LITERATURE REVIEW: Parallel and Fast Algorithm for Finding Top-K Frequent Elements

Mostafa Hosseini
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
mostafahosseini@cmail.carleton.ca

October 6, 2022

1 Introduction

Since dealing with large amount of data stream has become one of the hot topics recently, using parallel algorithms is very helpful to process a massive data stream, specially when we are facing a time limit. One of the important issue in data stream processing is finding the most frequent items. This topic has many applications in the real world such as Internet traffic analysis for anomaly detection to find heaviest users and high-traffic IPs [4].

Finding proper algorithms for solving this issue is very important considering two main limitations. First is processing time when our algorithm must be processed very quickly in a real time aspect, second is the space limit when we just have a short memory space for working with the algorithm. Therefore, we should consider a tradeoff between space, accuracy, and efficiency.

The main problems that we seek to improve are speed of frequent items estimation with respect to the working space limitation and accuracy of algorithm. For addressing this issue, Count-Min Sketch (CMS) [6] has been selected as a famous approximate algorithm and recently, several researches have been done in order to improve the speed and accuracy of this algorithm. A sketch is a 2-dimensional array that has h hash functions and w cells. Each hash function maps an element uniformly at random to a cell. Also, we need a memory space for storing top-K frequent items. Therefore, there are several ways for implementing parallel processing to improve the performance of the algorithm such as running each sketch in different thread or parallelizing counting the elements and finding the top frequent items at the same time.

This project works on state-of-the-art algorithms based on CMS and uses parallel processing methods for boosting processing speed. First, we review the main related works as a literature review. Next, we explained proposed algorithm in detail, and we will show how parallel processing is used in this project to increase the speed. After that, we evaluate the performance and accuracy of proposed algorithm, and we compare suggested algorithm with a baseline algorithms.

2 Literature Review

There are several methods for data stream summarization, but the most famous are sketche-based and counter-based methods. Counter-based data structures such as Frequent [10], Lossy Counting [12] and Space Saving [3] are designed for finding the top-K frequent items and they maintain a set of counters associated with a subset of words from the data stream it has traversed. Sketches can keep approximate counts for all items , while counter-based approaches maintain approximate counts only for the frequent items. Sketches can support top-K queries with an additional heap or a hierarchical data structure [9]. However, counter-based approaches have slower update time than sketches for the top-K estimation [5], as they do not count the frequency of all items.

Sketches have been widely applied to item frequency estimation in data streams. The most widely used sketch is the Count-Min Sketch [6]. CMS will be discussed in this article as a baseline model. Other famous sketch-based algorithms include Count-sketch [11] and CM-CU sketch [1]. In contrast to the CMS, the Count-Sketch does not require the strict turnstile model, where the final histogram may have positive or negative entries. Also, Count-Sketch suffers from both overestimation and underestimation, Whereas CMS and CM-CU sketch suffer from overestimation only. The CM-CU sketch achieves higher accuracy than CMS and the only difference is that CM-CU only increments the smallest one(s) among all hashed counters. CM-CU does not support deletions, and consequently it has not received as wide acceptance in practice as the CMS.

There are several methods for improving throughput of sketches such as Holistic UDAFs [8] and Frequency-Aware Counting [7] and gSketch [14]. However, they usually use additional data structures and increase the overall storage requirement in compare with CMS, which is often significant with respect to small-space sketch data structures.

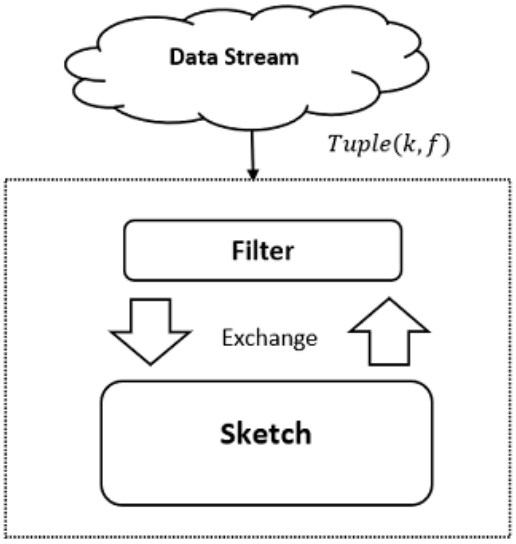

Figure 1: Main structure of Asketch

Augmented sketch framework (Askech) [13] improves both accuracy and throughput considering it has same space as CMS. As it is shown in Figure 1, Asketch is structurally

similar to CMS, but it has a filter for saving the most frequent items (e.g., 32 hot items) and data can be exchanged between the sketch and the filter. The filter size is very smaller than sketch size, so it has higher speed than sketch for saving and updating data. The filter has two variables named new count and old count. New count keeps last frequency of each item and old count keeps the frequency of each item when it was in sketch. First of all, the algorithm checks if the item is in the filter or not. If it is in the filter, it just updates new count and if it is not in filter and filter is not full, it adds the item into the filter and set old count to zero and new count to its item frequency. If filter is full, it updates the sketch and related buckets for this item and send an item to the filter if the sketch has an item with higher frequency than items in the filter. The Filter can run in one core and the sketch can run in another core to improve the speed of finding hot items.

Topkapi [2] uses the advantage of both counter-based and sketch-based methods. CMS has an excellent update time, but it suffers from reducibility which is needed for exploiting available massive data parallelism. On the other side, the popular Frequent algorithm (FA) leads to reducible summaries, but its update costs are significant. Topkapi proposes a fast and parallel algorithm for finding top-K frequent items, which gives the best of both methods. It is reducible and has fast update time similar to CMS. As it is shown in Figure 2, this algorithm inserts a counter to each cell of the sketch in order to store the frequency of the most frequent item. Therefore, it keeps a local heavy hitter for each cell. It has high accuracy, but because of having additional structure for saving the local heavy hitter, the total performance could be slower in compared with ASketch. For speed up, the authors used multi-core processing and the result of each thread will be merged, after all threads finished their process.

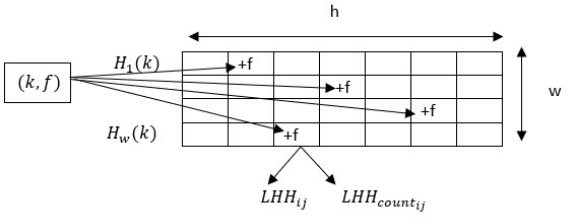

Figure 2: Main structure of Topkapi

Cold Filter [15] proposes a novel meta-framework by filtering small flows at the first stage and running additional algorithms separately on the residual large flows. Cold Filter uses a two-layer filter to prevent mouse flows from entering some data structures (e.g., Space-Saving and the CMS). Although Cold Filter shares some similarity with Asketch it uses entirely different algorithms, and it has complex data structures and cannot be easily implemented in the data-plane.

A recent state-of-the-art method presented the Diamond Sketch [16], which dynamically assigns to each flow a proper number of atom sketches in order to reduce memory footprint. Though it greatly reduces the relative error, it is designed specifically for flows with non-uniform distribution and this algorithm is incapable of working with arbitrary distribution.

All mentioned methods have fixed counter size for recording the frequency of items and maybe in very large data stream, some hot items have more counts than the size of counter. Pyramid sketch [17] framework can automatically enlarge the size of counters according to

the current frequency of the incoming item, and has good accuracy and speed compared with other methods. However, hot items require quite a few memory accesses, and thus insertion speed of the Pyramid sketch in the worst case is poor.

References

[1] H. Daume A. Goyal and G. Cormode. Sketch algorithms for estimating point queries in nlp. *Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, page 1093–1103, 2012.

[2] A. Shrivastava A. Mandal, H. Jiang and V. Sarkar. Topkapi: Parallel and fast sketches for finding top-k frequent elements. *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[3] D. Agrawal A. Metwally and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. *International Conference on Database Theory*, 3363:398–412, 2005.

[4] A. Anbarasu B. Sigurleifsson and K. Kangur. An overview of count-min sketch and its applications. *EasyChair Preprint*, (879), 2019.

[5] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[6] G. Cormode and S. Muthukrishnan. An improved data-stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[7] R. Bordawekar D. Thomas, C. Aggarwal, and P. S. Yu. On efficient query processing of stream counts on the cell processor. *IEEE 25th International Conference on Data Engineering*, 2009.

[8] F. Korn G. Cormode, T. Johnson, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, page 35–46, 2004.

[9] M. Garofalakis G. Cormode, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *IFoundations and Trends in Databases*, 4(1-3):1–294, 2012.

[10] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.

[11] K. Chen M. Charikar and M. Farach-Colton. Finding frequent items in data streams. *International Colloquium on Automata, Languages, and Programming*, 2380:693–703, 2002.

[12] G. Singh Manku and R. Motwani. Approximate frequency counts over data streams. *In Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357, 2002.

[13] A. Khan P. Roy and G. Alonso. Augmented sketch: Faster and more accurate stream processing. *SIGMOD '16: Proceedings of the 2016 International Conference on Management of Data*, page 1449–1463, 2016.

[14] C. C. Aggarwal P. Zhao and M. Wang. gsketch: On query estimation in graph streams. *Proceedings of the VLDB Endowment (PVLDB)*, 5(3):193–204, 2012.

[15] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li. Cold filter: A meta-framework for faster and more accurate stream processing. *Proceedings of the 2018 International Conference on Management of Data*, page 741–756, 2018.

[16] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li. Diamond sketch: Accurate per-flow measurement for big streaming data. *IEEE Transaction Parallel Distribution Systym*, 30(12):2650–2662, 2019.

[17] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li. Pyramid sketch: a sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.