold and hot flow exchange strategy for the cold part is proposed to reduce the impact on the cold flow accuracy when exchange occurs.In addition, this paper also proposes an update strategy in the case of high-speed traffic to ensure higher throughput, and proposes a scalable Hotcount structure to adapt to a small memory space and achieve high-precision hot flow statistics.

This paper implements two statistical tasks, per-flow size estimation and heavy hitter detection. The experimental results show that the accuracy of the two statistical tasks is significantly improved compared to the previous scheme.

C. Contribution

- This paper proposes a memory space allocation strategy when multiple tenants complete traffic statistics tasks at the same time.
- This paper proposes Hotcount, which separates the cold flow and the hot flow, and can estimate the flow size with high accuracy.
- This paper counts traffic characteristics and sets a suitable counter for cold flows to reduce the impact of hash collisions on accuracy.
- This paper has conducted extensive experiments, and the results show that Hotcount has the highest accuracy in multiple statistical tasks.

II. RELATED WORK

A. Sketch principle

In network statistics, network flows are mainly classified according to quintuples. The main statistical method is to allocate a counter for each flow to ensure the accuracy of statistics. The problem is that the memory space is very complicated. The sampling method infers the total amount of network traffic by selecting some flows, reducing the amount of data statistics, and the problem that it brings is the decline of statistical accuracy. In order to weigh the accuracy of traffic statistics and memory space, the hash-based traffic statistics structure Sketch came into being. Sketch maps a larger amount of data to a smaller storage structure, reducing memory space and ensuring the accuracy of traffic statistics.

Sketch is a sub-linear data structure based on hash, which is often used for traffic statistics. Through different hash functions, flows with the same hash value are recorded in the same bucket. It does not need to store all the information of the flow, only the count of flow, thereby reducing memory overhead. You can obtain traffic statistics data through query operations. To this end, many Sketch-based algorithms such as Countmin Sketch [3], Count Sketch [4], Conservative Update Sketch [7], etc. are proposed.

B. Research on multi-tenant problems

Multi-tenant technology is a software architecture technology that realizes the sharing of the same system or program components in a multi-user environment and ensures the isolation of data between individual users. Multi-tenant resources are dynamically created according to service requests. The service provider should dynamically deploy according to the agreement to meet the needs of tenants.

III. TRAFFIC STATISTICS FOR MULTI-TENANT

According to the requirements put forward in Background and Motivation, this section introduces statistical techniques for multi-tenant, and in section A introduces statistical architecture for multi-tenancy. Section B introduces the data structure and update and query operations of Hotcount. Section C introduces the conservatively updated cold and hot flow exchange strategy.

A. Statistical architecture for multi-tenancy

When the flows arrive, the tenants are divided according to the VLAN ID, the flows of different tenants are stored in different buffers, and the tenant information is recorded in the resource allocation table. After that, Hotcount will allocate memory for different tenants according to the resource allocation table, and complete the statistical task by hashing the quintuple of the flow to the corresponding position. The statistical process is shown in Figure 1.

When a flow of tenants arrives, controller determine whether the tenant is in the resource allocation table, and if so, controller allocate it to the corresponding buffer. If the tenant is not in the resource allocation table, the resource allocation table records the tenant and allocates the memory space of the hot part to the tenant.

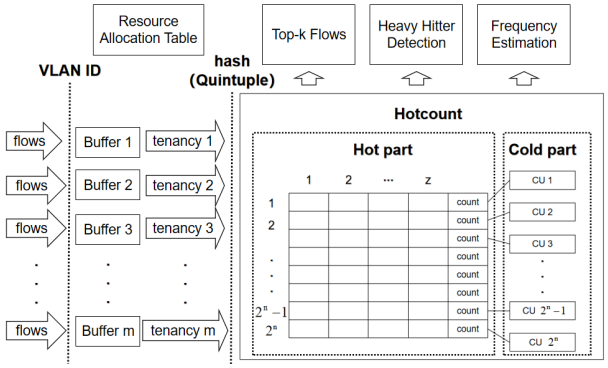

Fig. 1. Hotcount implements multi-tenant flow statistics tasks

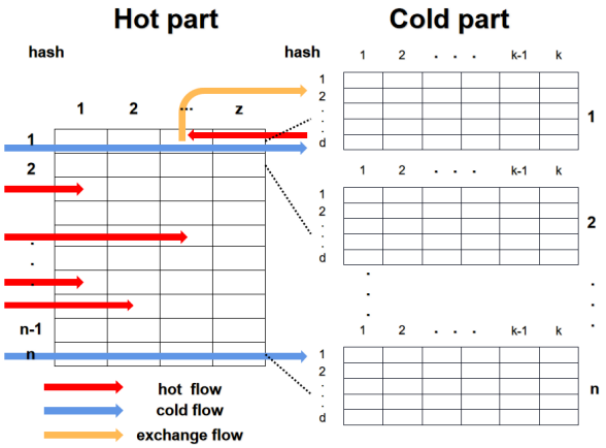

Fig. 2. Architecture of Hotcount

When a new tenant arrives, controller first update the resource allocation table and divide tenant memory space into two according to the resource allocation table. Controller merges the CU sketch [7] of row 1 and row $2^{n-1}+1$, and when the two CU sketches [7] are merged, the counters at the same position in the two CU sketches [7] take the minimum value and store them in the merged CU sketch [7].Then controller compare the $2*z$ hot flows in row $2^{n-1}+1$ and row 1 to filter out the largest z hot flows and store them in the hot part in row 1, and store the remaining flows in the CU Sketch [7] corresponding to row 1. The rest of the lines are as above. Because the hot part has 2^n line, it is guaranteed that the hot flow can be stored in the corresponding bucket after the merger. When a new tenant arrives, controller update the resource allocation table, and then calculate the average count of the flow in each tenant in the hot part, and divide the hot part memory space of the tenant with the smallest flow average count into two for new tenants to use. The parts that need to be cleared are merged into the corresponding positions.

B. Architecture of Hotcount

Section A introduces the memory allocation of Hotcount in the case of multi-tenant. This section introduces the structure of Hotcount of a single tenant.

1) Data structure:

Hotcount is composed of a hot part and a cold part. The "hot part" records the hot flows and the "cold part" records the cold flows. The n rows of the hot part are associated with n CU Sketches [7] in the cold part. The Hotcount structure is shown in Figure 2.

Hot part: Consists of n rows, respectively associated with n cold parts CU Sketch [7], each row contains z buckets, and records the flow key value and flow frequency of z hot flows. When a flow arrives, the hash function is used to determine which row of the hot part the flow is mapped to.

Cold part: Contains n CU Sketches [7], which correspond to a row in the hot part. Each CU Sketch [7] consists of d rows and w buckets, and each bucket uses a 8-bit counter to record the cold flow.

2) Key operations:

Update: Assuming that the arriving flow ID is f, controller hash it to the i-th row of the hot part ($0 \leq i \leq n$), then check the z hot buckets in the i-th row, (z is the number of hot buckets included in each row of the hot part). If f in the j-th bucket, ($0 \leq j \leq z$), controller add one to the j-th bucket flow count. Otherwise, controller check whether there is an empty bucket. If there is an empty bucket, controller insert the flow f into the empty bucket, record the flow key, and record the flow count value as one. If there is no empty bucket, controller expel flow f to the i-th CU Sketch [7] corresponding to the cold part, and only update the count value of the smallest counter in the d positions corresponding to the d hash functions. If the minimum value of the corresponding position after the update is greater than the count value of the smallest hot bucket in the i-th row in the hot part, flow f replaces the flow in the smallest hot bucket, and the corresponding counters of flow f in the cold part are all subtracted from flow f during exchange. The minimum query value, and the minimum hot flow is mapped to the cold part.

Details are as follows:

Case 1: When the flow f is stored in the bucket in the hot part, the flow f count value is increased by one.

Case 2: When there is an empty bucket in the hot part, controller insert (f, 1) to the empty position, and the insertion ends.

Case 3: The hot bucket is full, the flow f is updated to the corresponding CU Sketch [7], and the minimum value of the update value is less than the minimum value of the hot bucket, and the update ends.

Case 4: The hot bucket is full, and the flow f is updated to the corresponding CU Sketch [7]. The minimum value of the updated value is greater than the minimum value of the hot bucket. Controller replace the minimum hot flow in the hot bucket with the ID of the flow f and the updated minimum value, and the flow in CU Sketch [7] . The bucket corresponding to f is subtracted from the minimum value, and the minimum hot flow is updated to the corresponding position of CU Sketch [7].

Query:

Algorithm 1 Stream update algorithm for Hotcount

Input: key of flow$f$

function UPDATE HOTCOUNT
  $i \leftarrow$ hash(key) ($1 \leq i \leq n$)
  if f in Hotpart(i) **then**
    V(key$=f$) + 1
  else
    Check if there are empty buckets in the Hotpart
    if Ture **then**
      K(key=empty) $\leftarrow f$
      V(key=empty) $\leftarrow$ 1
    else
      Send f to Coldpart
      Query min$V$(h1(f)),V(h2(f)),V(h3(f)),V(h4(f))
    end if
    for $j \leftarrow$ 1 to 4 **do**
      if V(hj(f))= min **then**
        V(hj(f)) + 1
      end if
      if min+1$\geq V$ $k$min **then**
        for $j \leftarrow$ 1 to 4 **do**
          V(hj(f))$-$min+1
        end for
        (f $k$min,V $k$min)$\leftarrow$(f,countmax)
        Send f $k$min to Coldpart
      end if
    end for
  end if
end function
=0

Case 1: If f is in the hot part, controller return the counter value of the corresponding position in the hot part.

Case 2: If f is not in the hot part, controller find and return the minimum value of f in the cold part.

Algorithm 2 Stream query algorithm for Hotcount

Input: key of flow$f$
Output: value of flow f

function QUERY HOTCOUNT
  if f in Hotpart **then**
    $V \leftarrow V$(key$=f$)
  return V
  else
    Query f in Coldpart
    $V \leftarrow$min(V(h1(f)),V(h2(f)),V(h3(f)),V(h4(f))
  return V
  end if
end function=0

C. Conservatively updated cold and hot flow exchange strategy

When a flow is replaced from the hot part to the cold part, the previous strategy will hash the flow to the corresponding position according to the key value of the flow, and then increase the flow count to each corresponding bucket.However, directly increasing the count will bring a greater loss of precision, so in the exchange strategy of Hotcount, controller determine the count of the flow replaced to the cold part and the count of the corresponding bucket flow in CU Sketch [7].If the count value of the corresponding bucket is greater than the count of the replaced flow, no action is taken, because this indicates that the bucket is a bucket with serious hash conflicts. If the larger count value of this bucket is caused by a flow, then this flow should be stored in the hot part, not the cold part counter. Therefore, this bucket has a large hash conflict, so no action should be taken. If the count value of the corresponding bucket is less than the count of exchanged flow, then exchange the count value of the corresponding position counter with the high-precision count of the exchanged flow instead of increasing the count value. Above operation can ensure that the flow is stored in the cold part and counted with high accuracy just after the exchange. The optimization of cold and hot flow exchange strategy is shown in Figure 3.

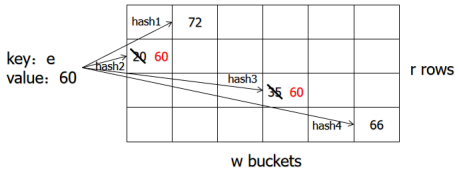

Fig. 3. Optimized version of cold and hot flow exchange strategy

IV. ANALYSIS

In this section,we analyze the accuracy of Hotcount in section A and hot flows collision rate in section B.

A. Accuracy Analysis

Hotcount is more accurate in most cases, because Hotcount uses the hot part to record the hot flows and the cold part to record the cold flows. In the hot part, because the ID and count of the hot flows are stored, for the hot flow that has not been replaced, the count in the hot part is the true frequency of the hot flows. For the hot flow that has been replaced, the counting accuracy in the hot part is slightly reduced. The count includes the true frequency of the hot flows and the count sum of a small number of cold flows due to hash collisions when the hot flows are stored in the cold part. Since the cold flow frequency has little effect on the hot flow frequency, the accuracy of the hot part is high.

In the cold part, Conservative Update Sketch [7] is used, and only the frequency of the flow is recorded. The use of four independent hash functions greatly reduces the probability of hash collisions. Because most of the hot flows are stored in the hot part, 8-bit counters can be used in the cold part. Since the usual Sketch structure cannot separate the hot flows and the cold flows, a larger counter is usually needed to prevent the flow frequency count from overflowing, and 32-bit counters are generally used. Therefore, under the same

memory space, the cold part of Hotcount can allocate more counters, which reduces the probability of hash collisions and improves the accuracy of flows frequency statistics.

Hotcount accuracy will decrease when hot flows conflict occurs. Hot flows conflict means that two or more hot flows are mapped to the same bucket in the hot part, causing a relatively small hot flows to be expelled to the cold part, making the count of cold flow in the cold part is overestimated.In Section 4.2, the probability of hot flows conflict will be analyzed. In order to reduce the impact of hot flows conflict on accuracy, the hot part adopts a storage method of multiple hot buckets to reduce the probability of hot flows conflict.

B. Hot flows collision rate

For each bucket in the hot part of Hotcount, the probability of hot flows conflict is:

$$p = 1 - \left(\frac{n}{w} + 1\right) * e^{-\frac{n}{w}}$$

Where n is the number of hot flows and w is the number of buckets in the hot part.

Proof. There are totally H hot flows, and each flow is randomly mapped to a certain bucket by the hash function.Given an arbitrary bucket and an arbitrary flow, the probability that the flow is mapped to the bucket is $\frac{1}{\mathrm{w}}$.

Therefore,for any bucket, the number of hot flows that mapped to the bucket Z follows a Binomial distribution$\mathrm{B}\left(\mathrm{n}, \frac{1}{\mathrm{W}}\right)$ .When H and w is large, then Z approximately follows a Poisson distribution,$\pi\left(\frac{\mathrm{n}}{\mathrm{W}}\right)$.

$$\mathrm{p}\{\mathrm{Z}=\mathrm{i}\}=\mathrm{e}^{-\frac{\mathrm{n}}{\mathrm{w}}} \frac{\left(\frac{\mathrm{n}}{\mathrm{W}}\right)^{\mathrm{i}}}{\mathrm{i}!}$$

There are hot collisions within one bucket $if\ Z \geq 2$ for this bucket. Therefore, we have

$$\mathrm{p} = 1 - p\{Z=0\} - p\{Z=1\} = 1 - \left(\frac{n}{w} + 1\right) * e^{-\frac{n}{w}}$$

V. EXPERIMENTAL RESULTS

In this section, we provide experimental results of Hotcount.We describe the experiment setup in Section A.We show the accuracy and throughput of the per-flow size estimation task in Section B. We show the accuracy and throughput of the heavy-hitter detection task in Section C.All abbreviations used in the evaluation and their full name are shown in Table 1.

A. Experimental Setup

Implementation: We have implemented Hotcount [11] and all other algorithms in C++. The hash functions are implemented using the 32-bit Bob Hash.

Datasets: We use four one-hour public traffic traces collected in Equinix-Chicago monitor from CAIDA [12]. We use the CAIDA [12] trace with a monitoring time interval of 5s as default trace, which contains 1.1M to 2.8M packets with 60K to 110K flows (SrcIP). Due to space limitations, we only show the results with the source IP as the flow ID; the results are qualitatively similar for other flow IDs (e.g., destination IP, quintuple).

TABLE I
ABBREVIATIONS IN EXPERIMENTS

Abbreviation	Full name
CMS	Count-min Sketch [3]
CS	Count Sketch [4]
CUS	Conservative Update Sketch [7]
ES	Elastic Sketch [5]
SS	Space Saving [8]
UM	Univ-Mon [9]
CS+H	Count Sketch [4] with a heap
CMS+H	Count-min Sketch [3] with a heap
HP	Hash-pipe [10]

Computation Platform: We conduct all the experiments on a machine with IntelRCoreTMi7-105 10U CPU@1.80GHz 2.30 GHz and16GB DRAM memory.

Evaluation metrics:

1)ARE (Average Relative Error): $\frac{1}{n}\sum_{i=1}^{n}\frac{|f_i-\hat{f}_i|}{f_i}$, where n is the number of flows, andf_i and$\hat{f}_i$are the actual and estimated flow sizes respectively.We use ARE to evaluate the accuracy of per-flow size estimation and heavy hitter detection.

2)Throughput:million packets per second (Mpps). We use Throughput to evaluate the processing speed of per-flow size estimation and heavy hitter detection.

B. Experiments on per-flow size estimation

Parameter settings: Through experiments, the hot flow accounts for about 5.8% of the total flow, and 50KB of memory is allocated to the hot part. With a total memory of 0.2MB, the number of counters allocated by the hot part accounts for 4% of the total number of counters. Under different memory, only the cold part counter quantity is changed. 4 hash functions and 8-bit counters are used in the cold part. For each algorithm in per-flow size estimation, the default memory size is 0.6MB.

We compare five approaches:CMS,CS,CUS, ES and Hotcount.

ARE: Experiments were performed on five algorithms with memory sizes of 0.2MB, 0.4MB, 0.6MB, 0.8MB, and 1MB. The results are shown in Figure 4.

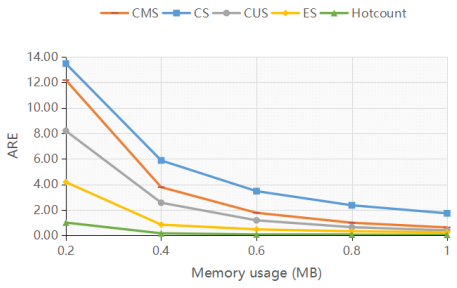

Fig. 4. ARE of per-flow size estimation

Throughput: In the experiment, the throughput of each algorithm was tested five times under the memory size of

0.6MB, and the experimental results are shown in Figure 5.

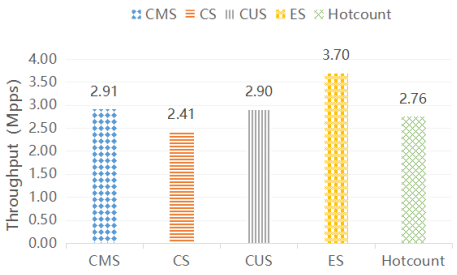

Fig. 5. Throughput of per-flow size estimation

Analysis: We find that Hotcount offers a better accuracy than CMS,CS,CUS and ES.With 0.6MB of memory, the ARE of Hotcount is only 0.07. It is 49.5 times lower than Count Sketch [4], the algorithm with the highest ARE in the experiment, and 6.7 times lower than Elastic Sketch [5], the algorithm with the lowest ARE in the experiment.The throughput of the best-performing algorithm Elastic Sketch [5] is only 1.3 times that of Hotcount.

C. Experiments on heavy hitter detection

Parameter settings: Through experiments, the hot flow accounts for about 5.8% of the total flow, and 50KB of memory is allocated to the hot part. With a total memory of 0.2MB, the number of counters allocated by the hot part accounts for 4% of the total number of counters. Under different memory, only the cold part counter quantity is changed. 4 hash functions and 8-bit counters are used in the cold part. For each algorithm in heavy hitter detection, the default memory size is 0.6MB.

We compare seven approaches:CMS+H,ES,UM,HP,CS+H,SS and Hotcount.

ARE: Experiments were performed on seven algorithms with memory sizes of 0.2MB, 0.4MB, 0.6MB, 0.8MB, and 1MB. The results are shown in Figure 6.

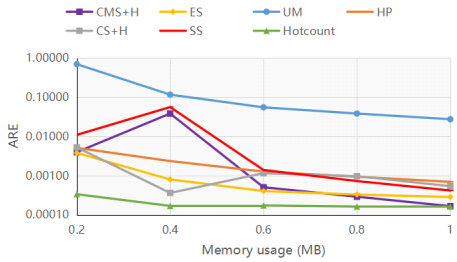

Fig. 6. ARE of heavy hitter detection

Throughput: In the experiment,the throughput of each algorithm was tested five times under the memory size of 0.6MB,and the experimental results are shown in Figure 7.

Analysis: We find that Hotcount offers a better accuracy than CMS+H,ES,UM,HP,CS+H and SS.The ARE of Hotcount is very low when the memory is small. Experimental results show that under 0.2MB of memory, the ARE of Hotcount

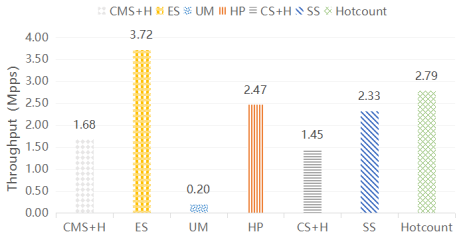

Fig. 7. Throughput of heavy hitter detection

is only 0.000335.It is 2065.6 times lower than Univ-Mon [9], the algorithm with the highest ARE in the experiment, and 11.3 times lower than Elastic Sketch [5], the algorithm with the lowest ARE in the experiment. The performance of Hotcount in processing heavy-hitter detection tasks is higher than most algorithms, and the throughput of the best-performing algorithm Elastic Sketch [5] is only 1.3 times that of Hotcount.

VI. Conclusion

Hotcount, a large-scale data stream size statistics structure oriented to multi-tenant division, solves the memory resource allocation problem in multi-tenant traffic statistics tasks. By separating the cold and hot flows, high-precision hot flows statistics are realized, and the cold flows statistics accuracy is further improved. Various update strategies are proposed to greatly improve the accuracy of statistics while ensuring the processing speed.

References

[1] M. ALLMAN, "Tcp congestion control," *Rfc*, 2009.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *Acm Computing Surveys*, vol. 41, no. 3, 2009.

[3] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2004.

[4] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, 2004.

[5] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 561–575. [Online]. Available: https://doi-org-s.nudtproxy.yitlink.com/10.1145/3230543.3230544

[6] J. Mudigonda and B. Stiekes, "Net lord: A scalable multi-tenant network architecture for virtualized datacenters," *Acm Sigcomm Computer Communication Review*, vol. 41, no. 4, pp. p.62–73, 2011.

[7] C. ESTAN and G. VARGHESE, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *Acm Transactions on Computer Systems*, vol. 21, no. 3, pp. p.270–313, 2003.

[8] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Database theory :*, Edinburgh, Scotland, 1 2005, pp. 398–412.

[9] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *the 2016 conference*, 2016.

[10] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," 2016.

[11] https://github.com/2536786901/Hotcount.

[12] "The CAIDA Anonymized Internet Traces," http://www.caida.org/data/overview/.