3rd National Conference on Innovative Research Trends in Computer Science and Technology (NCIRCST 2018) ISSN: 2454-4248
Volume: 4 Issue: 3 84 – 88

IOT Based ATM Monitoring

R.Karthikeyan
UG Scholar, I.T
S.A.Engineering College.

M.S.Dhamodharan
UG Scholar, I.T
S.A.Engineering College.

M.Kumaran
Assistant professor, I.T
S.A.Engineering College

Abstract—Distributed data stream mining in a sliding window has emerged recently, due to its applications in many domainsincluding large Telecoms and Internet Service Providers, financial tickers, ATM and credit card operations in banks and transactionsin retail chains. Many of these large-scale applications prohibit monitoring data centrally at a single location due to their massivevolume of the data; therefore, data acquisition, processing, and mining tasks are often distributed to a number of processing nodes,which monitor their local streams and exchange only the summary of data either periodically or on demand. While this offer manyadvantages, distributed stream applications possess significant challenges including problems related to an online analysis of therecent data, communication efficiency and various estimation of various complex queries. There are few existing techniques whichsolve problems related to distributed sliding window data stream; however, those techniques are focused on solving only simpleproblems and require high space, query, and communication cost, which can be a bottleneck for many of these large scaleapplications. In this paper, we propose an efficient query estimation technique by constructing a small sketch of the data stream. Theconstructed sketch uses a deterministic sliding window model and can estimate various complex queries, for both centralized anddistributed applications; including point queries.

Index Terms*—Distributed Query Estimation, Distributed Heavy Hitters, Distributed Data Streams.*

*****

1 Introduction

In many applications data needs to be processed on a nonstop continuous basis, i.e., in the form of a stream. For example, in large Telecoms and Internet Service Providers, detailed usage information (e.g., Call-Detail-Records, SNMP packet-flow data, etc.) needs to be continuously collected and analyzed for billing, capacity planning, and identification of trends and anomalies. A common need for these applications is to extract useful intelligence and actionable rules from high speed multidimensional streams, which can be accomplished through a summary sketch that can rep-resent the statistical properties of data while allowing fast real-time updates and query. Such summary sketch can im-prove scalability and efficiency of various data analytic tasks such as estimating self-join and inner product, quantiles, frequency statistics and heavy hitters. The heavy hitters problem is one of the most studied questions in data streams research, due to its simplicity to state, and its value in many applications. Informally, given a series of items arriving in a stream, the objective is to find those items which occur most frequently.This is useful in many domains including databases (e.g.,)finding attribute values with high frequencies. IP network traffic monitoring (e.g., identifying heaviest bandwidth users), online analytical processing (e.g.,online queries performed in real time monitoring systems),and search engines (e.g., the most frequently searched terms in queries made to an Internet search engine).Due to their massive volume of the data, many of these large-scale application prohibit monitoring data centrally at a single location; therefore, data processing task is distributed to a number of processing nodes, which monitor their local streams and exchange only the summary of processed data either periodically or on demand. While this offer many advantages, distributed streams applications posse significant challenges including problems related to online analysis of recent data, communication efficiency and various complex queries estimation. First, often recent data is more important and emphasized in these applications while historic data is accumulated and archived in the DBMS of a data warehouse where access to it is time-consuming and prohibitively expensive. Second, many of these large-scale distributed monitoring applications impose critical communication efficiency requirements

2 Previous Work

Distributed systems have seen a wide range of development recently [6], [7], [8]. Studies on distributed stream processing focus on communication efficiency for handling various query types, including monitoring of simple distributed aggregates [9], join aggregates [10], distributed quantiles [11], dynamic continuous queries [12] and distributed threshold conditions [13], [14]. However, these approaches are based on full history stream computational model, which do not solve the problems that are specific tosliding window stream computational model, hence cannot be applied to distributed sliding window stream processing. Chakrabati et al. [15] proposed to combine exponential histogram [4] with count-min sketch [16] for computing theentropy of the stream in sliding window. Their algorithmrequires O(12 log1log N) space, and processes each updatein O(1+ log log1+ log log)time. here is a bulk of work[17], [18] on processing data streams in sliding window,however, their focus is on proximating simple centralizedqueries (i.e., heavy hitters) and cannot answer complexRecently ECM-sketches were proposed in [3], which similarto [15], combine exponential histogram with count-min sketchto estimate various queries in the distributed sliding windowstreams. There are several differences be-tween ECM-sketchand ESS-sketch. First, ECM-sketch com-bines exponentialhistogram with sketches like count-min sketches to designdistributed sliding window stream min-ing technique, becausecount-min sketches are trivially com-posable due their matrixlike structures. This work uses Space Saving [5] structure andreplaces its Integer counters by exponential histograms ordeterministic waves to de-sign distributed sliding windowstream mining technique.

Nevertheless, using Space Saving ina distributed setting is non-trivial due to the complexity inmerging data structures containing both keys and counters.We address this issue by developing an approach to solve thecomposability issue ofESS sketches.

3 Preliminaries

In data streams mining, there are two types of compu-tationalmodels; full-history, where the complete data is available forquery estimation; and sliding window, where time decayedpartial data is available for query estima-tion. The main ideabehind the proposed framework is to combine the basicstructures from full-history and sliding window streams to builda composable structure (ESS-sketch) for the centralized aswell as distributed sliding window streams. ESS-sketch offersthe following benefits. a) ESS-sketch allows solving complexproblems in the sliding window model, including sliding windowquery processing (e.g., computing quantiles, maintainingfrequency statisticsand finding heavy hitters), and database operations such asjoins and inner products that are not currently available insliding window model; b) moreover, we have designed theESS-sketch to be composable so that it allows the samecomplex queries in a distributed environment; c) Theproposed distributed ESS-sketch enable accurate merging of distributed sketches which improves on existing work (see Section 2) by offering deterministic accuracy of queryestimation (bounded by) with significantly lower requirementsfor memory O(12 log n), query processing O(1), anddistributed merge complexity O(d2 log n), which is far lessthan existing solutions (see Table 2).In Section 4, we describe the proposed ESS-sketch forcentralized setting, and in Section 5, we provide details ofthe proposed merge algorithm for ESS-sketches to summarizethe distributed sliding window data streams. Next,we describe the two computational models in more detail,focusing on the aspects relevant to our work .

3.1 Full-history Stream Computational Model

In this model, data items arrive in the form of a continuousstream, where all the elements arrived so far are consideredrelevant at any time to answer queries on the data set. Manymining algorithms, database operations, and Internet searchengine query processing systems require efficient ex-ecutionwhich can be difficult to accomplish with a fast data stream.

For instance, Google processes over 40,000 search queriesevery second on average, and a typical router inter-faceprocesses around 1 million packets every second [19].Therefore, it is often acceptable to compute approximateanswers for such problems. Many algorithms are proposed inliterature [5], [16] that can estimate frequencies of items in fullhistory

streams. Space Saving (SS) [5] is a widely usedcounter-based algorithm for estimating item frequencies. It cantrack a subset T of items from a universe by maintaining anapproximate count for each item in the subset. Each tuple in Tcontains three entries, [R; f^(R); R], where R is the record,f^

(R) is the estimated frequency of R, and R is theestimationerror. If the incoming record R from stream S is in T , SSincrements the corresponding counter. Otherwise, the recordwith the smallest counter in T say P) is removed from T andreplaced by R. Also, the counter of R is set to f^(P) + 1, andthe error of R is set to R = f^(P).

3.2 Sliding Window Stream Computational Model

In this model, data items arrive in a form of continuous stream,where only the last n (window size) items are consideredrelevant at any given time. The most recent n items are activedata items, while the rest of the data items are expired and areno longer contributing to any query answers on the data set. Adata item that has been processed cannot be retrieved forfurther processing at a later stage. The available memory forprocessing the stream is limited and often required to besublinear.Hence,algorithms which need to store all theactive data items are considered inadequate in the slidingwindow computational model. The sliding window model wasfirst proposed by Datar et al. [4], where they considered asimple basic counting problem.

4 PROPOSED FRAMEWORK FOR A CENTRALIZEDSETTING

In a centralized setting, it is often required to compute anaggregate of a multidimensional high-speed stream. Thissection describes how the proposed approach can be used tofind a summary of a high-speed stream. We first describe ESSsketchwith its pseudo code and later provide an exam-ple.ESS-sketch supports both count-based and time-based slidingwindows. The core idea of ESS-sketch is a modifiedSS structure [5]. SS structure is designed for full-history datastreams and cannot handle sliding window constraints. ESSsketchaddresses this limitation by replacing the Integercounters of SS structure by sliding window counters W, whereeach W is an exponential histogram or deterministic waves.

Specifically, it associates each record P with a sliding windowcounter that counts the number of occurrences of P within thesliding window, covering the last n arrivals, or the last n timeunits, depending on whether we need count-based or time basedsliding windows. For each P 2 S, letSP = e1e2e3 be a bit stream, where, for any i 1, ei = 1 if Pi= P , and ei = 0 if Pi 6=P . This means that for each

unique record P there is a corresponding bit stream SP of 0'sand 1's constructed using the above rule. For instance, seeFigure 1 to ustream S. The constructed bit stream SP for a recordP has a digit 1 when the corresponding record in the actualstream is P and has a digit 0 otherwise. Once the bit streamsare constructed, we can use window counters W to estimatethe number of 1's in those bit streams (such as SP), which inturn finds the estimated number of occurrences of a record(such as P) in sliding window in the original stream S.Next, we explain how ESS-sketch summarizes the streamof records in a sliding window. ESS-sketch maintains a smalldata structure T , where each entry in T is a tple [R; WR].We denote the number of tuples jT j by m (m = jT j). For eachincoming record R that is present in T , we insert a 1 with thearrival timestamp i of R into the corresponding sliding windowcounter. If R is not present in T , then we need to make roomto insert R by first removing any ex-pired records(s) (i.e.,records that are older than the current window). If there is noexpired record, then the record with the lowest count isreplaced with R and 1 is inserted into its sliding windowcounter. The insertion of 1 ensures to add 1

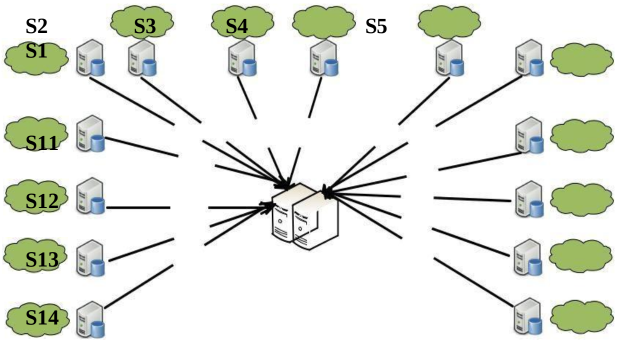
to the count ofrecord R, similar to the case of Space Saving with Integercounters. Algorithm 1 outlines the steps of theESS-sketch. ESS-sketch can also handle a generic casewhere each record in the stream is a pair (R; c). For eachincoming pair (R; c) from stream S, if R is in T , ESS-sketchinserts c number of 1's into WR , all with same timestamp i,and inserts 0 to all other W in T . Both exponential histogram and deterministic waves require the timestamp tobe non-decreasing. Therefore, inserting any number of 1'sinto W with same timestamp (as long as the timestamp isnon-decreasing) does not affect the working principle of W .Although ESS-sketch can handle streams where itemshave an arbitrary count, for the sake of simplicity, andclarity, from here onward we will consider f(Ri) to be 1 (i.e.,
streams with unitary updates). We will consider the case inwhich the stream S is simply a sequence of records Riwithout an associated frequency f(Ri), hence, n N.Before providing further detail about ESS-sketch, wefirst describe how exponential histogram or deterministicwaves can be used to estimate the number of occurrencesof an element in the stream S in a sliding window.

5 Proposed Framework For A Distributed Setting

In a distributed setting, it may be required to compute aggregates on the union of the data in all streams rather than just any individual stream. For example, in a network monitoring problem, packets can enter or leave the network at multiple locations, and each of this location is monitored separately, using multiple distributed sites (see Figure 2 left). The network monitoring team might be interested in querying the aggregates on the union of the data in all streams. One simple solution to this problem would be to send all streams directly to a single site (summarizer or the sink) which can then summarize the entire stream received.

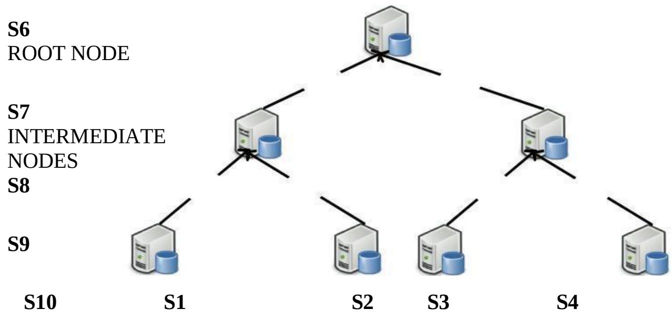
5.1 Merging Window Counters

In this section, we explain the merging algorithm for the time- based sliding window exponential histograms. Al-though exponential histogram supports both time-based and count- based sliding windows, however, we are only able to merge time-based exponential histograms, and the same is true for the deterministic waves [3]. Consider a set of d exponential histograms EH1, EH2, , EHd, each of which represents a summary of a time-based sliding window of L last n time units (minute, hourly, etc.). Let, EHL = EH1 EH2 L L EHd denote the merge operation of the d ex-ponential histograms. Recall that an exponential histogram consists of up to a maximum of k = O(1 log n) buckets (see section 3.2). Let a bucket i = 1; 2; 3; ; k; of an exponential histogram j = 1; 2; 3; ; d; be represented by ij. The buckets are numbered from right to left, such that the most recent bucket is 1. Let, the number of 1's (the size of the bucket) be denoted by j ijj.



5.3 Correctness and Complexity of ESSL Sketch

This section explains some theoretical results of the merge algorithm. First, we provide proof of the correctness of the algorithm using error bound in estimation and then provide update cost of the merge algorithm.
Lemma 10. Let NL be the sum of the frequencies of the records in the union stream SL. Let N1; N2; ; Nd be the sum of the frequencies of the records in the respective distributed summaries ESS1, ESS2, , ESSd, for the d distributed data streams S1; S2; ; Sd. Let be the error parameter and n be the length of the time- based sliding window for each of the distributed summaries. Then, the estimation error by the merged summary ESSL that corresponds to SL is bounded by NL. For proof see appendix. Theorem 11. The worst-case time to merge d summaries, each of size m, is O(dkm), where k is the maximum number of buckets in a single window counter. For proof see appendix.The worst-case time required by the merge algorithm can also be written as O(dkm) = O(d2 log n), where k = 1 log n and m = 1 . Theorem 12. The communication cost or amount of volume required to transfer over a network for the merge algorithm to merge d distributed summaries is O(d2)



Theorem 12 can simply be followed from the fact that the size of a single distributed summary is 12 log n, and merging d such summaries requires O(d2 log n) volume to be transferred over the network**.**

5.4 Query Estimation in a Distributed Setting

In this section, we explain how the merged summary ESSL supports point queries, range queries and heavy hitter queries in a distributed settings. Note that the following analyses are valid for time-based sliding window models only.

5.4.1 Distributed Point and Range Queries

In the case of point and range queries in a distributed settings, an ideal case to save the communication cost is to query each ESS-sketch and then accumulate the results from all the distributed instances of ESS-sketch. For point and range queries, there is no need for fetching the entire summary from each site and merging them at a central site. A point query (respect range query) is issued by the querying site using an identifier R (respect (R; r)) over all the distributed sites. The distributed sites then return their local estimates (an integer value) corresponding to the query in the same way as explained in centralized settings (see Section 4.2 for estimating point and range queries). The querying site then sums all the estimates to find the answer to the queries corresponding to the union stream SL. The point query is then answered with the following estimation guarantees.

Theorem 13. Let f(R), and f ^ (R) be the true and estimated frequencies of R in sliding window of size n in the union stream SL. Then the estimation error for the point query is bounded by

f(R) f ^ (R) f(R) + NL.

6 Implementation And Evaluation

We have implemented all our proposed algorithms using Java (v 1.8). The ECM sketches were also implemented in Java and obtained from the authors [3]. For the testing, we have used an Intel Core i7 machine with a 3.4GHz processor, 16 GB RAM, and 64 bit Windows operating system installed on it. The data structures used in all the algorithms imple-mented and acquired are based on hashing techniques; it requires one hashing operation to lookup a particular item in the data structure. Datasets: In our experiments, we have used real Internet traffic traces datasets [23] that are openly available from the WAND Network Research Group at the University of Waikato, New Zealand. Each of these datasets contains 30minutes trace of network traffic data in tcpdump1 format. Wireshark2 was used to read the tcpdump format data for extracting source IP addresses.

Experimental Setup: In real-word applications, it is difficult to predict the maximum number of items that will arrive in a sliding window; therefore, these values are typically decided by analyzing a small stream sample. The exponential histogram and deterministic waves window counters were initialized with an upper bound of 1000 events per milliseconds. Exponential histograms based al-gorithms (e.g., ESS-sketch) are better in this regards as they do not require this information at initialization time. The frequency of a transaction is considered 1, as it is associated with the packet. Note, the frequency of a single transaction need not necessarily be 1; it can be any arbitrary number including the number of bytes in the packet, as it does not affect the algorithms presented. Thus, both ESS and ECM sketches would estimate the number of packets sent by a machine with a unique IP address. We considered two variants of the proposed ESS-sketch, ESS(EH) and ESS(DW), distinguished by the type of window counter used; and compared them against variants of the ECM-sketches, which we call ECM(EH) and ECM(DW).

Evaluation Criteria: The objective is to evaluate ESS and ECM sketches with respect to efficiency and effectiveness, scalability and suitability for centralized and distributed settings. To compare the efficiency and effectiveness of ESS and ECM sketches, we have considered different factors including memory usage, execution and query time and quality of the output in both centralized and distributed settings. The memory usage is compared using the maxi-mum number of bytes used by the sketches during runtime in the centralized settings or transferred over the network in distributed settings. The quality of output is compared using estimation error in different queries.

Summary of the Results: Detail experimental analysis of the ESS-sketch shows that in both centralized and dis-tributed settings, overall its memory has sub-linear growth with respect to both data size and time, which is a significant improvement for streaming data. The variants of ESS and ECM sketches that are based on exponential histograms are better due to lower space usage, and identical estima-tion error. In centralized setting, comparing ESS and ECM.

6.1 Evaluation in a Centralized Setting

In this set of experiments we explain results of the central-ized settings, where a single machine observes the whole stream at a central point using ESS and ECM sketches. We have evaluated the sketches for various parameter settings, and illustrated the results against those settings. For ECM sketches, the parameter was set to 1, which corresponds to a probability of 99% accurate results. **Memory Usage:** In this section we explain the memory usage of ESS-sketches. Memory is one of the the critical resources for streaming applications, consequently, better sketches have lower memory usage. Although, we have given theoretical memory usage in terms of maximum num-ber of buckets that ESS and ECM sketches can maintain (see Table 2), we are providing results of practical memory usage in terms of maximum actual bytes used. Figures 3a and 3b compares the memory usage of ESS and ECM for varying within the range of [0.001 to 0.017]. The difference be-tween variants of ESS-sketch, that is, ESS(EH) and ESS(DW) is slight. Particularly, ESS(EH) has slightly lower memory usage than the ESS(DW). This can also be observed for the variants of ECM sketch. The memory usage of both ESS and ECM sketches decreases with larger values of . The differ-ence between the memory usage for ESS and ECM sketches is substantial. ECM sketch uses higher memory compared to ESS-sketch. For instance, for = 0:003 the proposed ESS-sketch uses 8890512 bytes (or 8.5 MB) of memory while ECM sketch uses 64962149 bytes (or 62 MB) of memory. The average memory usage is illustrated in Figure 3c, which shows that on average, the memory usage of ESS-sketch is about seven times less than the memory usage of ECM-sketches. Such a significant memory reduction is clearly an important advantage of ESS-sketches over ECM-sketches.

Update and Query Performance: In this section, we ex-plain the updates and queries performance of ESS- sketches. Like memory, execution time is also one of the critical resources for streaming applications. Theoretical update and query complexities are given in Table 2. Here we provide the time required for the update and query observed in our experimental settings for = 0:001.

Thus, we note that ESS-sketches have competitive up- date performance and have substantially better query per- formance compared to ECM-sketches. This can be very useful for the applications that have very fast query require- ments.

Estimation Error: In this section we explain the aver- age observed error in estimating the frequencies of items monitored in a centralized stream. In Section 4.2 we have provided theoretical analysis to demonstrate the theoret-ical upper bounds on estimation error, here we provide experimental estimation error for point, range and self-join queries.

6.2 Evaluation in a Distributed Setting

We have already studied the effect of on memory usage, update time and query time for both ESS and ECM sketch variants in a centralized setting. In this section, we evalu-ate ESS and ECM sketches in a distributed setting. Since variants of ESS and ECM sketches based on exponential his-tograms offer better performance, therefore, in a distributed setting we considered only exponential histograms basedvariants of ESS and ECM sketches.

7 Conclusions

The paper introduced efficient ESS-sketches for query es- timation over sliding window data streams, both in cen- tralized and distributed settings. Theoretical analysis is provided to highlight efficiency and effectiveness of ESS- sketches. Detailed experimental analysis of the ESS-sketch reveals that, for both centralized and distributed settings, its overall memory growth is sub-linear with respect to data size and length of sliding window, which is a significant improvement over existing techniques for streaming data. Comparative evaluation of ESS and ECM sketches in a cen- tralized setting shows that ESS-sketches offer many advan- tages over ECM-sketches. For example, the proposed ESS- sketches has significantly lower memory (about six times less) and query time (about eight times less) requirements; ESS-sketches provide deterministic error guarantees and has about a similar update cost requirement both in theory and in practice. In distributed setting, the communication cost and merge time increases with the increase in network size (number of network nodes) for both ESS and ECM sketches; however, the increase in both cases (i.e., communication cost and merge time) for ECM sketch is much higher than the proposed ESS-sketch. In summary, although both sketches provide the same average estimation error, ESS-sketch outperforms ECM-sketch in terms of communication cost of distributed queries, e.g., transferred volume and merge time.

References

[1] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, “Sketch- based querying of distributed sliding-window data streams,” VLDB Endowment, vol. 5, no. 10, pp. 992–1003, 2012.

[2] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting, “Continuous monitoring of distributed data streams over a time-based sliding window,” Algorithmica, vol. 62, no. 3-4, pp. 1088–1111, 2012.

[3] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, “Sketching distributed sliding-window data streams,” The VLDB Journal, pp. 1–24, 2015.

[4] M. Datar, A. Gionis, P. Indyk, and R. Motwani, “Maintaining stream statistics over sliding windows,” SIAM Journal on Comput- ing, vol. 31, no. 6, pp. 1794–1813, 2002.

[5] A. Metwally, D. Agrawal, and A. El Abbadi, “Efficient computa-tion of frequent and top-k elements in data streams,” in Database Theory-ICDT 2005. Springer, 2005, pp. 398–412.

[6] B. P. Rimal and M. Maier, “Workflow scheduling in multi-tenant cloud computing environments,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 1, pp. 290–304, 2017.

[7] C. Chen, X. Zhu, P. Shen, J. Hu, S. Guo, Z. Tari, and A. Y. Zomaya, “An efficient privacy-preserving ranked keyword search method,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 4, pp. 951–963, 2016.

[8] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, “Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption,” IEEE transactions on parallel and distributed systems, vol. 24, no. 1, pp. 131–143, 2013.

[9] C. Olston, J. Jiang, and J. Widom, “Adaptive filters for continuous queries over distributed data streams,” in Pro. ACM SIGMOD Mgmnt of data. ACM, 2003, pp. 563–574.

[10] G. Cormode and M. Garofalakis, “Approximate continuous query- ing over distributed streams,” (TODS), vol. 33, no. 2, p. 9, 2008.