PeriodicSketch: Finding Periodic Items in Data Streams

Zhuochen Fan[†], Yinda Zhang[†], Tong Yang[†‡], Mingyi Yan[†], Gang Wen[†], Yuhan Wu[†], Hongze Li[†] and Bin Cui[†]
[†] School of Computer Science, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing, China [‡] Peng Cheng Laboratory, Shenzhen, China
{fanzc, yanmingyi, jnwengang, yuhan.wu, pku_lhz, bin.cui}@pku.edu.cn, {hgdkgszyd, yangtongemail}@gmail.com

***Abstract*—In this paper, we study periodic items in *data streams*, which refer to those items arriving with a fixed interval. All existing works involving mining periodic patterns does not fit for data stream scenarios. To find periodic items in real time, we propose a novel sketch, PeriodicSketch, aiming to accurately record top-K periodic items. To the best of our knowledge, this is the first work to find periodic items in *data streams*. Any interval may occur many times, and we use frequency to denote the number of an interval occurred. To pick out periodic items with high frequency, we propose a key technique called Guaranteed Soft Uniform (GSU) replacement strategy. Our theoretical proofs show that when replacement is successful, it is more likely that the new item has a higher frequency than the current smallest frequency; and GSU can ensure that our items in the sketch will approach the true periodic items closer and closer. And as soon as we get all the periodic items, the state would not change worse with high probability. We conduct extensive experiments, and the experimental results show that the Average Absolute Error (AAE) of our sketch using 1/10 memory is around 737 times (up to 2019 times) lower than the baseline solution. Finally, we provide a concrete case: Cache prefetch, which proves that PeriodicSketch can significantly improve the Cache hit ratio. All related codes of PeriodicSketch are open-sourced and available at GitHub [1].**

I. Introduction

A. Background and Motivations

Nowadays, data stream processing is an important task where we need to extract the needed information from a large volume of data with high speed in one pass. However, due to the contrast between the large volume of data and limited memory we have, we cannot accurately extract the information while keeping up with the high speed of data streams. Therefore, many probabilistic data structures (called sketches in this paper) [2]–[5] have been widely accepted because they can work in a small memory and catch up with the high speed with small errors. These sketches work on common tasks, such as finding frequent items [2], [6], [7], persistent items [8]–[10], and super-spreaders [5], [11], [12].

In this paper, we work on finding periodic items in the data stream model. To the best of our knowledge, this is the first work to find periodic items in the data stream model. In this paper, periodic items refer to items that arrive with a fixed interval. Finding periodic items is to report top-K periodic items and their intervals. For example, an item arrives at 8:00 every day, and lasts for one month. We consider it as a periodic item. Moreover, its period/interval is a day, and the interval's frequency is 30. The formal definition of this problem is provided in Section III. Finding periodic items is important, and below we show four use cases on finding periodic item in data streams.

Case 1 - Cache: In the Cache scenario [13], the requests of items form a stream, and some requests may arrive periodically. If we can pick out such periodic requests and measure its period, we can significantly improve the performance of *prefetching* [14]: we can prefetch/load the item to the Cache just before its next arrival time, and delete it afterwards. In this way, we can not only significantly increase the Cache hit ratio for the periodic requests, but also make room for other items.

Case 2 - APT detection: Recently, frequent Advanced Persistent Threats (APT) [15] has received widespread attention due to its great harm, strong destruction, long duration, and difficulty in detection and early warning. Its main feature is to use advanced information technologies to carry out targeted, continuous and concealed attacks on specific target systems within a long time span. In fact, apart from low frequency, APT generally has periodicity [16]. The implementer of the APT refers to the working hours of the target system and major festivals when implementing the attack, so it is likely to be periodic. APT may periodically establish TCP connections and perform DNS queries. In addition, when controlling the hacked server to send back the stolen data, the data sending behavior also show periodicity. Therefore, we can first find periodic items, and then further ascertain whether they are APT by existing detection methods.

Case 3 - Network traffic prediction and classification: Nowadays, machine learning (ML) is playing an increasingly important role in data stream mining [17]. As an important property, the periodic items in the data stream can make the ML model perform prediction and even classification tasks well. For example, for network traffic prediction and classification, item[1] periodicity is a significant feature, and periodic items often worthy of being given a higher weight to obtain more accurate output results. Moreover, they can reduce the complexity of the ML model and improve its performance.

Case 4 - Periodic transactions in finance and customer purchases: First, periodic transactions by the same customer

Corresponding author: Tong Yang (yangtongemail@gmail.com).

[1]Here, the item ID refers to the 5-tuple of the packet which consists of source IP, destination IP, source port, destination port, and protocol.

in financial transactions stream may be suspected of illegal market manipulation [18]. Therefore, we can quickly find suspicious customers who may be laundering money by real-time detecting periodic transaction items. Second, when users click or purchase different products online [19], a transaction stream is generated. In this case, users' periodic clicking on a category of commodities can be regarded as a periodic item. Therefore, mining such periodic items helps to understand the users' buying behaviors and recommend relevant products or deliver advertisements to them.

To find periodic items in data streams, a baseline solution consists of many Bloom filters [20] and a Space-Saving [21]. These Bloom filters are used to record the historic appearances of items, and the Space-Saving is used to record top-K frequent items with the same intervals. Specifically, given a data stream, the item interval ranges from 1 to 10000 seconds. It builds 10000 sketches: $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{10000}$. $\mathcal{S}_i$ is used to record and report the top-K frequent items with interval of i seconds. At the end of the data stream, we can traverse all sketches to update the top-K periodic items in the Space-Saving. Unfortunately, this solution is time- and space- consuming. What is worse, this solution cannot report those periodic items with other non-predefined intervals, such as 1.5, 1.6, 2.7 seconds. More details are provided in Section VI-A. We aim to replace these two algorithms to achieve better performance.

B. Our Solution

Although there are some works on mining periodic patterns, their definitions are different from ours. None of them can be used to find periodic items in data stream model, because their insertion for each item in the data stream cannot be finished in $O(1)$ time, see Section II-D for details. Therefore, this paper proposes a novel sketch for finding periodic items in data stream model for the first time, named PeriodicSketch. PeriodicSketch is compact. It only requires 50KB of memory consumption when working on 30M items. PeriodicSketch is fast. The throughput of PeriodicSketch is 3.1 times faster than the baseline solution in average. PeriodicSketch is accurate. Compared with the baseline solution, PeriodicSketch improves the Precision Rate (PR) by 77.3% and the Recall Rate (RR) by 74.4%, respectively, and decreases the Average Absolute Error (AAE) up to 2019 times (737 times in average) in finding periodic items.

PeriodicSketch includes two parts: a Cover-Min sketch and a GSU sketch to replace the above Bloom filters and Space-Saving, respectively. We first use the Cover-Min sketch to record and report the interval of the incoming item, then use the GSU sketch to record and report periodic items. The Cover-Min sketch is used to report the interval between now and the last arrival time for the incoming item, which is our first key technique. The details are provided in Section IV-B. It combines with the second part can significantly reduce the time and space overhead of the above baseline solution: we only need to build two sketches instead of 10000 sketches.

The second part of PeriodicSketch, *i.e.*, GSU sketch, is used to catch the potential periodic items. By appropriate modifications (see Section IV-C), this problem can be transformed into the hot issue – finding frequent elements/items. The most popular prior art of finding frequent elements is Space-Saving [21] and Unbiased Space-Saving (SIGMOD'18) [22]. They use a min-heap to keep the potential frequent elements. Every time a new element comes, they increment the frequency of least frequent element recorded ($Element\_min$). Most elements are infrequent, and every infrequent element increment the frequency of $Element\_min$. This leads to serious over-estimation errors of the elements' frequency in the min-heap. To address this problem, our GSU selectively increments the frequency of $Element\_min$ with an adaptive probability $\mathcal{P}$, which is often small (*e.g.*, < 0.01). In this way, we can minimize the influence of infrequent elements, and thus achieve much higher accuracy. In addition, GSU can also treat elements that arrive early/late fairly/uniformly. More details are provided in Section IV-D.

We conduct rigorous mathematical derivation in Section V and extensive experiments in Section VI, and the results show that PeriodicSketch has obvious advantages over comparison algorithm in the accuracy of finding periodic elements/items.

Key Contributions:

- We propose and define a problem named finding periodic items in *data streams*, which are important in many applications but not studied before.
- We propose a novel sketch named PeriodicSketch to accurately find periodic items using small memory with high speed.
- We get to concise theoretical results by strict derivation.
- We conduct extensive experiments, and the results show that our sketch significantly outperforms the baseline solution. Particularly, the AAE of PeriodicSketch using 1/10 memory is around 737 times (up to 2019 times) lower than the baseline solution, and the throughput of PeriodicSketch is about 3.1 times faster than the baseline solution.

II. Related Work

To the best of our knowledge, no prior work can directly deal with finding periodic items in data streams. In this section, we first present the CM sketch, which is the inspiration of our algorithm. Next, we present the similar problem of finding frequent items and typical solutions. Then, we briefly introduce Bloom filter and Space-Saving, because they are the components of the baseline algorithm we proposed in Section VI-A. Finally, we give some existing work on mining periodic pattern. Note that we give this part of the related work only because they have some similarity with our work. In fact, the problems raised and solved are different.

A. CM Sketch

CM sketch [2] is composed of d arrays, each array $A_i\,(1 \leq i \leq d)$ has w counters, and A_i is associated with a hash function $h_i(.)$. For the CM sketch, when the coming item e is inserted, it increments the d hashed counters $A_i[h_i(e)]$ by one. It is easier to delete, just directly decrements the mapped counter by 1, which is the reverse process of insertion.

For querying the item e, CM sketch reports the minimum one among the d hashed counters. Some successors include sketches of CU [23], Count [24], CSM [25], CMM [26].

B. Finding Frequent Items

The task is to find items with large frequencies. Frequency, $i.e.$, the number of appearances of an item. To find frequent items, two types of solutions exist. The first type, `record all`, records the frequencies of all items. Typical algorithms are made of a sketch (*e.g.*, sketches of CM [2], CU [23], and Count [24]) plus a min-heap, and ASketch [27]. Recording the frequencies of cold items is unnecessary. The second type only records hot items: the information of items with large frequencies. Typical algorithms are `Space-Saving` [21], Unbiased Space-Saving [22], Lossy Counting [6], and Cold filter [28].

C. Bloom filter & Space-Saving

A Bloom filter [20] is an array of m bits associated with k independent hash functions, which is set to 0 at the beginning. Its k mapped bits are to 1 for each incoming item. For a membership query, $i.e.$, querying whether an item occurs in the data stream, the Bloom filter checks whether all its k mapped bits are 1. Space-Saving [21] designs a data structure named Stream-Summary to record and keep top-K frequent items. When the Stream-Summary is full and a new item that it has not recorded arrives, the item in the Stream-Summary with the least frequency will be replaced by the new item. These two algorithms are the components of our baseline scheme, see Section VI-A for details.

D. Mining Periodic Pattern

There are indeed some algorithms for mining periodic patterns in time-series data [29]–[34], but their `(1) Problem Definitions` and `(2) Application Scenarios` are quite different from ours. In fact, the difference in (2) is more obvious, because `only` our work meets the two requirements (2-REQ for short) of processing data stream model: 1) The processing of each item is one-pass; 2) The processing of each item is fast enough to catch up with the high-speed data stream, and the processing time is $O(1)$ complexity. Next, we will introduce the distinctions between PeriodicSketch and the literature in more detail based on (1) and (2).

TiCom [32] addresses the problem of mining periodicity in interweaved, noisy, and incomplete sequence data, while its time complexity reaches $O(n^2)$. RobustPeriod [34], SAZED [33], STAGGER [31] and others [35], [36] have similar problem definitions to TiCom. Among them, SAZED can only detect single period, while STAGGER and [36] can only process fixed-length data fragment. Some [33], [34], [36] of them have a time complexity of $O(n \log n)$, while others [31], [35] are worse. Compared with the previous ones, [37] has no time dimension and defines a counter-based matching problem with slightly better time complexity. There is another type of existing works [29], [30] whose ultimate goal is to mine regular patterns in data streams. Since they define a problem of finding special subsets/supersets and calculating the difference among the sets, their time complexity is much worse than $O(1)$. In summary, the above algorithms obviously cannot satisfy 2-REQ, which are different from our application and focus. Therefore, we cannot borrow the ideas of them to address the problem of periodic items in data streams.

III. Problem Statement

Data stream model: Given a data stream $\mathcal{S} = (e_1, e_2, e_3, \dots, e_i, \dots)$, given an item e, let t_i be the i^{th} appearance of e. Then e has many intervals, where interval V_i is $t_{i+1} - t_i$.

Periodic Items: Generally, for e, if one of its interval occurs many times, we consider it as a periodic item. Formally, we characterize periodic items from two aspects: interval and frequency ($i.e.$, the number of intervals occurred, denoted as f). Specifically, for an interval V, the frequency is defined as the number of arriving intervals that fall in the range $[V - \mathit{\Delta T}, V + \mathit{\Delta T}]$, where $\mathit{\Delta T}$ refers to the threshold of allowable error. This $\mathit{\Delta T}$ can be set to a fixed value, or it can be set as a ratio of the interval (*e.g.*, if the ratio is 5%, then $\mathit{\Delta T} = 0.05V$). *Users can flexibly set the $\mathit{\Delta T}$ according to their own needs and application scenarios.*

Finding Top-K Periodic Items: Given a data stream, the problem is to find K periodic items whose frequency is the K largest. Note that one item could have several different intervals, and thus could be reported more than once.

Example: We present an example to make our problem definition clearer. Given a data stream $\mathcal{S} = (a, b, a, b, d, d, a, c, a, c, e, e, a, b, a, b)$, suppose they arrives at the constant speed and the time interval between them is $V = 2$, and suppose the threshold $\mathit{\Delta T} = 0$ for convenience. Then we should obtain the periodic items as follows. Top 1: $\langle a, 2\rangle$ occurs 3 times. Top 2: $\langle a, 4\rangle$ occurs twice. Top 2: $\langle b, 2\rangle$ occurs twice.

IV. The PeriodicSketch

In this section, we show our solution with two key techniques: Cover-Min and Guaranteed Soft Uniform. We provide the symbols frequently used and their meanings in Table I.

TABLE I: Symbols frequently used in this paper.

Notation	Meaning
$\mathcal{S}$	a data stream
e	a distinct item in $\mathcal{S}$
S_V	Cover-Min sketch of PeriodicSketch
$S[i]$	i^{th} bucket of the GSU sketch
t	timestamp of the item
V	time interval
$\mathit{\Delta T}$	the threshold of allowable error
f	the interval's frequency
$\mathcal{P}$	the probability of replacing the item/element (GSU)
$h_d(.)$	d^{th} hash function of the Cover-Min sketch
$h(.)$	hash function of the GSU sketch
E	an element formed by combining e and its interval V
L	the least frequent element in the mapped bucket

A. Overview

As shown in Figure 1, PeriodicSketch includes two parts: a Cover-Min sketch and a GSU sketch. Our first key technique, the Cover-Min sketch, is used to record and report the interval for each incoming item, while the GSU sketch is used to keep candidates of top-K periodic items. Given an incoming item e and its timestamp t, we first insert them into the Cover-Min sketch, and get the time interval V. Then we try to insert item e and its interval V into the GSU sketch. Note that before this, we first combine the ID of item e and its interval V to form an `element` $E = \langle e, V\rangle$, and then insert this element into the GSU sketch. The same item ID with different intervals will be treated as different elements. In the GSU sketch, we propose another key technique called Guaranteed Soft Uniform (GSU) replacement strategy to capture and keep potential periodic elements with a higher probability. Finally, to find top-K periodic elements, we just traverse the GSU sketch, and report top-K elements according to the K largest frequencies.

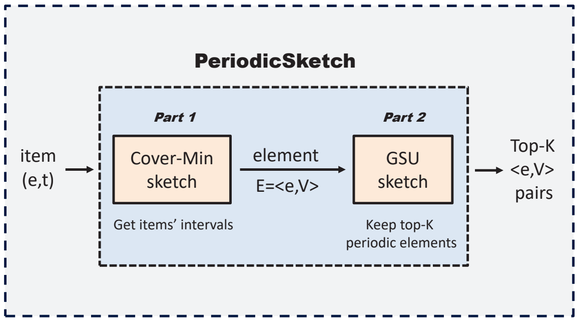


Fig. 1: The workflow of the PeriodicSketch.

B. The Cover-Min Sketch

Data Structure (Figure 2): In order to record and report the time interval of item e, we propose a novel sketch called the Cover-Min sketch, inspired by the well-known CM sketch [2] (mentioned in Section II-A). A Cover-Min sketch S_V has $d \times w$ buckets, and d pairwise independent hash functions $h_1(.), h_2(.), h_3(.), \dots, h_d(.)$. Each bucket has two cells, which record ID e and timestamp t, respectively. For any time interval V, it could occur/recur many times.

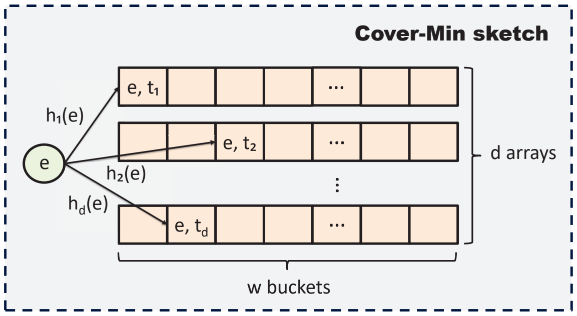


Fig. 2: The data structure of Cover-Min sketch.

Insertion (Algorithm 1): When inputting a coming item e and its timestamp t, it is mapped to one bucket in each of the d arrays by computing the associated d hash functions. Next, the Cover-Min sketch covers/rewrites the timestamp in each corresponding bucket to the current time.

Report (Algorithm 1): For the current item e and its timestamp t, we find the d buckets and extract the d timestamps inside by calculating the associated d hash functions. We use the current timestamp t minus [the smallest/earliest timestamp among the above d timestamps (written as min_t)] to obtain the time interval V, *i.e.*, $V = t - min\_t$, and report it with e.

Algorithm 1: Insertion and report of a coming item e.

```
    Input: input a coming item $e$ with the timestamp $t$
    Output: output the time interval $V$ of $e$
1 $min\_t \leftarrow infinity$;
2 foreach $i$ $(0 \le i \le d-1)$ do
3     $min\_t \leftarrow min(min\_t, S_V[i][h_i(e)])$;
4     $S_V[i][h_i(e)] = t$;
5 return $t - min\_t$;
```

C. Modification

The union of ID and interval: As shown in Figure 1, after obtaining the time interval V of item e through the Cover-Min sketch, we combine the ID of item e and its time interval V to form an `element` $E = \langle e, V\rangle$, and then insert this element into the GSU sketch. In this way, the same item ID with different intervals will be treated as different elements.

Example: For the same item e and its different time interval V_1 and V_2, we treat $E_1 = \langle e, V_1\rangle$ and $E_2 = \langle e, V_2\rangle$ as different elements.

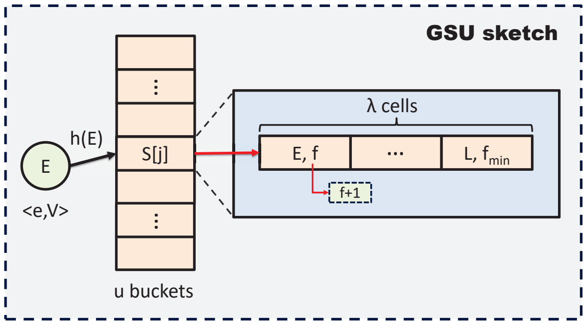


Fig. 3: The data structure of GSU sketch. We assume that $\langle e, V\rangle$ is hashed into $S[j]$, and regard $\langle e, V\rangle$ as an element E for convenience.

D. The GSU Sketch

Data Structure (Figure 3): We propose the GSU sketch to keep and report the top-K periodic elements. The data structure of the GSU sketch can be regarded as a hash table. It consists of u buckets $S[1 \dots u]$, and is associated with a hash function $h(.)$. Each bucket has $\lambda (\lambda > 0)$ cells. Each cell stores an element $E = \langle e, V\rangle$ and its frequency f, and the `frequency` refers to the number of occurrences of the time interval V. Therefore, each cell actually stores the 3-tuple information of the original item e: $\langle e, V, f\rangle$. In the following, we use `English capital letters` to represent elements.

Insertion (Algorithm 2): For an incoming item e, we first query the Cover-Min sketch to obtain its time interval V, and combine the ID of item e and its time interval V to form the element $E = \langle e, V\rangle$. As shown in Figure 3, then we insert E into one of the bucket $S[j]\,(0 \le j \le u)$ through the hash function $h(.)$. There are two cases for insertion:

Case 1: E is not in the bucket $S[j]$. This can be divided into the following two sub-cases: 1) If $S[j]$ is not full, we insert E into an empty cell of $S[j]$, and set the frequency to 1; 2) If $S[j]$ is full, we try to replace the least frequent element L in $S[j]$ with E through the Guaranteed Soft Uniform replacement strategy (GSU for short, explain later). We propose the GSU to probabilistically select the elements that have high probability to be periodic elements.

Case 2: There is already an element E in a cell of $S[j]$. In this case, we increment the frequency f by one.

Report: In order to report periodic elements, we directly traverse the buckets of the GSU sketch S and return the elements (*i.e.*, $\langle item's\, ID, interval\rangle$ pairs) with the top-K largest frequencies.

Algorithm 2: Insertion process of GSU Sketch.

```
Input: An element $E$
1  $random() \in [0, 1]$
2  if $E \in S[h(E)]$ then
3      $f(E) + +$;
4  else
5      if $S[h(E)]$ has empty cells then
6          $S[h(E)].insert(E)$;
7      else
8          if $random() \leqslant \frac{1}{2*f_{min}-t_{fail}+1}$ then
9              $L \leftarrow E$;
10             $f_{min} \leftarrow f_{min} + \frac{t_{fail}}{f_{min}}$;
11             $t_{fail} \leftarrow 0$;
12         else
13             $t_{fail} + +$;
```

Guaranteed Soft Uniform Replacement (GSU): The GSU technique is one of the key novelties of this paper. The workflow of GSU is as follows. Suppose that the smallest cell of $S[j]$ stores element L with frequency f_{min}. Given an incoming element E which is not in the cell, we replace L with incoming element E with a probability

$$\mathcal{P} = \frac{1}{2 * f_{min} - t_{fail} + 1}$$

where t_{fail} is the number of replacement failures. If L is successfully replaced by E, indicating that the frequency of E is likely to be larger than L, we increment the frequency of L from f_{min} to $f_{min} + \lfloor t_{fail}/f_{min} \rfloor$ and set t_{fail} to 0. Otherwise, t_{fail} is incremented by 1. For convenience, $f_{min} + \lfloor t_{fail}/f_{min} \rfloor$ is abbreviated to $f_{min} + t_{fail}/f_{min}$ in this paper.

Designing the Expression of $\mathcal{P}$: We carefully design the expression of $\mathcal{P}$ to meet the following five properties. 1) To successfully replace the original element, the value of t_{fail} should reach an expectation of f_{min}. 2) The larger f_{min} is, the less likely it is to be replaced. 3) The more replacement failures there are, the more likely the replacement will happen. 4) When the number of replacement failures reaches $2 * f_{min}$, the probability $\mathcal{P}$ increases to 1. This can avoid too many replacement failures. 5) When the replacement happens when number of replacement failures is small, t_{fail}/f_{min} = 0, we do not increase the smallest frequency. When the replacement happens when number of replacement failures is large, t_{fail}/f_{min} = 2, we increase the smallest frequency by 2. In other cases, we increase the smallest frequency by 0 or 1.

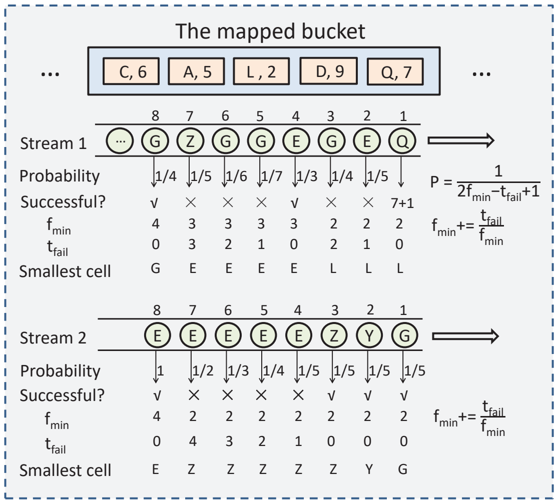


Fig. 4: Examples of GSU.

To clearly show the advantage of GSU, we give two common running examples to show how GSU works.

Example 1 (Figure 4): Suppose that there is a data stream consisting of a series of elements Q, E, G, E, G, G, Z, G. For convenience, we assume that they have the same ID but different intervals, and they are all mapped to the same bucket $S[j]\,(0 \le j \le u)$. For the first incoming element Q, we increment the frequency of Q in the cell by one. For the following two elements E and G, replacements fail, and thus t_{fail} is incremented to 1 and then 2. The probability increases to 1/3 for the fourth element E. Then L is successfully replaced by the fourth element E in the cell, and the frequency f_{min} is incremented by $\frac{t_{fail}}{f_{min}} = 1$, from 2 to 3, and t_{fail} is reset to 0. Since the actual frequency of element E is 2 at present but we record it as 3, the frequency is slightly overestimated. Then the following 3 elements (G, G, Z) arrive and replace E with probabilities 1/7, 1/6, and 1/5, but all fail. Finally, the eighth element G successfully replaces E and the frequency f_{min} is incremented from 3 to 4, which is exactly the frequency of element G by now in data stream 1.

Example 2 (Figure 4): Given another data stream (stream 2): G, Y, Z, E, E, E, E, E, and G, Y, Z, E also have the same ID but different intervals. We suppose that each of the first three elements successfully replaces the cell with the same probability of 1/5, and increments the frequency f_{min} by $\frac{t_{fail}}{f_{min}}(= 0)$ each time. Then element E arrives five times in succession. After four unsuccessful replacements with probabilities 1/5, 1/4, 1/3, and 1/2, the eighth element E replaces element Z in the cell with probability 1, and increments the frequency f_{min} by $\frac{t_{fail}}{f_{min}}(= 2)$, from 2 to 4. The frequency f_{min} is slightly underestimated, since element E has appeared 5 times in the data stream by now. The first three replacements are wrong, but do not bring a large error to the final value of f_{min}.

From the two examples, we see that in our algorithm, the frequency f_{min} might be overestimated or underestimated. Fortunately, an element might be overestimated at first, but it could be underestimated later, and finally the estimate of the element is probably very close to its true value. Moreover, successful replacement of non-periodic elements hardly impacts the final result of f_{min}.

E. Key Parameters of PeriodicSketch

r: The ratio of the memory size of Cover-Min sketch to the memory size of the whole PeriodicSketch.

d: The number of hash functions in Cover-Min sketch. Thus, the Cover-Min sketch can be regarded as d hash tables.

We set the total memory of PeriodicSketch as $\mathcal{M}$. Based on the above parameters, it is easy to get the memory of each hash table of the Cover-Min sketch and GSU sketch:

$$\begin{cases}\mathcal{M}_{HC} = \mathcal{M}r/d \\ \mathcal{M}_{HG} = \mathcal{M}\left(1-r\right)\end{cases}$$

According to the results of our parameter setting experiment, we recommend $r = 15\%$ and $d = 2$, please refer to Section VI-B for more details.

V. Mathematical Analysis

In this section, we separately conduct mathematical analysis on the Cover-Min Sketch and GSU sketch that constitute PeriodicSketch. Initially, we derive the error and memory bounds of the Cover-Min sketch in Section V-A. For the GSU sketch, since its error bound is difficult to derive directly, so we conduct the following mathematical analysis. In Section V-B, we first separately analyze the properties of the probability equation $\mathcal{P}$ proposed in Section IV-D and provide some theoretical results. On the basis of these conclusions, we further analyze the GSU sketch in Section V-C and Section V-D under the real data stream model, as shown below: 1) We prove the feasibility of the GSU in Section V-C, and obtain Theorem V.6, which proves that our GSU sketch can well keep the top-k frequent elements as periodic elements. 2) We derive the memory bound of the GSU sketch in Section V-D, which proves the superiority of GSU from the perspective of memory cost. Finally, we discuss the memory cost of the entire PeriodicSketch and prove that its worst case is $o(n)$ when n is the length of the stream in Section V-E.

A. Error and Memory Bounds of the Cover-Min sketch

Let $A_i[j]$ be the value recorded in the j^{th} bucket of the i^{th} array ($1 \le j \le w, 1 \le i \le d$, $h_i(\cdot)$ be the i^{th} hash function, and let ϵ and δ be two numbers that are related to d and w as follows: $d = \lceil ln(\frac{1}{\delta})\rceil$ and $w = \lceil \frac{exp}{\epsilon}\rceil$. For a given item e at time t, let T_e, be its last occurrence and $\hat{T}_e$ be the estimated timestamp, and $X_{i,(e)}[j] = A_i[j] - T_e$ where $j = h_i(e)$. Note that the time interval $V = t - T_e$ and the estimation $\widehat{V} = t - \hat{T}_e$. Let N be the total number of items, then as all hash function have uniformly distributed output, $Pr[h_i(e_1) = h_i(e_2)] = \frac{1}{w}$. Therefore, $\forall i, j, E(X_{i,(e)}[j]) \le E(A_i[j]) \le \frac{t\cdot N}{w} \le \frac{\epsilon t\cdot N}{exp}$.

First, we derive the probabilistic bound on the underestimation error of the Cover-Min sketch.

$$\begin{aligned}Pr[\widehat{V} \ne V - \epsilon t\cdot N] &= Pr[\hat{T}(e) \ge T(e) + \epsilon t \cdot N] \\ &= Pr[\forall i, A_i[j] \ge T(e) + \epsilon t\cdot N] \\ &= (Pr[X_{i,(e)}[j] \ge \epsilon t \cdot N])^d \\ &\le (Pr[X_{i,(e)}[j] \ge exp \cdot E[X_{i,(e)}[j]])^d \\ &\le exp^{-d} \le \delta\end{aligned}$$

We can also see from above results that given fixed ϵ and δ, the Cover-Min sketch requires the memory of $d \times w = \lceil \frac{exp}{\epsilon} ln(\frac{1}{\delta})\rceil$ counters. For example, suppose $d = 2$, $\epsilon = 0.01$, then w must be at least 272 so that the result holds. Note that $\delta = 0.13$, and the total memory requirement of Cover-Min sketch is $2 \times 272 \times 16$ byte = 8.5KB.

Then, we present the correctness rate of Cover-Min sketch. Suppose that there are K other items between its two occurrences, therefore $\widehat{V} < V$ if and only if hash collisions occur in all d arrays. According to the pairwise independence of hash functions:

$$\begin{aligned}Pr\left[\widehat{V} < V\right] &= \left[1-\left(1-\frac{1}{w}\right)^K\right]^d \\ &\le \left(\frac{K}{w}\right)^d\end{aligned}$$

Note that if K is small compared to w when $V \in [V - \varDelta T, V + \varDelta T]$, then the Cover-Min sketch has a high probability to be able to compute the time intervals precisely.

B. Theoretical Properties of the GSU sketch

Initially, we assume that our hash table only contains one bucket and that f_{min} does not increase before the replacement. Let P_i be the probability that the i^{th} replacement succeeded.

Theorem V.1. *(**Guaranteed** to be replaced)* $0 \le t_{fail} \le 2 \cdot f_{min}$.

Proof. When t_{fail}'s incrementation reaches $2 \cdot f_{\min}$, the probability of replacement becomes

$$\mathcal{P} = \frac{1}{2\cdot f_{min} - t_{fail} + 1} = 1$$

Therefore the success of replacement is guaranteed after several GSU processes and $t_{fail} \le 2 \cdot f_{min}$. End proof. □

Remark. We call the replacement strategy a **soft** method because it is an "uncertain" replacement strategy by probabilistic method.

Theorem V.2. *(**Uniformly** distributed individual replacement probability) For all* $0 \le i \le 2 \cdot f_{min}$, $P_i = \frac{1}{2\cdot f_{min}+1}$.

Proof. When the i^{th} replacement succeeded, $t_{fail} = i - 1$, which means that the first $i-1$ GSU processes all failed. Therefore,

$$\begin{aligned}P_i &= \frac{1}{2\cdot f_{min} - (i-1) + 1} \times \prod_{j=0}^{i-2} \frac{2\cdot f_{min} - j}{2 \cdot f_{min} - j + 1} \\ &= \frac{1}{2\cdot f_{min} - i + 2} \times \frac{2\cdot f_{min} - i + 2}{2\cdot f_{min} - i + 3} \times \cdots \times \frac{2\cdot f_{min}}{2\cdot f_{min}+1} \\ &= \frac{1}{2\cdot f_{min}+1}\left(1 \le i \le 2\cdot f_{min}+1\right).\end{aligned}$$

End proof. □

Theorem V.3. *On average the GSU process activates f_{min} times before the replacement occurs.*

Proof. We calculate the expectation of t_{fail}:

$$\begin{aligned}\mathbb{E}\left(t_{fail}\right) &= \sum_{i=0}^{2f_{min}} \frac{i}{2\cdot f_{min}+1} \\ &= \frac{1}{2\cdot f_{min}+1}\cdot\frac{1}{2}\cdot(2f_{min})\cdot(2f_{min}+1) \\ &= f_{min}.\end{aligned}$$

End proof. □

Remark. According to above results, on average f_{min} increases by 1 and the smallest element is most likely to be replaced by the element that activates the GSU process the most.

Theorem V.4. *(Non-asymptotic upper* ***error bound****)* $\hat{f} \le f + f_{min}$

Proof. We have $f = \hat{f}$ if an element has never been the least frequent element (L). Otherwise, suppose f'_{min} to be its frequency the last time when it's L, note that from then on, the increment of its frequency would not exceed f and $f'_{min} \le f_{min}$. Thus, $\hat{f} \le f + f_{min}$. End proof. □

C. Quasiconsistency of the GSU sketch

Consider an i.i.d data stream with a finite domain $S = \{E_1, E_2, \cdots, E_M\}$, and their arrival probability $p_1 > p_2 > \cdots > p_M$. Assume that the hash table only contains one bucket with λ cells, where $\lambda << M$ and elements are denoted as $H = \{\widehat{E_1}, \widehat{E_2}, \cdots, \widehat{E_\lambda}\}$. Note that the least frequent element (L) is $\widehat{E_\lambda}$, and let $S_1 = \{E_1, E_2, \cdots, E_{\lambda-1}\}, S_2 = S \backslash S_1$, and $\hat{f}(E)$ be the estimated frequency of E.

In this subsection we would like to show from two aspects that when the data stream is sufficiently large, then GSU choose elements with largest frequency with high probability. (We call this quasiconsistency) Thus, our algorithm can provide an accurate solution to the top-k problem.

Theorem V.5. *If $p_i^2 \ge p_j$, and $E_j \ne L$ is in the hash table while E_i is L or not in the hash table. Then as $t \to \infty$, E_i would be inserted in the hash table and not be L.*

Remark. Theorem V.5 shows that GSU choose elements whose frequency are larger to some extent.

Next, we present the main theorem of **quasiconsistency**:

Theorem V.6. ***(quasiconsistency)*** *If at the time t', $\widehat{E_\lambda} \in S_2$, $f_{min} > \frac{1}{\min_{E_i\in S_1 E_j \in S_2}|p_i - p_j|}$, and $S_1 \cap H \backslash \{\widehat{E_\lambda}\} = S_1 \backslash \{\widehat{E_\lambda}\}$. Then*

$$S_1 \cap H \backslash \{\widehat{E_\lambda}\} = S_1 \backslash \{\widehat{E_\lambda}\}$$

holds always with probability at least

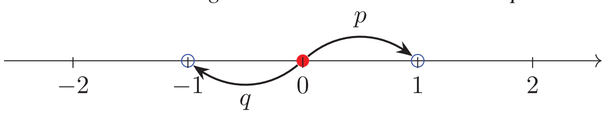
$$1 - c\left(\frac{p_{\lambda-1}}{p_\lambda}\right)^{(\min \hat{f}(H\backslash\{\widehat{E_\lambda}\}) - f_{min})}.$$

where c is a constant determined by $p_{\lambda-1}$ and p_λ.

Remark. The equation in the above theorem means that all elements in S_1 appear in the bucket. Keeping this equation established means that the vast majority of hot elements have a certain probability that they will not be replaced if they are recorded by GSU. Because we consider an infinite data stream here, there will certainly be a time when GSU arrives at a situation where this equation holds. Also, consider the fact that we do not have any prior knowledge of the coming data streams, it is natural to assume that those elements who come the earliest and the most will remain in the hash table. According to the theorem, as the time goes by, those with highest frequency will almost not change, which is enough for our top-k estimation.

Before we begin the proof of the main theorem, we need two more lemmas. We will prove Lemma V.2 using Lemma V.1 and then we will give the proof of Theorem V.6.

Lemma V.1. *Suppose a simple random walk on $\mathbb{Z}$ has a diagram with transition probabilities*



satisfying $p > q$. Let τ_{n0} denote the hitting time starting from n and ending at 0, and let $\rho_{n0} = P(\tau_{n0} < +\infty)$. Then we conclude that $\rho_{n0} = \alpha^n, \alpha = \frac{q}{p}, n \ge 0$.

Proof. By definition, if $i \ge 1$, we have

$$\rho_{i0} = q\rho_{i-1,0} + p\rho_{i+1,0}.$$

Obviously $\rho_{00} = 1$. Then the equation above is indeed an order-2 homogeneous linear frequency relations with constant coefficients. Thus, we can easily derive that ρ_{n0} is of the form $\alpha^n, n \ge 0$. Along with the transient property for the random walk, $0 < \alpha < 1$.

Substitute this exponential form into the above equation, we get $\alpha = \frac{q}{p}$. End proof. □

Lemma V.2. *Suppose a random walk with biased steps has a*

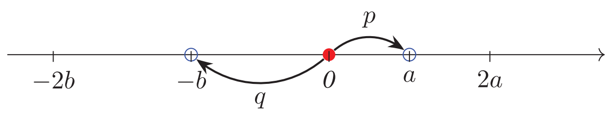


diagram with transition probabilities satisfying $p > q$, and biased steps $a < b$. Let τ_{n0} denote the hitting time starting from n and ending at the negative half axis, and let $\rho_{n0} = P(\tau_{n0} < +\infty)$. More precisely, set $a = 1 - \frac{1}{f_{min}}, b = 1 + \frac{1}{f_{min}}$. If $pa > bq$, we conclude that $\rho_{x0} \le (c+1)\cdot\alpha^{\lfloor x/b \rfloor}, \alpha = \frac{q}{p}, x \ge 0$, where c is a constant determined by p and q.

Proof. By Kolmogorov's strong law of large numbers, we easily know that the biased random walk is transient. And it will approach $+\infty$ eventually. For any $x > b > 0$, we can get

$$\rho_{x0} = q\rho_{x-b,0} + p\rho_{x+a,0}.$$

Since it is obvious that ρ_{x0} is decreasing with respect to x, the equation can transform to the following inequality:

$$\rho_{x,0} \geq q\rho_{x-a,0} + p\rho_{x+a,0}.$$

Assume that $x = an, n \in \mathbb{Z}^+$ without harm to our aim. From this inequality, we get

$$q(\rho_{x,0} - \rho_{x-a,0}) \geq p(\rho_{x+a,0} - \rho_{x,0}).$$

$$\Rightarrow \rho_{x+a,0} - \rho_{x,0} \leq (\frac{q}{p})^n(\rho_{a0} - 1).$$

Let $\kappa = \rho_{a0} - \frac{p}{q}$, then

$$\Rightarrow \rho_{x+a,0} - \rho_{x,0} \leq (\frac{q}{p})^n(\frac{p}{q} + \kappa - 1).$$

From Lemma V.1,

$$\rho_{x+a,0} - \rho_{x,0} \leq \alpha^n + \kappa \cdot \frac{p}{p-q}.$$

It remains to bound κ. After direct calculations, $\kappa \leq (1 - q)(P(\tau_{x0} > f_{min}, \tau_{x0} < +\infty))$. End Proof. □

Now we present proof of Theorem V.6 based on the above lemma:

Proof. We will prove the theorem using skills called the decomposition of stochastic process and comparison of Markov chain.

For any $E_i \in S_1, E_j \in S_2$, we have

$$p_i - p_j > \frac{1}{\cdot f_{min}}.$$

Let $\Delta_t = \min \hat{f}(H \backslash \{\widehat{E_\lambda}\}) - f_{min}, t \geq t'$, then Δ_t is a discrete time non-homogeneous Markov chain. So we decompose it to

$$\Delta_t = \Delta_1^t + \Delta_2^t.$$

Then we construct another Markov chain Π_1^t, which is a birth and death chain with delay, starting from $\Delta_{t'}$. The birth rate is $p_{\lambda-1}$ and the death rate is p_λ. Easy to prove that Δ_1^t is stochastically larger than Π_1^t. Thus, without loss of generality, suppose that Δ_1^t is a birth and death chain with delay. The transition probabilities are

$$\begin{cases} P(\Delta_1^{t+1} = \Delta_1^t + 1 | \Delta_1^t) = p_{\lambda-1}. \\ P(\Delta_1^{t+1} = \Delta_1^t - 1 | \Delta_1^t) = p_\lambda. \\ P(\Delta_1^{t+1} = \Delta_1^t | \Delta_1^t) = 1 - (P(\Delta_1^{t+1} = \Delta_1^t + 1 | \Delta_1^t) \\ \qquad + P(\Delta_1^{t+1} = \Delta_1^t - 1 | \Delta_1^t)) \end{cases}$$

However, the complex part hides in Δ_2^t. It is a non-homogeneous Markov chain with transition probabilities:

$$\begin{cases} P(\Delta_2^{t+1} = \Delta_2^t + 1 | \Delta_2^t) = 0. \\ P(\Delta_2^{t+1} = \Delta_2^t - 1 | \Delta_2^t) = 0, \quad 0 \leq t_{fail} \leq f_{min}. \\ P(\Delta_2^{t+1} = \Delta_2^t - 1 | \Delta_2^t) = \dfrac{1}{2 \cdot f_{min} - t_{fail} + 1}, \\ \qquad f_{min} + 1 \leq t_{fail} \leq 2 f_{min}. \\ P(\Delta_2^{t+1} = \Delta_2^t | \Delta_2^t) = 1 - (P(\Delta_2^{t+1} = \Delta_2^t + 1 | \Delta_2^t) \\ \qquad + P(\Delta_2^{t+1} = \Delta_2^t - 1 | \Delta_2^t)) \end{cases}$$

Construct Π_2^t. The transition probabilities are

$$\begin{cases} P(\Delta_2^{t+1} = \Delta_2^t + 1 | \Delta_2^t) = 0. \\ P(\Delta_2^{t+1} = \Delta_2^t - 1 | \Delta_2^t) = \delta(t_{fail}, f_{min}) \\ P(\Delta_2^{t+1} = \Delta_2^t | \Delta_2^t) = 1 - (P(\Delta_2^{t+1} = \Delta_2^t + 1 | \Delta_2^t) \\ \qquad + P(\Delta_2^{t+1} = \Delta_2^t - 1 | \Delta_2^t)) \end{cases}$$

where δ is the Kronecker delta symbol. Easy to prove that Δ_2^t is stochastically larger than Π_2^t, since f_{min} is monotonically nondecreasing. Now we have transformed our original complex question to an easier question on the process $\Pi_t = \Pi_1^t + \Pi_2^t$. This, however, is equivalent to a biased random walk with steps defined in Lemma V.2, and $p = p_{\lambda-1}, q = p_\lambda$ (here we may ignore some normalised constants). Easily check that the condition in the Lemma V.2 has been satisfied. Thus, the conclusion is immediate. End proof. □

D. Memory Bounds of the GSU sketch

In this subsection, we will focus on i.i.d streams like V-C and compute memory cost of the GSU sketch. Suppose a Zipf stream with a finite Domain $S = \{E_1, E_2, ..., E_M\}$ and skewness $\alpha = 1$, and let the arrival probability of E_i be $p_i = \frac{i^{-1}}{\Sigma_{n=1}^M n^{-1}}$. Define $\Sigma_1^D = \Sigma_{n=1}^D n^{-1} = log\,(1.78D) + o\,(1)$ ($D \geq 1$). We again assume that the hash table only contains one bucket. We define that the algorithm can successfully identify the top-k elements when $lim_{n\to\infty} P_{\lambda,k}(n) = 1$($\lambda$ is the number of cells in the bucket, k is the number of the most frequent elements we need, and n means after processing n elements). We call the largest k cells in the bucket head counters and the other $\lambda - k$ cells tail counters. In order to make $lim_{n\to\infty} P_{\lambda,k}(n) = 1$, we must guarantee that the increment rate of tail counters must below all the increment of head counters. Because the increment rate of the k^{th} element is the slowest among all head counters, we only need to guarantee that the increment rate of tail counters is below the the increment rate of the k^{th} element.

Theorem V.7. *To solve the top-k problem with i.i.d Zipf stream and skewness α = 1, for any fixed $k << M$, GSU requires $O(\sqrt{logM})$ cells.*

Proof. Every time when an element E_i appears, if $i \in [k + 1, \lambda]$, approximately that one of tail counters increase by 1. But if the rank of element is $\in [\lambda + 1, M]$, the smallest element will be replaced with probability $\tilde{P}$. Therefore, the increment rate of tail counters is:

$$\hat{p} = \frac{\Sigma_{i=k+1}^\lambda p_i + \tilde{P} \cdot \Sigma_{i=\lambda+1}^M p_i}{\lambda - k} = \frac{\Sigma_1^\lambda - \Sigma_1^k + \tilde{P} \cdot (\Sigma_1^M - \Sigma_1^\lambda)}{\Sigma_1^M \cdot (\lambda - k)}$$

GSU must guarantee $p_k > \hat{p}$, therefore:

$$\begin{aligned} \lambda &> k + k(\Sigma_1^\lambda - \Sigma_1^k + \tilde{P} \cdot (\Sigma_1^M - \Sigma_1^\lambda)) \\ &= k + k \cdot O\left(log\frac{\lambda}{k} + \tilde{P} \cdot log1.78M\right) \\ &= O\left(\tilde{P} \cdot klogM\right) \end{aligned}$$

We suppose N is the total number of the elements. We must guarantee the least frequent element will appear. Therefore,

$$N \cdot \frac{M^{-1}}{\Sigma_1^M} \geq 1 \Rightarrow N \geq \frac{\Sigma_1^M}{M^{-1}}$$

Then, we consider $\tilde{\mathcal{P}}$, which is relative to f_{min} and the number of the λ^{th} frequent element, and that M is sufficiently large:

$$\begin{aligned}\tilde{\mathcal{P}} &= \frac{1}{(2 \cdot f_{min} - t_{fail} + 1)} \\ &= O\left(\frac{1}{f_{min}}\right) = O\left(\frac{\lambda^{-1}}{\Sigma_1^M} \cdot N\right)^{-1} \\ &\leq O\left(\frac{\lambda}{M}\right) = O\left(\frac{klogM}{M}\right) \leq O\left(\frac{1}{k\sqrt{logM}}\right)\end{aligned}$$

Therefore, GSU requires $\lambda = O\left(\sqrt{logM}\right)$. End proof. $\square$

E. Memory Cost of the Entire PeriodicSketch

Here, we discuss the overall memory cost of PeriodicSketch. Firstly, suppose M is the number of items inserted into the GSU sketch, and GSU has one bucket with $\lambda = M$ cells. Note that all items will be placed in a corresponding position and no item will be replaced, we can conclude that in this case GSU has zero error and we can solve the top-k problem with at most $O(M)$ memory. Secondly, let N be the number of items in the data stream (the stream length) and N_0 be the number of distinct items, we have $M = o(N), N_0 = o(N)$ and that if $w = O(N_0)$ then there is a high probability that there will be no hash collisions, so the Cover-Min sketch has high accuracy with $O(N_0)$ memory. In conclusion, we can find the periodic items with at most $o(N)$ memory.

Since it is hard to accurately analysis the memory cost of the entire PeriodicSketch, we only get the above result $o(N)$, which is a loose theoretical bound. In fact, the experimental results shows that our memory cost is much better compared with N (*e.g.*, $1.8 * 10^6$ bytes for $N = 10^8$ items).

VI. Experimental Results

In this section, we show the experimental results of PeriodicSketch. First, we describe the experimental setup in Section VI-A. Second, how parameter settings affect PeriodicSketch's performance is shown in Section VI-B. Third, we evaluate the performance of PeriodicSketch on different datasets and compare it with the baseline solution in Section VI-C. Finally, through the Cache prefetch experiment, we provide a concrete case of the application of PeriodicSketch.

A. Experimental Setup

Dataset: We use four real-world datasets, CAIDA datasets (CAIDA2016, CAIDA2018), MAWI datasets and MACCDC datasets. They can be divided into the following 3 types according to the sources.

1) IP Trace Dataset. The IP Trace Dataset is streams of anonymized IP traces collected by CAIDA [38]. Our experiment uses CAIDA2016 and CAIDA2018. They consist of many types of traffic (*e.g.*, DDoS attack, TCP and UDP probing, BGP monitoring) [39], and have different flow size distribution and different portion of the traffic type (*i.e.*, the proportion of their TCP is 44% and 65%) [40]. For IP Trace Dataset, there are around 30M items and 900K distinct items.

2) Packet Traffic Traces Dataset. This real traffic traces data is provided by the MAWI Working Group [41]. For MAWI Dataset, there are around 9M items and 13K distinct items.

3) MACCDC: The MACCDC datasets is provided by the U.S. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) [42]. For MACCDC Dataset, there are around 13M items and 400K distinct items.

For the above datasets, each item contains a source IP address (4 bytes) and a destination IP address (4 bytes), 8 bytes in total. Timestamps are in microseconds.

Default Parameter: We set ΔT to a fixed value of $1ms$.

Implementation: We have implemented PeriodicSketch and the baseline solution in C++. The hash functions are implemented using the 32-bit Bob Hash (obtained from the open-source website [43]) with different initial seeds.

Computation Platform: We conducted all the experiments on a machine with one 4-core processor (8 threads, Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz) and 16 GB DRAM memory. The processor has 64KB L1 cache, 256KB L2 cache for each core, and 6MB L3 cache shared by all cores.

Metrics:

1) Precision Rate (PR): Precision Rate (PR) is the ratio of the number of correctly periodic items to the number of periodic items reported.

2) Recall Rate (RR): Recall Rate (RR) is the ratio of the number of correctly reported periodic items to the number of correctly periodic items.

3) Average Absolute Error (AAE): For each ID e that is periodic, we calculate the average absolute error for the predicted mean. We define the average absolute error as $\text{AAE} = \frac{1}{|\Psi|}\sum_{e_i \in \Psi} |f_i - \hat{f}_i|$, where f_i is the real frequency of periodic item e_i, $\hat{f}_i$ is its estimated frequency of periodic item, and Ψ is the query set.

4) Average Relative Error (ARE): We define the average relative error as $\text{ARE} = \frac{1}{|\Psi|}\sum_{e_i \in \Psi} \frac{|f_i - \hat{f}_i|}{f_i}$, where f_i is the real frequency of periodic item e_i, $\hat{f}_i$ is its estimated frequency of periodic item, and Ψ is the query set.

5) Throughput: We use Million of operations (insertions) per second (Mips) to measure the throughput. Experiments are repeated 10 times and the average throughput is reported.

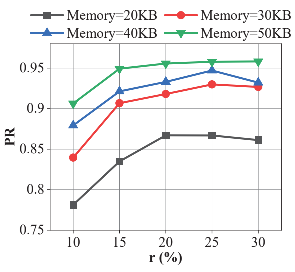
Baseline Solution: As the first method to find periodic items in data streams, and considering that existing works cannot be directly compared with our work, we propose a sketch-based baseline approach of finding periodic items in data streams. Similar to PeriodicSketch, our baseline solution processes new items by two steps. First, we record the historic appearances of the item using Bloom filters [20]. Second, we use the Space-Saving [21] algorithm to find frequent items with the same intervals.

STEP 1: Specifically, we first divide the timeline into many pieces of size $\mathcal{X}$, *i.e.*, the i^{th} time piece is a time span $[(i-1) \times \mathcal{X}, i \times \mathcal{X})$. Each time piece has a Bloom filter, a compact data structure recording all the items arrived in the time piece. So that we know whether any item appears in any time piece, with tolerable error.

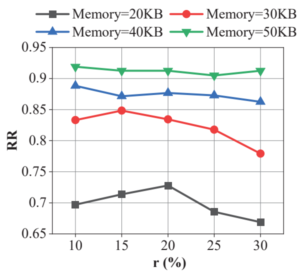
STEP 2: For each inserted item (e), we first look up the Bloom filters, and then subtract the last arrival time (*i.e.*, time piece ID l) from the current time (*i.e.*, time piece ID c) to get an estimate of the time interval $V^* = (c - l)\mathcal{X} \approx V$. Next, we combine the item ID e with the interval V^* to get item pair $e' =< e, V^* >$. We use the Space-Saving algorithm, the most well known algorithm of finding top-K frequent items, to find K most frequent item pairs. Finally, we treat them as periodic elements for return.

B. Experiments on Parameter Settings

In this section, we conduct several experiments to measure the effects of key parameters of PeriodicSketch, namely, the ratio r of the memory size of Cover-Min sketch to the memory size of the whole PeriodicSketch, and the number of hash functions d in Cover-Min sketch. In other words, the Cover-Min sketch is composed of d hash tables, while the GSU sketch is composed of one hash table. We use the CAIDA2016 in these experiments, and PR and RR to evaluate the effects.



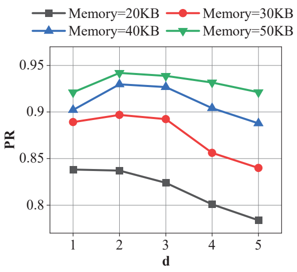
(a) PR of r



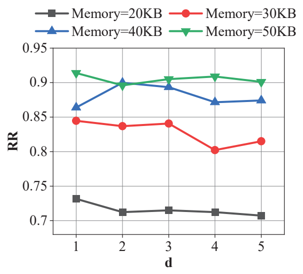
(b) RR of r

Fig. 5: Effect of the system parameter r.

Effect of ratio r (Figure 5): The experimental results show that the best value for the ratio r is from 15 to 20. When $r = 10$ to $r = 25$, PR increases with the increase of r, and tends to be stable when $r = 20$ and $r = 25$. RR first increases slightly, and stabilizes at $r = 15$ and $r = 20$, and then decreases at $r = 25$ and $r = 30$.



(a) PR of d



(b) RR of d

Fig. 6: Effect of the system parameter d.

Effect of ratio d (Figure 6): The experimental results show that the optimal value for d is 2. When $d = 2$, PR has a maximum value. PR tends to decrease when $d = 3, 4, 5$. RR has an optimal value when $d = 2$ and $d = 3$. When $d = 3$ to $d = 5$, RR has a tendency to decrease and fluctuate.

We must emphasize that even if the ARE of r and d is slightly larger in some values, our sketch is far superior to the baseline solution.

Parameter Selection: Based on the comprehensive analysis of the results (Figure 5 and 6), we recommend the selection of key parameters as follows: $r = 15(\%)$ and $d = 2$.

C. Comparison with Baseline Solution

In this section, we compare PeriodicSketch's performance with the baseline solution in the metrics below. For PeriodicSketch, we set the memory size range to 50-150KB. In addition, for the values of the main system parameters r and d, we directly choose the recommendations in Section VI-B: $r = 15(\%)$ and $d = 2$. **It should be noted here that for the baseline solution, the memory size range is set to 500-1500KB, which is 10 times higher than that of PeriodicSketch.**

PR (Figure 7): *This experiment shows that the PR of PeriodicSketch is greatly outperforms the baseline solution.* The results show that PeriodicSketch's PR is about 77.3% higher than the baseline solution.

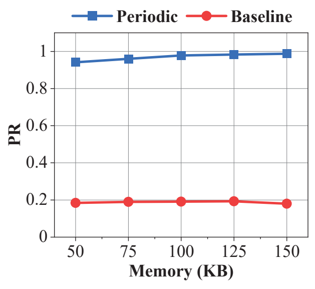
RR (Figure 8): *This experiment shows that RR of PeriodicSketch is overwhelmingly higher than the baseline solution.* Compared to the baseline solution, RR of PeriodicSketch is about 74.4% higher in average.

AAE (Figure 9): *This experiment shows that AAE of PeriodicSketch is significantly lower than the baseline solution.* On different datasets, the AAE of PeriodicSketch has different degrees of advantage over the baseline solution. Specifically, the AAE of the PeriodicSketch is between 173 to 206, 273 to 360, 1346 to 2019, and 656 to 830 times lower than the baseline solution on the four datasets, respectively. On average, the AAE of the PeriodicSketch is 737 times smaller than the baseline solution.

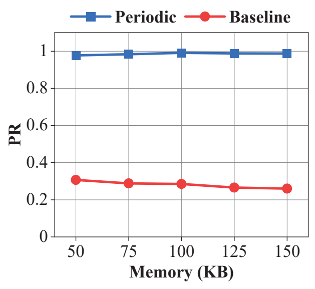
ARE (Figure 10): *This experiment shows that ARE of PeriodicSketch is obviously lower than the baseline solution.* On different datasets, the ARE of PeriodicSketch has different degrees of advantage over the baseline solution. Specifically, the ARE of the PeriodicSketch is between 128 to 168, 231 to 290, 218 to 270, and 479 to 551 times lower than the baseline solution on the four datasets, respectively. On average, the ARE of the PeriodicSketch is 284 times smaller than the baseline solution.

Throughput (Figure 11): *Our results show that the insertion throughput of the PeriodicSketch is much higher than that of the baseline solution.* The throughput of PeriodicSketch is 14.5Mips in average. Moreover, the throughput of PeriodicSketch is between 2.8 to 4.1 times higher than that of the baseline solution.

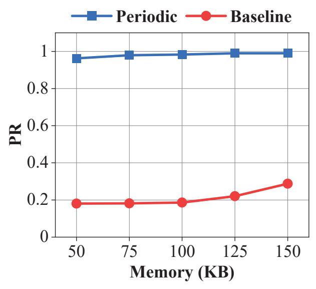
Analysis: Our experimental results show that our PeriodicSketch can achieve high accuracy and high throughput in small memory. The baseline solution needs much more memory to work because it needs multiple Bloom filters to record the information of every time piece, while our PeriodicSketch only use one Cover-Min sketch to record the approximate state of each item, which largely save the space. In addition, the baseline solution needs to check multiple Bloom filters in one insertion, which decreases its throughput, while our PeriodicSketch guarantees that it can finish one insertion in $O(1)$ time.



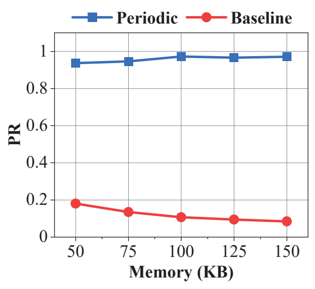
(a) CAIDA2016



(b) CAIDA2018

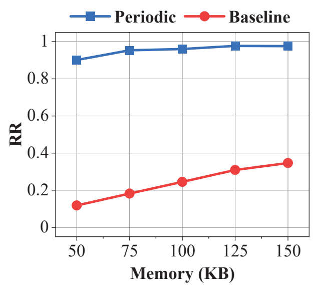


(c) MAWI

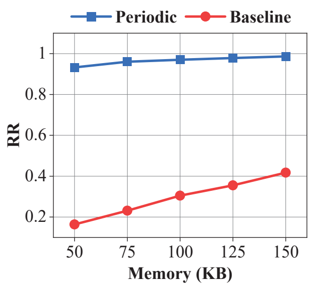


(d) MACCDC

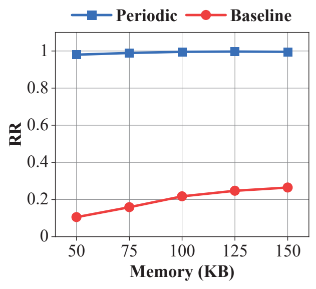
Fig. 7: The Periodic Precision Rate (PR) vs. memory.



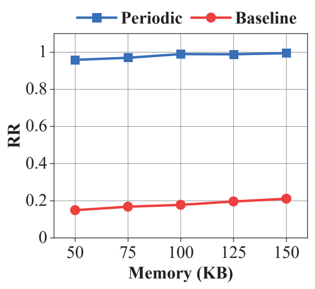
(a) CAIDA2016



(b) CAIDA2018

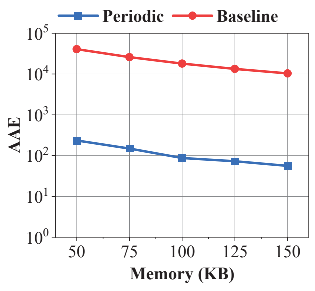


(c) MAWI

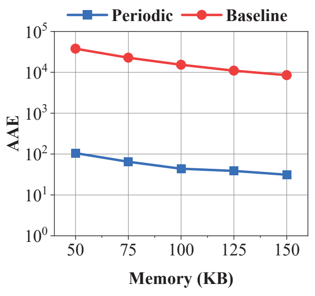


(d) MACCDC

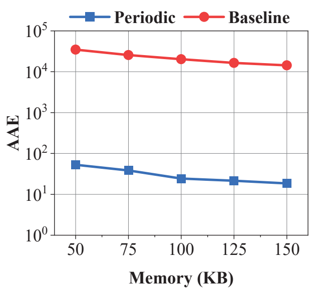
Fig. 8: The Periodic Recall Rate (RR) vs. memory.



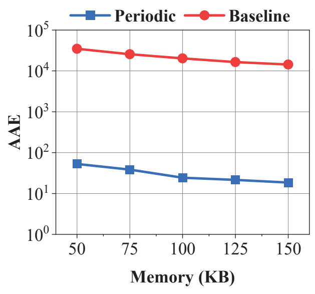
(a) CAIDA2016



(b) CAIDA2018

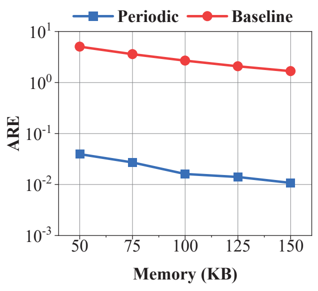


(c) MAWI

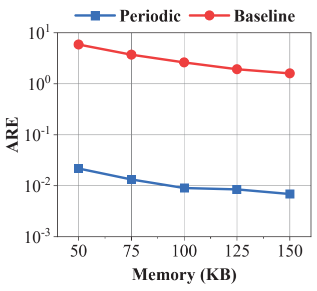


(d) MACCDC

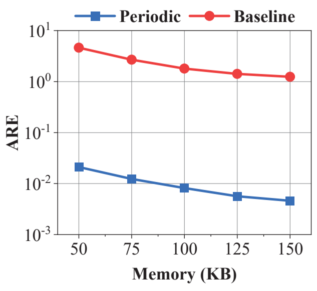
Fig. 9: The Periodic Average Absolute Error (AAE) vs. memory.



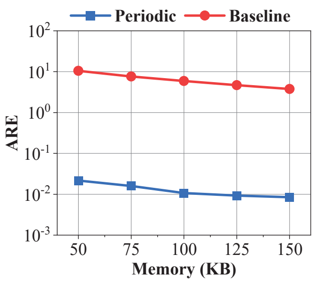
(a) CAIDA2016



(b) CAIDA2018



(c) MAWI



(d) MACCDC

Fig. 10: The Periodic Average Relative Error (ARE) vs. memory.

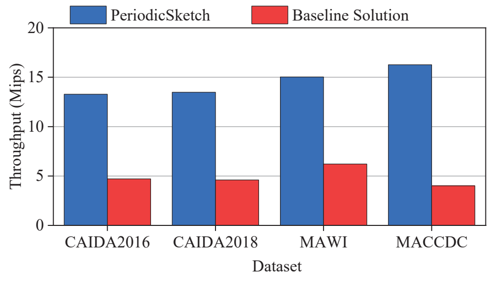


Fig. 11: Throughput comparison.

D. End-to-End Experiments for a Concrete Case

We implement the **Cache Prefetching** (Case 1) through end-to-end experiments in this section. Specifically, we observe whether applying PeriodicSketch to Least Recently Used (LRU) [44] and Least Frequently Used (LFU) [45] algorithms improves the Cache hit ratio. The details are as follows:

Experimental Setup: In this experiment, there is a Cache and a PeriodicSketch. When each period/interval starts, we prefetch the periodic items with a frequency grater than 5 from the PeriodicSketch, and insert them into the Cache. We use this prefetch method to improve the performance of two famous Cache strategies: LRU and LFU. LRU evicts the least recently used items when a new item comes. LFU evicts the least frequently used items instead. *Cache Hit Ratio* refers to (Number of Cache hits)/(Number of Cache hits + Number of Cache misses). We show the experimental results of two real-world datasets[2] (CAIDA2018 and Criteo[3]) and one Synthetic dataset. Our Synthetic dataset has 10^8 items totally. Among them, the items of the $P_{\%}$ ratio are periodic items, each of which has a period with the interval of $5*10^4$ items and repeats for 2000 times. The other items are random items, and most of them only appear once. The memory usage is as follows: 1) PeriodicSketch uses $3*10^5$ bytes (293KB) in the real-world datasets; 2) PeriodicSketch uses $1.8*10^6$ bytes (1758KB) in the Synthetic dataset[4]; 3) the memory used by the Cache is the Cache size*4 bytes in the figures below.

Experimental Results (Real-World Datasets): We find that PeriodicSketch significantly improves the Cache hit ratio of LFU or LRU by more than 10% in a large Cache Size range from Figure 12(a)-12(b). In CAIDA2018 and Criteo, the proportions of periodic items among all items/packets are 45.0% and 96.0%, respectively.

Experimental Results (Synthetic Dataset): Figure 13(a)-13(b) shows that PeriodicSketch can improve the Cache hit ratio of LFU and LRU by more than 100 times when the Cache size is small. The reason why LRU and LFU performs poorly is the Cache thrash [48], which is a cascade of Cache misses caused by a special traffic pattern ($e.g.$, periodic items).

[2]In fact, we have conducted experiments on more than 10 real-world datasets, but the proportion of periodic items in some datasets is too small to help improve the Cache Hit Ratio (explain later). In addition, we also find that the real-world datasets we have experimented so far are difficult to significantly improve the Cache hit ratio for both LRU and LFU. Thus, we separately select a real-world dataset as a representative from all datasets that can help improve LRU or LFU to show the experimental results.

[3]Criteo Dataset [46] contains feature values and conversion feedback for clicked display ads sampled over a two-month period. Every ad is associated with a timestamp and 9 categorical terms hashed for anonymity, for a total of 150K unique hashed categorical terms [47].

[4]Since our Synthetic dataset has many periodic items, a larger memory is needed to store these periodic items.

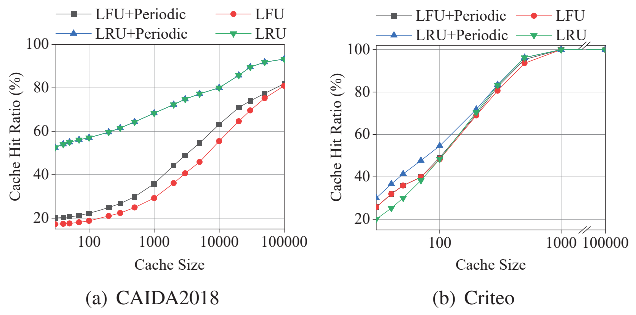


Fig. 12: The Cache hit ratio of LFU and LRU on the CAIDA2018 and Criteo datasets, where the proportion of periodic items to the total items in the two datasets is as follows: $1.22*10^7/2.71*10^7 \approx 45.0\%$, $9.6*10^6/10^7 = 96.0\%$.

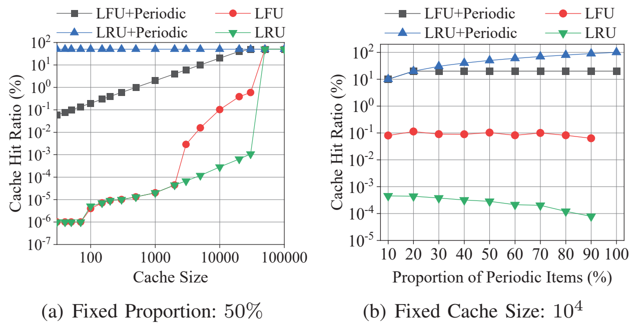


Fig. 13: The Cache hit ratio of LFU and LRU on the Synthetic. When $P_{\%} = 100\%$, the Cache hit ratio of LFU and LRU is 0.

VII. Conclusion

Finding periodic items in high-speed data streams in real time plays an essential role in many applications. This paper proposes a novel algorithm called PeriodicSketch for finding top-K periodic items in *data streams*, which is the first sketch-based method to solve this problem. Our first key technique, the Cover-Min sketch, is used to record and report the interval for each incoming item. Our second key technique is called GSU replacement strategy, which is able to differentiate periodic items from others with high probability, in limited memory space. Experimental results show that PeriodicSketch can achieve around 77.3% higher PR, 74.4% higher RR, 3.1 times higher throughput, and up to 2019 times (737 times in average) lower ARE than the baseline solution. Finally, our Cache prefetch experiment verify that PeriodicSketch can significantly improve the Cache hit ratio on several datasets.

Acknowledgment

We would like to thank our Sheng Long, Ruwen Zhang and the anonymous reviewers, for their help in improving this paper. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, National Natural Science Foundation of China (NSFC) (No. U20A20179, 61832001).

REFERENCES

[1] "The source codes of our and other related algorithms." [Online]. Available: https://github.com/pkufzc/PeriodicSketch

[2] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, 2005.

[3] 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 (SIGCOMM)*, 2018, pp. 561–575.

[4] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang, and N. Zhang, "Lightguardian: A Full-Visibility, lightweight, in-band telemetry system using sketchlets," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021, pp. 991–1010.

[5] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch." in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 29 – 42.

[6] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1699 – 1699, 2012.

[7] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li, "Heavyguardian: Separate and guard hot items in data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018, pp. 2584 – 2593.

[8] Y. Zhang, J. Li, Y. Lei, T. Yang, Z. Li, G. Zhang, and B. Cui, "On-off sketch: A fast and accurate sketch on persistence," *Proceedings of the VLDB Endowment*, vol. 14, no. 2, pp. 128 – 140, 2020.

[9] S. A. Singh and S. Tirthapura, "Monitoring persistent items in the union of distributed streams," *Journal of Parallel and Distributed Computing*, vol. 74, no. 11, pp. 3115–3127, 2014.

[10] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 289 – 300, 2016.

[11] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proceedings of the 28th Conference on Computer Communications (INFOCOM)*, 2009, pp. 504 – 512.

[12] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," *Department of Electrical and Computing Engineering*, 2005.

[13] A. Floratou, N. Megiddo, N. Potti, F. Ozcan, U. Kale, and J. Schmitz-Hermes, "Adaptive caching in big sql using the hdfs cache," in *Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*, 2016, pp. 321 – 333.

[14] Y. Zhang and R. Gupta, "Enabling partial cache line prefetching through data compression," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, 2003, pp. 277 – 285.

[15] E. Cole, *Advanced Persistent Threat: Understanding the Danger and How to Protect Your Organization*. Syngress Publishing, 2012.

[16] "Mila Parkour. Contagio malware database." [Online]. Available: https://www.mediafire.com/folder/c2az029ch6cke/base

[17] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. a. Gama, "Machine learning for streaming data: State of the art, challenges, and opportunities," *ACM SIGKDD Explorations Newsletter*, vol. 21, no. 2, p. 6–22, 2019.

[18] C. Pirrong, "Energy market manipulation: Definition, diagnosis, and deterrence," *The Energy Law Journal*, vol. 31, 2010.

[19] T. Chen, H. Yin, H. Chen, H. Wang, and X. Li, "Online sales prediction via trend alignment-based multitask recurrent neural networks," *Knowledge and Information Systems*, vol. 62, no. 7, pp. 1–29, 2020.

[20] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[21] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of the 10th International Conference on Database Theory (ICDT)*, 2005, pp. 398–412.

[22] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2018, pp. 1129 – 1140.

[23] 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. 270 – 313, 2003.

[24] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2002.

[25] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1622 – 1634, 2012.

[26] D. Fan and R. Davood, "New estimation algorithms for streaming data: Count-min can do more." *Webdocs. Cs. Ualberta. Ca*, 2007. [Online]. Available: https://webdocs.cs.ualberta.ca/∼fandeng/paper/cmm.pdf

[27] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2016, pp. 1449 – 1463.

[28] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2018, pp. 741 – 756.

[29] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-k periodic-frequent pattern from transactional databases without support threshold," *Communications in Computer and Information Science*, vol. 55 CCIS, pp. 18 – 29, 2009.

[30] S. K. Tanbeer, C. F. Ahmed, and B.-S. Jeong, "Mining regular patterns in data streams," in *Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2010, pp. 399–413.

[31] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Stagger: Periodicity mining of data streams using expanding sliding windows," in *Proceedings of the 6th International Conference on Data Mining (ICDM)*, 2006, pp. 188 – 199.

[32] Q. Yuan, J. Shang, X. Cao, C. Zhang, X. Geng, and J. Han, "Detecting multiple periods and periodic patterns in event time sequences," in *Proceedings of the 26th ACM on Conference on Information and Knowledge Management (CIKM)*, 2017, pp. 617 – 626.

[33] M. Toller, T. Santos, and K. R., "Sazed: parameter-free domain-agnostic season length estimation in time series data," *Data Mining and Knowledge Discovery*, vol. 33, no. 6, pp. 1775 – 1798, 2019.

[34] Q. Wen, K. He, L. Sun, Y. Zhang, M. Ke, and H. Xu, "Robustperiod: Time-frequency mining for robust multiple periodicities detection," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2021, pp. 2328 – 2337.

[35] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010, pp. 1099 – 1108.

[36] S. Ma and J. Hellerstein, "Mining partially periodic event patterns with unknown periods," in *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, 2001, pp. 205 – 214.

[37] F. Ergun, H. Jowhari, and M. Sağlam, "Periodicity in streams," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2010, pp. 545–559.

[38] "The CAIDA Anonymized Internet Traces." [Online]. Available: http://www.caida.org/data/overview/

[39] A. Ferriyan, A. H. Thamrin, K. Takeda, and J. Murai, "Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic," *Applied Sciences*, vol. 11, no. 17, p. 7868, 2021.

[40] S. Bauer, B. Jaeger, F. Helfert, P. Barias, and G. Carle, "On the evolution of internet flow characteristics," in *Proceedings of the Applied Networking Research Workshop (ANRW)*, 2021, p. 29–35.

[41] "MAWI Working Group Traffic Archive." [Online]. Available: http://mawi.wide.ad.jp/mawi/

[42] "Capture files from Mid-Atlantic CCDC." [Online]. Available: https://www.netresec.com/?page=MACCDC

[43] "The source code of Bob Hash." [Online]. Available: http://burtleburtle.net/bob/hash/evahash.html

[44] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 1990, pp. 143 – 152.

[45] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies," in

Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), 1999, pp. 134 – 143.

[46] "The Criteo dataset Internet Traces." [Online]. Available: https://ailab.criteo.com/criteo-attribution-modeling-bidding-dataset/

[47] P. Chen, D. Chen, L. Zheng, J. Li, and T. Yang, "Out of many we are one: Measuring item batch with clock-sketch," in *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, 2021, pp. 261 – 273.

[48] P. J. Denning, "Thrashing: Its causes and prevention," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS)*, 1968, pp. 915 – 922.