Parallelizing Count-Min Sketch Algorithm on Multi-core Processors

BOWEN Yu[1], YU Zhang[2,*] and LUBING Li[1]

College of Computer and Control Engineering, Nankai University, China

[1]{bowenyu, lubinli}@mail.nankai.edu.cn

[2]zhangyu1981@nankai.edu.cn

Keywords: Count-Min sketch, Parallel algorithms, Frequent items.

Abstract. In this paper, we present a novel method that exploits the great parallel capability of multi-cores to speed up the famous Count-Min sketch algorithm. The proposed parallel Count-Min sketch algorithm equally distributes the input data stream into sub-threads which use the original Count-Min sketch algorithm to process the sub-streams. The counters in each local Count-Min sketch with frequency increments exceeding a pre-defined threshold are sent to a merging thread which is able to return the estimated frequencies satisfying the (ε, δ)-approximation requirement. Experiments with real traffic traces demonstrate the excellent performance as well as the effects of parameters. The parallel Count-Min sketch algorithm achieves near-linear speedup at the cost of greater memory use.

Introduction

Sketch-based algorithms [1] have been commonly used in high-speed network traffic monitoring and measurement applications, such as heavy-hitter identification [2], DoS and port scan detection [3], network change detection [4], and network-wide traffic anomaly detection [5]. In this paper, we focus on the well-known *Count-Min* (CM) sketch algorithm [6] which has proved to be a versatile summary for frequency-based properties of a stream that can be used for answering a variety of queries on the input stream, including point and range queries, quantiles, and heavy hitters, among others. However, as the network bandwidth grows exponentially [7], the ever-increasing volume of network traffic calls for a much higher per-packet processing speed. In the CM sketch, an update requires several hash calculations and memory accesses, and these operations must be performed for every packet. In practice, the per-packet processing on backbone links needs to be accomplished within time scales of a few nanoseconds. In this paper, we propose a simple method to parallelize the CM sketch algorithm on a commodity multi-core machine.

Problem Definition

We model the data stream $S = (p_1,p_2,...,p_t,...)$, where $p_t = (v,c_{v,t})$, $v \in U$ ($U = \{1...k\}$, k is a positive integer) is the item name, t is the timestamp, $c_{v,t} > 0$ is the weight of the item. We assume the items are ordered by timestamp. In the rest of the paper, we drop reference to t when we are interested only in the "current" values of frequencies. To process a stream in parallel, this paper uses the independent per-hardware-thread data structures. Suppose there are m+1 hardware threads in the multi-core processor. The stream S is equally partitioned into m sub-streams $S_1,S_2,...,S_m$. Each sub-stream S_i consists of a sequence of tuples $(v,c_{i,v,t})$ ordered by timestamp t, where $1 \le i \le m$ is the ID of the sub-stream, $v \in U$ is the item name, $c_{i,v,t} > 0$ is the weight of the item. Each hardware thread i ($1 \le i \le m$) processes S_i and maintains a local CM sketch D_i. The hardware thread m+1 maintains one global CM sketch D which is produced from $D_1,...,D_m$. The objective is for D to continuously guarantee that with probability at least $1-\delta$, $f_v \le \hat{f}_v \le f_v + \varepsilon N$, where f_v is the true frequency of item v, is the estimated frequency of item v given by D, N is the total frequencies of all items in S, $\varepsilon > 0$ is the user-specified error bound, and $\delta > 0$ is the user-specified probability of failure.

Parallel Count-Min sketch algorithm

In this section, we show how to effectively parallelize the CM sketch algorithm. The basic idea is to simulate sequential execution and run multiple copies of CM sketches in different hardware threads. Each hardware thread has its own independent CM sketch for storing the frequencies of items. If there are m hardware threads simultaneously running, then there are m different CM sketches. According to whether merging the CM sketches is required for answering the queries, the schemes are classified into two categories. In the first hash-based partitioning, the same items go to the same threads, thus there is no need for merging. However, in the real world the distributions of data streams are usually skewed, thus the threads processing the frequent items will be overloaded, while others will be lightly loaded. Therefore it suffers from problems in load balancing. In the second block-based partitioning (considered in this paper), the same items might go to different threads. In this case, merging the CM sketches is required for answering queries. One simple way is to periodically (e.g., every other second) merge the CM sketches into a global one. However, periodically merging can't provide continuous query answers. This is because that only at the time of merging, the accuracy of returned answers can be guaranteed. Moreover, the merging overhead is dependent on the merging frequency, greater the merging frequency and higher is the overhead. Here we propose a novel merging method which is not only capable of providing continuous answers, but also exhibits very low merge overhead, therefore achieves much higher processing throughput.

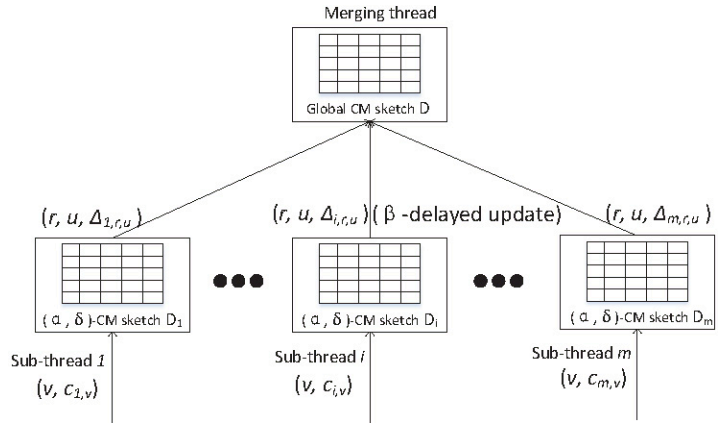

Fig. 1. Overview architecture of parallel CM sketches

The overview architecture is shown in Fig. 1. The system consists of m+1 hardware threads (including the merging thread). Each sub-thread i ($1 \le i \le m$) observes a continuous sub-stream of updates and communicates with the merging thread under certain condition to ensure the accuracy of estimated frequencies. All sub-threads run the same original CM sketch algorithm and each maintains a CM sketch locally, i.e., an (α, δ)-CM sketch (a 2-d array with $\lceil \ln(1/\delta) \rceil$ rows and $\lceil e/\alpha \rceil$ columns) and a set of pair-wise independent hash function $h_1,...,h_d$. It is important to note that all CM sketches share the same set of hash functions. The merging thread maintains a global CM sketch with same size of sub-threads. Precisely, the protocol of parallelizing the (ε, δ)-CM sketch algorithm can be described as follows.

First, each sub-thread i ($1 \le i \le m$) maintains an (α, δ)-CM sketch D_i (with the same set of pairwise independent hash function $h_1,...,h_d$) for the items in sub-stream S_i and a 2-d array Δ_i with the same size. D_i and Δ_i are initially all 0. When a new update $(v, c_{i,v})$ arrives, for each row $1 \le j \le d$ in arrays D_i and Δ_i, set $r = j$, $u = h_j(v)$, $D_i[r, u] = D_i[r, u]+c_{i,v}$, $\Delta_i[r, u] = \Delta_i[r,u]+c_{i,v}$. If $\Delta_i[r, u] \ge \beta N_i$, then set $\Delta_{i,r,u} = \Delta_i[r, u]$, sub-thread i sends a message $(r, u, \Delta_{i,r,u})$ to the merging thread, and resets $\Delta_i[r, u] = 0$, where $0 < \alpha < \varepsilon$ denotes the error tolerance, $\beta = \varepsilon - \alpha$ denotes the delayed update coefficient, ε denotes the user-specified error bound, δ denotes the user-specified probability of failure, N_i denotes the total frequencies of all items in S_i until now.

Second, the merging thread maintains a global CM sketch D with $\lceil \ln(1/\delta) \rceil$ rows and $\lceil e/\alpha \rceil$ columns. D is initially all 0. When receiving a message $(r, u, \Delta_{i,r,u})$ from sub-thread i ($1 \le i \le m$), it updates D by adding $\Delta_{i,r,u}$ to $D[r, u]$, i.e., $D[r, u] = D[r, u] + \Delta_{i,r,u}$.

Third, the parallel CM sketch algorithm answers a query about an item v by reporting

$$\hat{f}_v = \min\{D[j, h_j\{v\}] \mid j = 1, ..., d\} + \beta N, \quad (1)$$

where N denotes the total frequencies of all items in S.

In this parallel CM sketch algorithm, sub-thread i will not report the counters in D_i to the merging thread until $\Delta i[r, u] \geqslant \beta N_i$ which greatly reduces the communication overhead. We call this β-delayed update. That is to say, our approach does not periodically merge all the CM sketches, just updates the counters in the CM sketches violating the threshold. In fact we will see the communication overhead decreases rapidly with time, therefore it will not become a performance bottleneck.

In the implementation, we used a shared bounded buffer implemented with semaphores to handle the communication between the threads. The sub-threads generate messages which are processed by the merging thread, therefore from this perspective, the sub-threads and merging thread develop a similar *producer-consumer* relationship. Two counting semaphores *semmesgs* and *semslots* are used to represent respectively the number of messages in the buffer and the number of free slots. When a sub-thread needs to send a message to the merging thread, it first calls the sem wait to decrement *semslots*, and then acquires a lock on the buffer, inserts the message into the buffer, at last releases the lock on the buffer and calls the sem post to increment *semmesgs*. Correspondingly, the merging thread needs to receive the messages sent by the subthreads, it first calls the sem wait to decrement *semmesgs*, and then gets a message from the buffer, at last calls the sem post to increment *semslots*.

Evaluation

In our experiments, we used the parallel CM sketch algorithm to estimate the traffic volume (i.e., the total number of bytes) of each flow defined by the famous two-tuple (i.e., source IP address and destination IP address). Three real traffic traces were used in the experiments for illustration purposes. In the experiments, ε ranged from 0.0001 to 0.0005 (5 different values), $\delta = 0.1$, m ranged from 2 to 7 (6 different values), β ranged from 0.1ε to 0.9ε (9 different values), $\alpha = \varepsilon - \beta$.

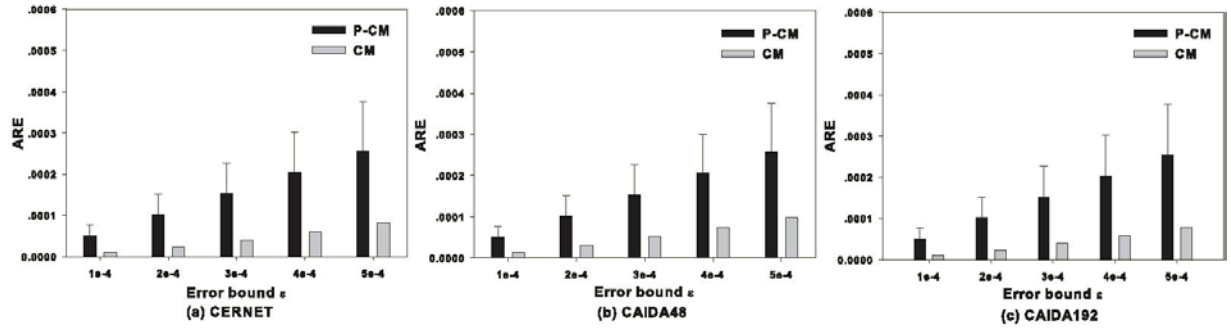

Fig. 2 Average relative error of P-CM and CM

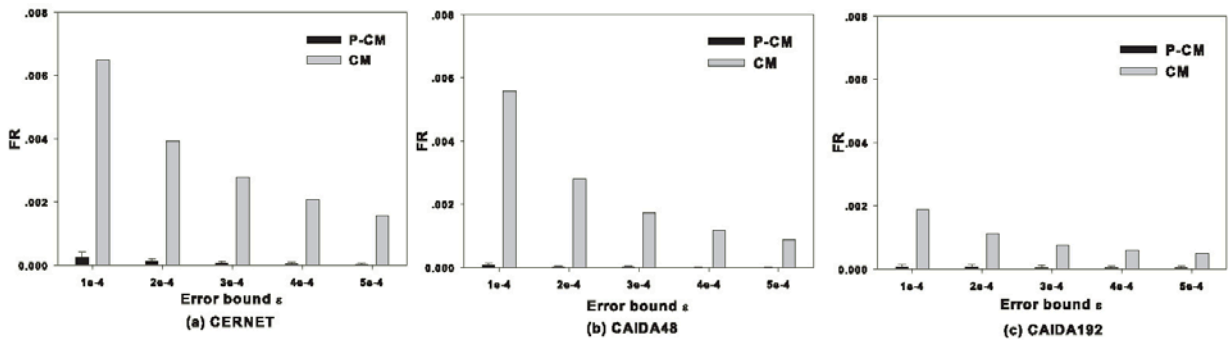

Fig. 3 Failure rate of P-CM and CM

Fig. 2 plots the average relative error with standard deviation of P-CM and CM on three real traffic traces. We can see that the average relative error of P-CM is much higher than that of CM with the

same error bound. This might be because that the estimated frequencies in P-CM are often too much overestimated as a result of the β-delayed update. Fig. 3 plots the failure rate with standard deviation of P-CM and CM on three real traffic traces. Note that the standard deviation of CM is 0. As can be seen, the failure rate of P-CM is much less than that of CM with the same error bound. Fig. 4 plots the speedup with standard deviation of P-CM compared to CM on three real traffic traces. From this figure, it can be seen that the speedup increases almost linearly with the number of sub-threads *m*.

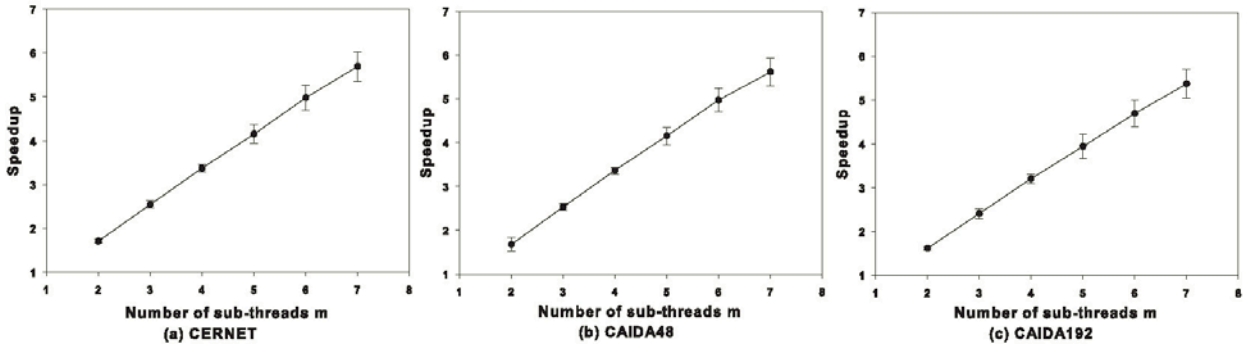

Fig. 4 Speedup of P-CM

Conclusion

In this paper, we consider the problem of parallelizing the famous Count-Min sketch algorithm in the context of multi-core processors. The proposed parallel Count-Min sketch algorithm makes use of the delayed update concept to significantly reduce the merge overhead and achieves near-linear speedup.

Acknowledgements

This work is partially supported by the Tianjin Municipal Science and Technology Commission under Grant No. 13ZCZDGX01098.

References

[1] S. Muthukrishnan, "Data streams: Algorithms and applications" Now Publishers Inc, 2005.

[2] C. Estan, G. Varghese. "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice." ACM TOCS, 2003.

[3] Y. Gao, Z. Li, Y. Chen. "A dos resilient flow-level intrusion detection approach for high-speed networks." IEEE ICDCS, 2006.

[4] B. Krishnamurthy, S. Sen, Y. Zhang, et al. "Sketch-based change detection: methods, evaluation, and applications." ACM IMC, 2003.

[5] Y. Liu, L. Zhang, Y Guan. "Sketch-based streaming PCA algorithm for network-wide traffic anomaly detection." IEEE ICDCS, 2010.

[6] G. Cormode, S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." Journal of Algorithms, 2005.

[7] A. Kumar, M. Sung, JJ. Xu and J. Wang. "Data streaming algorithms for efficient and accurate estimation of flow size distribution" ACM SIGMETRICS Performance Evaluation Review, 2004.