DUET: A Generic Framework for Finding Special Quadratic Elements in Data Streams

Jiaqian Liu [1], Haipeng Dai[1], Rui Xia[1], Meng Li[1], Ran Ben Basat[2], Rui Li[3], Guihai Chen[1]
liujiaqian@smail.nju.edu.cn, haipengdai@nju.edu.cn, {xiarui, menson}@smail.nju.edu.cn
r.benbasat@cs.ucl.ac.uk, ruili@dgut.edu.cn, gchen@nju.edu.cn
[1]Nanjing University, [2]University College London, [3]Dongguan University of Technology

ABSTRACT

Finding special items, like heavy hitters, top-k items, and persistent items, has always been a hot issue in data stream processing. While data streams nowadays are usually high-dimensional, most prior works focus on special items according to a certain primary dimension and yield little insight into the correlations between dimensions. Therefore, we propose to find special quadratic elements in data streams to reveal the close correlations between the primary and secondary dimensions. Here, both the primary and secondary dimensions are selected according to specific mining purposes. Based on the special items mentioned above, we extend our problem to three applications related to heavy hitters, top-k, and persistent items, and design a generic framework **DUET** to process them. Besides, we analyze the error bound of our algorithm theoretically and conduct extensive experiments on four publicly available data sets. Our experimental results show that DUET can achieve 3.5 higher throughput and three orders of magnitude lower average relative error compared with prior algorithms.

PVLDB Reference Format:
Jiaqian Liu [1], Haipeng Dai[1], Rui Xia[1], Meng Li[1], Ran Ben Basat[2], Rui Li[3], Guihai Chen[1]. DUET: A Generic Framework for Finding Special Quadratic Elements in Data Streams. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/callitwhatyouwannt13/DUET_Code.

1 INTRODUCTION

1.1 Background and Motivation

Nowadays, data stream processing has been widely deployed in the field of traffic management, performance diagnosis, intrusion detection, and popularity analysis [1–15]. One of the most fundamental tasks in stream processing is to mine or find special items with features such as frequency. Such items typically include *heavy hitters*, *top-k items*, and *persistent items* [16–26]. Here, heavy hitters (also called frequent items) refer to items with frequencies larger than a predefined threshold; top-k items are the k items with the largest frequencies; persistent items are defined to be the items that are widely spread in the time dimension. However, one hidden assumption taken by these works [16–26] is that they identify an item according to a primary data dimension. Considering that data nowadays are usually high-dimensional and the close correlation between different dimensions may even reveal structural or semantic information, finding special items in terms of not only the primary dimension but also the secondary dimension may be of great application value. Here, both the primary and secondary dimensions are selected according to special mining purposes. For example, the data packet in networks is typically defined as a 5-dimension tuple <*SrcIP, DstIP, SrcPort, DstPort, Protocol*> [27], which contains necessary information for applications like congestion control by locating heavy hitters in a primary dimension <*DstIP*>. However, the main purpose of congestion control is not only to locate but also to relieve congestion, and the latter requires us to also know the *SrcIPs* that send a large fraction of traffic to the heavy-hitter *DstIPs*. We will give more examples of practical applications after formally defining our problem. In other words, we not only need to find special items in the primary dimension, but also need to reveal the close correlations between the primary and secondary dimensions. Therefore, we propose to *find special quadratic elements* in data streams.

Before defining our problem, we first introduce two concepts: *item*, and *quadratic element*. By denoting two concrete values of the primary and secondary dimensions as x and y, respectively, we say that $(x \triangleleft y)$ is a *quadratic element* of an *item* x.

Definition 1.1 (Finding Special Quadratic Elements). Given a data stream of quadratic elements, find the special $(x \triangleleft y)$ that satisfy the following conditions:

$$x \in \mathcal{G}, \tag{1}$$

$$Fun((x \triangleleft y)) \geq \phi_1 Fun(x), \tag{2}$$

where $\mathcal{G}$ represents the set of special items according to the primary dimension, $Fun(.)$ is an evaluation function, and $0 < \phi_1 < 1$ is a user-defined parameter.

Finding special quadratic elements has many practical applications. Based on the typical special items mentioned above, we mainly focus on three important applications shown in Table 1 and briefly introduce them as follows.

The first application is **HH**, where $\mathcal{G}$ represents the set of heavy hitters, and $Fun(.)$ is a frequency evaluation function. Take traffic management [4] as an example, we are interested not only in the *SrcIP* that sends a large volume of traffic, but also in where most of its traffic is going. In this scenario, the chosen primary and the secondary dimensions are <*SrcIP*> and <*DstIP*>, respectively. Besides, the congestion control scenario mentioned above is also an example of HH, and it abstracts the data as $(DstIP \triangleleft SrcIP)$.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Table 1: Settings for three applications.

Applications	$\mathcal{G}$	$Fun(.)$
HH	Heavy hitters	Frequency evaluation function
TH	Top-k items	Frequency evaluation function
PP	Persistent items	Persistence evaluation function

The second application is **TH**. Here, $\mathcal{G}$ represents the set of top-k items, and $Fun(.)$ is a frequency evaluation function. In social websites, real-time trends track top-k hottest topics searched by users. By analyzing the characteristics of active users behind different hot topics, such as the age or occupation of users, websites can make more accurate recommendations and provide a higher quality of service for users. Correspondingly, the quadratic elements in streams are abstracted as (*Topic* ◃ *Age*) or (*Topic* ◃ *Occupation*).

Different from the previous two applications related to frequency, **PP** focuses on persistence. Here, $\mathcal{G}$ represents the set of persistent items, and $Fun(.)$ is a persistence evaluation function. In some stealthy attacks [28–31], victims are suffered from threats spread over multiple attackers and time periods, and need to be located by identifying persistent items. To further intercept the malicious traffic, we also need to locate attackers for each victim. Considering that locating all the attackers is of high overhead and most of the malicious traffic comes from active attackers that have long-term connections with the victim, we locate those active attackers for victims. Here, the quadratic elements in streams are abstracted as (*DstIP* ◃ *SrcIP*).

1.2 Limitations of Prior Art

Finding correlated heavy hitters (CHH) [32–34] is the most relevant problem to ours, and it is the same as our HH application. Existing algorithms for finding CHH can be divided into two types. The first type of algorithm [32, 33] uses two independent structures to separately track frequent items and frequent quadratic elements. However, the step of tracking frequent quadratic elements adds one more condition to our problem, which causes that CHH is a subset of our special quadratic element. Therefore, directly using this type of algorithm to find special quadratic elements will lead to low accuracy. The second type of algorithm [34, 35] uses a related two-tire structure to maintain quadratic elements for frequent items, and it avoids the problem in the first type of algorithm. However, the hierarchical update operations of these algorithms result in a slow processing speed. Therefore, existing algorithms are inefficient for our HH application. Moreover, for the other two applications TH and PP, they have never been considered in prior works.

1.3 Our Proposed Approach

We design a generic framework DUET to find special quadratic elements in data streams. As is shown in Fig. 1, **DUET** includes two structures: **DU**al filter and shar**E**d **T**able. In particular, dual filter (abbreviated as DFilter) consists of two parts. One is a sketch for recording and querying items. The other is a filter that consists of $d * w$ buckets divided into two parts, *Element*, and *Count*. Shared table (abbreviated as STable) consists of $l * r$ cells divided into two parts, *Element* and *Frequency*. As we all know, the size distribution of items in data streams is skewed [36–38]: most of the items called *cold items* have low frequencies and a few items called *hot items* have high frequencies. Considering that our problem focuses on the quadratic elements of hot items in $\mathcal{G}$, our key idea is to filter

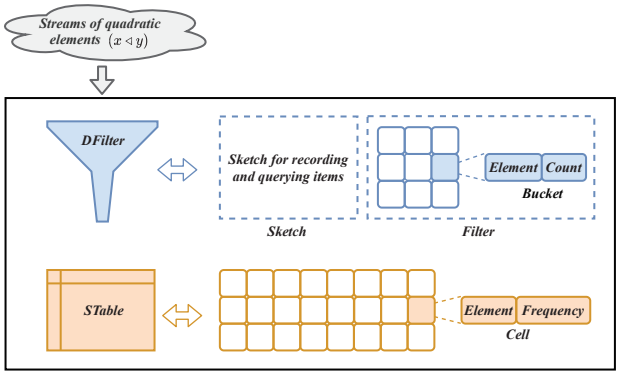


Figure 1: Our framework DUET and data structures of DFilter and STable.

out the useless quadratic elements of cold items and catch more quadratic elements of hot items, which requires us to address three technical challenges.

Our first challenge is to identify quadratic elements of cold and hot items in data streams without any prior knowledge of data. To deal with this challenge, we use the sketch in DFilter to estimate the frequency of the item for each incoming quadratic element, and check whether the item is hot or not by a predefined threshold. Then, obviously, a naive solution is to record the current quadratic element if the item is hot, or simply discard the quadratic element if it is not. However, hot items go through a cumulative process from cold to hot; during this process, their quadratic elements will be filtered out mistakenly, and some of them called *potential hot quadratic elements* are likely to be the special quadratic elements we need to find. So, our second challenge is to preserve these quadratic elements during filtering. To tackle the second challenge, we add a filter in our DFilter to track these potential hot quadratic elements. After that, our third challenge is to maintain quadratic elements of hot items efficiently. Considering the inefficiency of allocating independent space for each item to fully record its quadratic elements, our STable approximately tracks quadratic elements by sharing, *i.e.,* the items mapped into the same row of STable share all the r cells in the row to record their quadratic elements.

Based on the former solutions to the three challenges, we propose our insertion algorithm as follows. For each incoming $(x \triangleleft y)$, we record and query its item x in the sketch, and decide the operation on $(x \triangleleft y)$. If x is cold, we insert $(x \triangleleft y)$ into the filter to preserve potential hot quadratic elements. If x just becomes hot, we extract its potential hot quadratic elements from the filter to STable. If x is already hot, we insert $(x \triangleleft y)$ into STable. For the query process, we simply traverse each quadratic element recorded in STable and query its item in the sketch to obtain the special quadratic elements.

As indicated by our theoretical analysis and experimental results, DUET has the following three advantages. First, DUET reaches a high processing speed. Experimental results show that the throughput of DUET can be 3.5 times higher than that of prior algorithms. Second, DUET achieves a high estimation accuracy. It can reduce the average relative error by three orders of magnitude compared with prior algorithms in our experiments. Third, DUET is generic to be applied to all the three applications mentioned above. Especially, to extend our solution to PP, we propose a method equivalently

convert persistence to frequency after removing the duplicates in each of the T equal-sized time periods divided for measurement. We will give the detailed design for each application in Section 5, and conduct extensive experiments in Section 6 for them.

1.4 Our Contributions

Our key contributions are summarized below.

(1) To the best of our knowledge, we are the first to propose the problem of finding special quadratic elements in data streams, which can be applied into multiple practical applications.

(2) We propose a framework DUET, which is generic, accurate, and has a high processing speed, and show how to apply it to three practical applications, including HH, TH, and PP.

(3) We theoretically analyze the space and time complexities of our algorithm, and obtain the error bound of DUET.

(4) We conduct extensive experiments on four publicly available data sets to evaluate the performance of DUET. Our experimental results show that DUET can achieve 3.5 higher throughput and three orders of magnitude lower average relative error compared with prior algorithms.

2 RELATED WORK

In this section, we introduce the related work in terms of our three applications, HH, TH, and PP.

2.1 HH

There are two problems related to this application, *i.e.,* finding correlated heavy hitters (CHH) [32–34], and finding popular conditional heavy hitters (PCHH) [39].

The definition of CHH is the same as that of our special quadratic element in HH. Existing algorithms to find CHH can be classified into two types. The first type uses two independent structures to record frequent items and frequent quadratic elements separately, and obtain CHHs by checking these frequent quadratic elements. Typical algorithms include CSSCHH [32], which separately maintains items and quadratic elements by using Space-Saving [20], and PCSSCHH [33], which is essentially a parallel version of CSSCHH on distributed and shared-memory architectures. In fact, CHHs found by these algorithms need to satisfy more strict conditions than our special quadratic elements, for that they first need to be frequent enough to survive at the recording step. Therefore, this type of algorithm essentially limits the estimation accuracy. The second type uses a related two-tier structure to record items and quadratic elements, including one typical algorithm MGCHH [34]. In particular, MGCHH uses a set of primary counters updated by the Misra-Gries (MG) [35] algorithm to track frequent items; moreover, another set of secondary counters are associated with each primary counter to track quadratic elements of the primary item. Because the cardinality of items (quadratic elements) is much larger than the number of primary (secondary) counters, MGCHH needs to perform complex hierarchical operations to evict items (quadratic elements) during the update. Therefore, the processing speed of MGCHH is very slow.

PCHH is slightly different from our special quadratic element. That is, PCHH refers to $(x \triangleleft y)$ that satisfies $f_{x \triangleleft y} \geq \phi_1 f_x$, where f_x and $f_{x \triangleleft y}$ are the frequencies of x and $(x \triangleleft y)$, respectively, and $f_{x \triangleleft y}$ ranks in top-k with respect to all quadratic elements. However, two existing algorithms of finding PCHH, FamilyHH and GlobalHH [39], can also be adapted to process HH. Specifically, the key idea of FamilyHH and GlobalHH is similar to CSSCHH, *i.e.,* using two independent structures to record frequent items and quadratic elements separately. Thus, these two algorithms suffer the same problem as the first type of algorithm for finding CHH and have poor accuracy when the memory is tight.

2.2 TH

To the best of our knowledge, no prior works deal with this application TH. Specifically, prior works usually focus on top-k items in a primary dimension and ignore their quadratic elements. Therefore, we just introduce related algorithms for finding top-k items in this part.

The algorithms for finding top-k items can be divided into two categories. One is sketch-based, which approximately estimates the frequency of all items, and uses a min-heap to track top-k items. Classic sketches include CM sketch [17], C sketch [18], and CU sketch [19]. The other is counter-based, which maintains a part of items with large frequencies, and typical algorithms include Lossy-Counting [21], Space-Saving [20], and HeavyGuardian [22].

2.3 PP

Similar to the situation of TH, prior works usually focus on persistent items and ignore their quadratic elements. Thus, we just introduce related algorithms for finding persistent items.

The core idea of finding persistent items is to remove the duplicates in each time period, and two categories of algorithms exist. One is to record the original ID of items. Typical algorithms include small space [25] based on sampling and On-Off sketch [23] based on setting flag bits. The other is to record the code of items, including a typical algorithm PIE [24], which records the fingerprint and raptor codes [40] to decode the persistent items.

Table 2: Notations frequently used in this paper.

Notations	Descriptions
x	Item x
$(x \triangleleft y)$	Quadratic element $(x \triangleleft y)$
d	Number of arrays in the filter
w	Number of buckets in each array of the filter
N_{th}	Threshold of hot items
$B[i][j]$	The j-th bucket in the i-th array of the filter
$B[i][j].E$ $(*.C)$	*Element* (*Count*) part of $B[i][j]$
l	Number of rows in STable
r	Number of cells in each row of STable
$C[i][j]$	The j-th cell in the i-th row of STable
$C[i][j].E$ $(*.F)$	*Element* (*Frequency*) part of $C[i][j]$
f_x $(\widehat{f_x})$	(Estimated) frequency of x
$f_{x \triangleleft y}$ $(\widehat{f_{x \triangleleft y}})$	(Estimated) frequency of $(x \triangleleft y)$
p_x $(\widehat{p_x})$	(Estimated) persistence of x
$p_{x \triangleleft y}$ $(\widehat{p_{x \triangleleft y}})$	(Estimated) persistence of $(x \triangleleft y)$
$h_i(.)$ $(1 \leq i \leq d)$	Hash functions from items to $\{1, \ldots, w\}$
$H_f(.)$	Hash function from quadratic element to $\{1, \ldots, d\}$
$H_t(.)$	Hash function from items to $\{1, \ldots, l\}$
ϕ_1, ϕ_2	User-defined parameters, $0 < \phi_1, \phi_2 < 1$
N	Size of stream
T	Number of time periods

3 DESIGN OF DUET

In this section, we describe the design of DUET. We first introduce our solution framework in Section 3.1. Then, we describe the insertion and query operations in Section 3.2 and 3.3, respectively.

For convenience, we list the notations frequently used in this paper and their descriptions in Table 2.

3.1 Solution Framework

As is shown in Fig. 1, DUET consists of two structures: dual filter (DFilter), and shared table (STable). In this part, we introduce the design of these two structures in detail.

DFilter: The DFilter has two key functions: 1) Estimate the frequency of items and identify hot items by a predefined threshold N_{th}; and 2) Preserve the potential hot quadratic elements.

Concretely, the DFilter consists of two parts. One is a sketch for recording and querying items, and typical sketches include CM sketch, CU sketch, and Stream-Summary. The other is a filter that consists of d arrays, each of which is associated with a uniform random hash function [41] $h_i(.) \in [1, w]$ $(1 \le i \le d)$. For each array, there are w buckets and each of the buckets is divided into two parts: 1) *Element*, which records the identifier of the quadratic element mapped into the bucket using a uniform random hash function $H_f(.) \in [1, d]$; 2) *Count*, which is a counter to count for the quadratic element recorded in *Element*. For convenience, let $B[i][j]$ $(1 \le i \le d, 1 \le j \le w)$ denote the j-th bucket in the i-th array, and let $B[i][j].E$ and $B[i][j].C$ denote the *Element* part and the *Count* part in the bucket $B[i][j]$, respectively.

STable: Considering the inefficiency of allocating independent space for each item to fully record its quadratic elements, the key idea of our STable is to share space among items to approximately track their quadratic elements.

Specifically, the STable consists of a $l * r$ matrix of cells, each of which contains two parts *Element* and *Frequency* to record the quadratic element and its frequency, respectively. For convenience, let $C[i][j]$ $(1 \le i \le l, 1 \le j \le r)$ denote the cell in row i and column j, and let $C[i][j].E$ and $C[i][j].F$ denote the *Element* part and the *Frequency* part in the cell $C[i][j]$, respectively. Moreover, we assign a uniform random hash function $H_t(.) \in [1, l]$, and let items with the same value $l_x = H_t(x)$ share all the r cells in the l_x-th row to record their quadratic elements.

3.2 Insertion

As is shown in Algorithm 1, we firstly describe the main idea of the insertion. For each incoming $(x \triangleleft y)$, we first query x in the sketch (Line 1). If $\widehat{f_x} < N_{th}$, which means x is cold, we insert x and $(x \triangleleft y)$ into the sketch and the filter, respectively (Line 2-4). After that, if $\widehat{f_x} + 1 == N_{th}$, which means x just becomes hot, we extract its *potential hot quadratic elements* from d mapped buckets of x to STable, and reinitialize their *Element* and *Count* parts (Line 5-14). If $\widehat{f_x} \ge N_{th}$, which means x is already a hot item, we insert x into the sketch and insert $(x \triangleleft y)$ into STable (Line 15-17). Next, we introduce all the functions invoked in Algorithm 1.

Insert2Sketch(x) and QuerySketch(x): Both of these functions are invoked by the sketch in DFilter. Take the CM sketch as an example, it consists of D arrays, $A_1, ..., A_D$, each of which contains W counters. For Insert2Sketch(x), it uses D hash functions, $H_1, ..., H_D$, to map x to counters $A_1[H_1(x)\%W], ..., A_D[H_D(x)\%W]$, and increases them by 1. Before inserting an item x, we call the function QuerySketch(x), which also maps x to D counters and returns the smallest value of the D mapped counters.

Algorithm 1: Insert($(x \triangleleft y)$)

Input: $(x \triangleleft y)$; Threshold N_{th}; Hash functions $h_i(.)$ $(1 \le i \le d)$, $H_f(.)$, and $H_t(.)$.

```
1  f̂_x = QuerySketch(x);
2  if f̂_x < N_th then
3      Insert2Sketch(x);
4      Insert2Filter((x ◁ y));
5      if f̂_x + 1 == N_th then
6          for i ∈ [1, d] do
7              (x_i ◁ y_i) = B[i][h_i(x)].E;
8              if x_i == x then
9                  Insert2Table((x_i ◁ y_i), B[i][h_i(x)].C);
10                 B[i][h_i(x)].E = NULL;
11                 B[i][h_i(x)].C = 0;
12 else
13     Insert2Sketch(x);
14     Insert2Table((x ◁ y), 1);
```

Algorithm 2: Insert2Filter $((x \triangleleft y))$

Input: $(x \triangleleft y)$; Hash functions $h_i(.)$ $(1 \le i \le d)$ and $H_f(.)$.

```
1  p = H_f(y);
2  if B[p][h_p(x)].E == NULL then
3      B[p][h_p(x)].E = (x ◁ y);
4      B[p][h_p(x)].C = 1;
5  else if B[p][h_p(x)].E == (x ◁ y) then
6      B[p][h_p(x)].C + +;
7  else
8      B[p][h_p(x)].C − −;
9      if B[p][h_p(x)].C == 0 then
10         B[p][h_p(x)].E = (x ◁ y);
11         B[p][h_p(x)].C = 1;
```

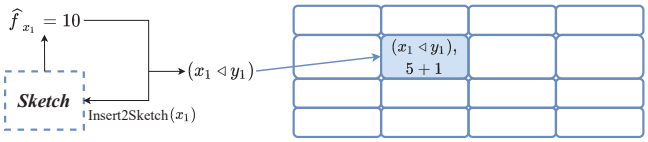
Insert2Filter($(x \triangleleft y)$): This function operates on *Element* and *Count* parts of the filter to preserve potential hot quadratic elements. The rationale is that each item uses its d mapped buckets, $B[1][h_1(x)], ..., B[d][h_d(x)]$, to approximately preserve its quadratic elements. As is shown in Algorithm 2, we first calculate the hash value $p = H_f(y)$ and insert $(x \triangleleft y)$ into the bucket $B[p][h_p(x)]$. There are three cases:

(1) $B[p][h_p(x)].E$ is NULL. We directly insert $(x \triangleleft y)$ by setting $B[p][h_p(x)].E = (x \triangleleft y)$ and $B[p][h_p(x)].C = 1$ (Line 2-4).

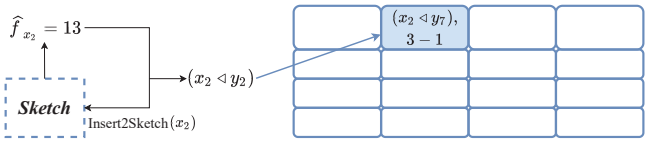
(2) The quadratic element recorded in $B[p][h_p(x)].E$ is $(x \triangleleft y)$. We increase $B[p][h_p(x)].C$ by 1 (Line 5-6).

(3) Otherwise, we decrease $B[p][h_p(x)].C$ by 1. After that, if $B[p][h_p(x)].C == 0$, we replace $B[p][h_p(x)].E$ with $(x \triangleleft y)$, and set $B[p][h_p(x)].C = 1$ (Line 7-11).

Example 1 (Fig. 2): We use an example to illustrate the operation in DFilter, where $d = 4$, $w = 4$, and $N_{th} = 16$. When $(x_1 \triangleleft y_1)$ is coming, we first query x_1 in the sketch and $\widehat{f_{x_1}} = 10 < 16$. Then, x_1 is inserted into the sketch, and $(x_1 \triangleleft y_1)$ is inserted into the filter. Here, $p = H_f(y_1) = 2$ and $h_p(x_1) = 2$. Thus, $(x_1 \triangleleft y_1)$ is mapped to

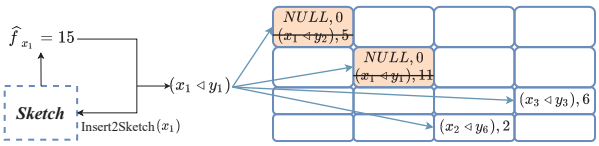


(a) Insert $((x_1 \triangleleft y_1))$.

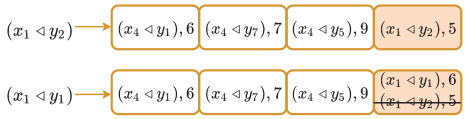


(b) Insert $((x_2 \triangleleft y_2))$.

Figure 2: Example of inserting $(x_1 \triangleleft y_1)$ and $(x_2 \triangleleft y_2)$ to DFilter when x_1 and x_2 are cold.



(a) Step 1: Extract two potential hot elements of x_1 and reset *Element* and *Count* of their buckets.



(b) Step 2: Insert $(x_1 \triangleleft y_2)$ and $(x_1 \triangleleft y_1)$ to STable.

Figure 3: Example of extracting potential hot quadratic elements to STable when x_1 just becomes hot.

Algorithm 3: Insert2Table$((x \triangleleft y), f)$

Input: Tuple $((x \triangleleft y), f)$; Hash function $H_t(.)$.

```
1  l_x = H_t(x);
2  if (x ◁ y) has been in some cell C[l_x][j] then
3      C[l_x][j].F += f;
4  else if There are some empty cells then
5      Choose one empty cell C[l_x][col];
6      C[l_x][col].E = (x ◁ y);
7      C[l_x][col].F = f;
8  else
9      Get the cell C[l_x][min] with the minimum Frequency;
10     C[l_x][min].F -= f;
11     if C[l_x][min].F < 0 then
12         C[l_x][min].E = (x ◁ y);
13         C[l_x][min].F = -C[l_x][min].F;
```

$B[2][2]$. Because $B[2][2].E = (x_1 \triangleleft y_1)$, we increase $B[2][2].C$ by 1. After that, $\widehat{f}_{x_1} = 10 + 1 < 16$, so the insertion procedure completes.

For the next incoming $(x_2 \triangleleft y_2)$, $\widehat{f}_{x_2} = 13 < 16$. Then, x_2 is inserted into the sketch, and $(x_2 \triangleleft y_2)$ is inserted into the filter. Here, $p = H_f(y_2) = 1$ and $h_p(x_2) = 2$. Thus, $(x_2 \triangleleft y_2)$ is mapped to $B[1][2]$. Because $B[1][2].E \neq (x_2 \triangleleft y_2)$, we decrease $B[1][2].C$ by 1. After that, $\widehat{f}_{x_2} = 13 + 1 < 16$, so the insertion procedure completes.

Insert2Table$((x \triangleleft y), f)$: This function is invoked by STable, which makes the items with the same hash value share space to record their quadratic elements. As is shown in Algorithm 3, we first calculate the hash value $l_x = H_t(x)$ and insert $(x \triangleleft y)$ into the l_x-th row. There are three cases of the insertion:

(1) $(x \triangleleft y)$ is already in one cell of row l_x. We simply increase $C[l_x][j].F$ by f (Line 2-3).

(2) $(x \triangleleft y)$ is not found and there are some empty cells in row l_x. We choose one empty cell to insert $(x \triangleleft y)$ and set its $Frequency$ as f (Line 4-7).

(3) Otherwise, we find the cell $C[l_x][min]$ with the minimum $Frequency$ in row l_x and decrease $C[l_x][min].F$ by f. After that, if $C[l_x][min].F < 0$, we replace $C[l_x][min].E$ with $(x \triangleleft y)$ and set $C[l_x][min].F$ as its absolute value (Line 9-13).

Algorithm 4: Query

Input: The filter and STable; The conditional statement; $\Theta = \varnothing$.
Output: The set of special quadratic elements Θ.

```
1  for i in [1, l] do
2      for j in [1, r] do
3          (x ◁ y) = C[i][j].E;
4          f̂_{x◁y} = C[i][j].F;
5          f̂_x = QuerySketch(x);
6          if The conditional statement is true then
7              Θ = Θ ∪ (x ◁ y);
8  return Θ;
```

Example 2 (Fig. 3): In this example, we show how to extract potential hot quadratic elements to STable when an item just becomes hot. We continue to set $N_{th} = 16$, and let $r = 4$ for STable. After the insertion of $(x_1 \triangleleft y_1)$ in DFilter, $\widehat{f}_{x_1} = 15 + 1 = 16$, which means x_1 just becomes hot, and there are two potential hot quadratic elements, $(x_1 \triangleleft y_2)$ and $(x_1 \triangleleft y_1)$, in d mapped buckets of x_1. As is shown in Fig. 3, the insertion process includes two steps: 1) Extract the potential hot quadratic elements, and reset their buckets; 2) We insert $(x_1 \triangleleft y_2)$ into STable first. It is mapped into the $H_t(x_1)$-th row, where just the last cell is empty. Thus, we insert $(x_1 \triangleleft y_2)$ into the last cell and set $Frequency = 5$; For $(x_1 \triangleleft y_1)$, it is also mapped into the $H_t(x_1)$-th row and there is no empty cell left. We replace the smallest cell with $(x_1 \triangleleft y_1)$, and set $Frequency = |5 - 11| = 6$.

3.3 Query

We invoke the function Query (shown in Algorithm 4) to obtain the special quadratic elements. Before starting the query process, we set the appropriate conditional statement for different applications. Take HH as an example, the statement is set as $\widehat{f}_{x \triangleleft y} \geq \phi_1 \widehat{f}_x$ *and* $\widehat{f}_x \geq \phi_2 N$. After that, the algorithm traverses the whole STable to get the quadratic element in each cell and query its item in the sketch. In the end, the algorithm returns a set of quadratic elements Θ that meet the condition. We will give the detailed settings for HH, TH, and PP in Section 5.

4 THEORETICAL ANALYSIS OF DUET

In this section, we give the theoretical analysis of DUET. We first give the space and time complexities of DUET. Then, we analyze the estimation error bound of quadratic elements. Considering that the sketch in DFilter is optional, we first give the error-bound model of quadratic elements. Then, we use the CM sketch as an example to obtain the specific error bound.

4.1 Space and Time Complexities

Space complexity: considering that DUET consists of a sketch, $d * w$ buckets, and $l * r$ cells, the space complexity is $O(s + (dw + lr)\log n)$, where s is the space complexity of the sketch.

Time complexity: for the insertion process, the most complicated insertion involves one query in the sketch, one insertion in the filter, and the operation to extract potential hot elements from the filter to STable. Therefore, the worst time complexity of the insertion is $O(t + dr)$, where t is the time complexity of the query in the sketch. For the query process, we simply traverse STable and check the filter. Thus, the time complexity of the query is $O(lrt)$.

4.2 No Over-estimation of Quadratic Elements

Lemma 4.1. *During the whole updating process, there is no over-estimation of quadratic elements. That is, denote the estimated and true frequency of quadratic element $(x_i \triangleleft y_i)$ on the u-th update as $\widehat{f}^{(u)}_{x_i \triangleleft y_i}$ and $f^{(u)}_{x_i \triangleleft y_i}$, respectively. We have*

$$\widehat{f}^{(u)}_{x_i \triangleleft y_i} \leq f^{(u)}_{x_i \triangleleft y_i}. \tag{3}$$

Proof. Generally, we complete the proof by mathematical induction. Due to space limitations, we omit the proof here. □

4.3 Error-bound Model of Quadratic Elements

As per Lemma 4.1, there is no over-estimation of quadratic elements, so we turn to derive the error bound of under-estimation. Considering that the sketch in DFilter is optional, in our error bound model, we abstract the expected number of quadratic elements inserted into DFilter and STable as $\mathcal{F}_d$ and $\mathcal{F}_s$, respectively, which can be derived according to the sketch used in DFilter. To pave the way towards our model derived in Theorem 4.3, we first give the expectation of under-estimation in DFilter in the following lemma.

Lemma 4.2. *Denote the expected number of quadratic elements inserted into DFilter as $\mathcal{F}_d$, the threshold of hot items as N_{th},and the true frequency of x_i as f_{x_i}. Then, the expectation of under-estimation in DFilter for each $(x_i \triangleleft y_i)$ is*

$$\mathbb{E}(D_i) = \frac{\mathcal{F}_d}{dw} + \frac{min\{N_{th}, f_{x_i}\}}{d}. \tag{4}$$

Proof. Assume that $(x_i \triangleleft y_i)$ is recorded in $B[d_i][h_{d_i}(x_i)]$ and the new incoming quadratic element is $(x_j \triangleleft y_j)$. If there is a hash collision between them, *i.e.,* $(x_i \triangleleft y_i) \neq (x_j \triangleleft y_j)$ and $(x_j \triangleleft y_j)$ is also mapped into $B[d_i][h_{d_i}(x_i)]$, the under-estimation of $(x_i \triangleleft y_i)$ occurs. Specifically, there are two situations that cause $(x_i \triangleleft y_i) \neq (x_j \triangleleft y_j)$. One is $x_i \neq x_j$ and the other is $(x_i = x_j) \wedge (y_i \neq y_j)$. For the first situation, let $I_{i,j}$ be an indicator function defined as

$$I_{i,j} = \begin{cases} 1, & x_i \neq x_j \wedge h_i(x_i) = h_i(x_j) \wedge H_f(y_i) = H_f(y_j), \\ 0, & otherwise. \end{cases} \tag{5}$$

and X_i denote the number of quadratic elements that cause hash collisions in the first situation, and it is clear that

$$\mathbb{E}(X_i) \leq \mathbb{E}\left(\sum_{j=1}^{\mathcal{F}_d} I_{i,j}\right) \leq \sum_{j=1}^{\mathcal{F}_d} \mathbb{E}(I_{i,j}) \leq \frac{\mathcal{F}_d}{dw}. \tag{6}$$

Analogously, for the second situation, let $L_{i,j}$ be an indicator function, defined as

$$L_{i,j} = \begin{cases} 1, & x_i = x_j \wedge y_i \neq y_j \wedge H_f(y_i) = H_f(y_j), \\ 0, & otherwise, \end{cases} \tag{7}$$

and Y_i denote the number of quadratic elements that cause hash collisions in the second situation. For hot items, N_{th} out of f_{x_i} quadratic elements are inserted into DFilter at most; and for cold items, all f_{x_i} quadratic elements are inserted at most. Therefore, let $N_i = min\{N_{th}, f_{x_i}\}$.

$$\mathbb{E}(Y_i) \leq \mathbb{E}\left(\sum_{j=1}^{N_i} L_{i,j}\right) \leq \sum_{j=1}^{N_i} \mathbb{E}(L_{i,j}) \leq \frac{N_i}{d}. \tag{8}$$

By adding $\mathbb{E}(X_i)$ and $\mathbb{E}(Y_i)$, we have

$$\mathbb{E}(D_i) \leq \frac{\mathcal{F}_d}{dw} + \frac{min\{N_{th}, f_{x_i}\}}{d}. \tag{9}$$

This completes the proof. □

Theorem 4.3. *Let $f_{x_i \triangleleft y_i}$ and $\widehat{f}_{x_i \triangleleft y_i}$ denote the true and estimated frequency of $(x_i \triangleleft y_i)$, respectively. Let $\mathcal{F}_d$ and $\mathcal{F}_s$ denote the number of quadratic elements inserted into DFilter and STable, respectively. Let N, N_{th} and f_{x_i} denote the stream size, the threshold of hot items, and the true frequency of x_i, respectively. Let $\mathcal{L}_i$ denote the set of quadratic elements whose frequencies are larger than $(x_i \triangleleft y_i)$. Given an arbitrary positive value ε, and let $\Delta_i = f_{x_i \triangleleft y_i} - \widehat{f}_{x_i \triangleleft y_i}$, we have*

$$Pr[\Delta_i \geq \varepsilon N] \leq \frac{\mathcal{F}_d + w * min\{N_{th}, f_{x_i}\}}{\varepsilon dwN} + \frac{\mathcal{F}_s * P_{dec}}{\varepsilon lN}, \tag{10}$$

$$where,\ P_{dec} = \binom{|\mathcal{L}_i|}{r-1} \left(\frac{1}{l}\right)^{r-1} \left(1 - \frac{1}{l}\right)^{|\mathcal{L}_i|-r+1}.$$

Proof. Firstly, we analyze the expectation of under-estimation in STable for $(x_i \triangleleft y_i)$, denoted by $\mathbb{E}(S_i)$. For $(x_i \triangleleft y_i)$ which is recorded in $C[l_i][r_i]$, it is decreased only when there is no empty cell for the new incoming quadratic element that is also hashed into the l_i-th row, and $C[l_i][r_i].F$ is the minimum in the l_i-th row. Let P_{dec} denote the probability of performing the decrement, and $\mathcal{L}_i$ denote the set of quadratic elements whose frequencies are larger than $(x_i \triangleleft y_i)$, we have

$$P_{dec} = \binom{|\mathcal{L}_i|}{r-1} \left(\frac{1}{l}\right)^{r-1} \left(1 - \frac{1}{l}\right)^{|\mathcal{L}_i|-r+1}. \tag{11}$$

Thus, the expectation of under-estimation in STable for each $(x_i \triangleleft y_i)$ is given by

$$\mathbb{E}(S_i) \leq \frac{\mathcal{F}_s * P_{dec}}{l}. \tag{12}$$

Next, we analyze the error bound for $(x_i \triangleleft y_i)$. Specifically, we divide all the quadratic elements into three sets: 1) $\mathcal{A}$, which is only

inserted into DFilter; 2) $\mathcal{B}$, which is only inserted into STable; 3) C, which is inserted into both DFilter and STable. It is clear that

$$\mathbb{E}(\widehat{f}_{x_i \triangleleft y_i}) = \begin{cases} f_{x_i \triangleleft y_i} - \mathbb{E}(D_i), & (x_i \triangleleft y_i) \in \mathcal{A}, \\ f_{x_i \triangleleft y_i} - \mathbb{E}(S_i), & (x_i \triangleleft y_i) \in \mathcal{B}, \\ f_{x_i \triangleleft y_i} - \mathbb{E}(D_i) - \mathbb{E}(S_i), & (x_i \triangleleft y_i) \in C. \end{cases} \tag{13}$$

Then, based on the Markov inequality,

$$Pr[\Delta_i \geq \varepsilon N] \leq \frac{\mathbb{E}(\Delta_i)}{\varepsilon N} \leq \begin{cases} \dfrac{\mathbb{E}(D_i)}{\varepsilon N}, & (x_i \triangleleft y_i) \in \mathcal{A}, \\ \dfrac{\mathbb{E}(S_i)}{\varepsilon N}, & (x_i \triangleleft y_i) \in \mathcal{B}, \\ \dfrac{\mathbb{E}(D_i) + \mathbb{E}(S_i)}{\varepsilon N}, & (x_i \triangleleft y_i) \in C. \end{cases} \tag{14}$$

Thus, for any $(x_i \triangleleft y_i)$, we have

$$Pr[\Delta_i \geq \varepsilon N] \leq \frac{\mathbb{E}(D_i) + \mathbb{E}(S_i)}{\varepsilon N}. \tag{15}$$

By combining Eq. (4) derived in Lemma 4.2, Eq. (11), Eq. (12), and Eq. (15), the result follows. □

4.4 Error Bound Based on CM Sketch

Based on the error-bound model derived in Theorem 4.3, we need to obtain $\mathcal{F}_d$ and $\mathcal{F}_s$ to get the error bound based on CM sketch. Before that, we first analyze the estimation error of items.

Specifically, CM sketch overestimates the frequency of items, for that some different items may be mapped into the same counter. Therefore, some cold items are identified as hot items mistakenly and further affect the estimation of quadratic elements. By the similar analysis of CM sketch [17], we obtain the following lemma.

Lemma 4.4. *Suppose that CM sketch consists of $D * W$ counters. Let $\widehat{f}_{x_i}$ and f_{x_i} denote the estimated and true frequency of x_i in DFilter, respectively, N denote the total frequency of all items, and ε_1 denote an arbitrary positive value. Then,*

$$Pr[\widehat{f}_{x_i} - f_{x_i} \geq \varepsilon_1 N] \leq (\varepsilon_1 W)^{-D}, \tag{16}$$

Proof. Due to space limitations, we omit the proof here. □

Lemma 4.5. *Let n denote the number of distinct items in the stream, and $\mathcal{F}_d$ denote the expected number of quadratic elements inserted into DFilter. Suppose that all the items in the stream are arranged as $\{x_1, x_2, ..., x_n\}$, where $f_{x_1} \geq f_{x_2} \geq ... \geq f_{x_n}$, and the items before x_h (including x_h) are hot, which means $f_{x_i} \geq N_{th}$ $(1 \leq i \leq h)$ and $f_{x_i} < N_{th}$ $(h < i \leq n)$. Then,*

$$\mathcal{F}_d \leq hN_{th} + \sum_{i=h+1}^{n} f_{x_i}. \tag{17}$$

Moreover, suppose that the frequency distribution of cold items obeys the distribution $\mathcal{D}$ and the mean value of $\mathcal{D}$ is $\mu_{\mathcal{D}}$. Let N denote the total frequency of all the items, and $\mathcal{F}_s$ denote the expected number of quadratic elements inserted into STable. Then,

$$\mathcal{F}_s \leq \sum_{i=1}^{h} f_{x_i} + \mu_{\mathcal{D}} * \sum_{i=h+1}^{n} min\left\{1, \left[\frac{N}{W(N_{th} - f_{x_i})}\right]^D\right\}. \tag{18}$$

Proof. According to the insertion process, hN_{th} quadratic elements of hot items and all the $\sum_{i=h+1}^{n} f_{x_i}$ quadratic elements of cold items should be inserted into DFilter. Actually, due to hash collisions, some items are overestimated and their quadratic elements are inserted into STable when their true frequencies are less than N_{th}. Therefore, $\mathcal{F}_d \leq hN_{th} + \sum_{i=h+1}^{n} f_{x_i}$.

The quadratic elements inserted into STable can be divided into two types. The first type belongs to hot items, and let $\mathbb{E}(N_{sh})$ denote the expectation of total frequencies of them. Because some of their quadratic elements may be inserted into DFilter first and underestimated in DFilter, as per Lemma 4.1. Thus, $\mathbb{E}(N_{sh}) \leq \sum_{i=1}^{h} f_{x_i}$. The second type belongs to cold items. If the estimated frequency of a cold item x_i is larger than N_{th}, its quadratic elements will be inserted into STable, and denote the probability of it as P_i^{over}. As per Lemma 4.4, let $\varepsilon_1 N = N_{th} - f_{x_i}$, we have

$$P_i^{over} = Pr[\widehat{f}_{x_i} \geq N_{th}] \leq min\left\{1, \left[\frac{N}{W(N_{th} - f_{x_i})}\right]^D\right\}. \tag{19}$$

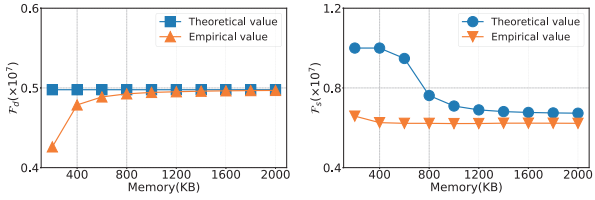
Let U_c be the set of all the cold items, ζ be the set of cold items whose estimated frequencies are larger than N_{th}, and N_{sc} be the total frequency of these cold items, then the expectation of N_{sc} is

$$\begin{aligned} \mathbb{E}(N_{sc}) &\leq \sum_{x_i \in \zeta} \mathbb{E}(f_{x_i}) \leq \sum_{x_i \in U_c} P_i^{over} * \mu_{\mathcal{D}} \\ &\leq \mu_{\mathcal{D}} * \sum_{i=h+1}^{n} min\left\{1, \left[\frac{N}{W(N_{th} - f_{x_i})}\right]^D\right\}. \end{aligned} \tag{20}$$

By adding $\mathbb{E}(N_{sh})$ and $\mathbb{E}(N_{sc})$, Eq. (18) holds.

This completes the proof. □

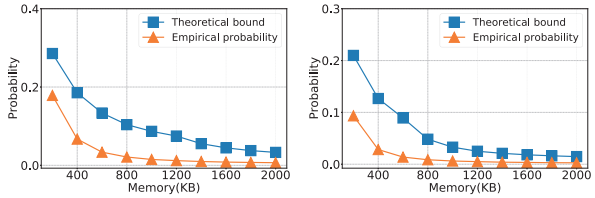
To validate the correctness of Lemma 4.5, we conduct experiments on the data set CAIDA1 (mentioned in Section 6.1). Here, let $N = 10^7$, and vary memory from 200KB to 2000KB. As is shown in Fig. 4, the empirical $\mathcal{F}_d$ and $\mathcal{F}_s$ are always lower than the theoretical $\mathcal{F}_d$ and $\mathcal{F}_s$, respectively, conforming the correctness of Lemma 4.5.



(a) $\mathcal{F}_d$. (b) $\mathcal{F}_s$.

Figure 4: Theoretical value vs. empirical value.

By plugging in $\mathcal{F}_d$ and $\mathcal{F}_s$ derived in Lemma 4.5 into Theorem 4.3, we obtain the error bound of Quadratic elements based on CM sketch. To validate that, we conduct experiments on CAIDA1. Here, we vary the memory from 200KB to 2000KB and let $N = 10^7$. In Fig. 5, we show the results when $\varepsilon = 2^{-17}$ and $\varepsilon = 2^{-16}$. It is clear that the empirical probability of DUET is always lower than the theoretical bound derived in Theorem 4.3.



(a) $\varepsilon = 2^{-17}$. (b) $\varepsilon = 2^{-16}$.

Figure 5: Theoretical bound vs. empirical probability.

5 IMPLEMENTATION FOR THREE APPLICATIONS

In this section, we briefly introduce how to apply DUET to three applications shown in Table 1, *i.e.,* HH, TH, and PP.

5.1 HH

Problem definition: Given a stream of quadratic elements, find all $(x \triangleleft y)$s that satisfy $f_x \geq \phi_2 N$ and $f_{x \triangleleft y} \geq \phi_1 f_x$.

Data structures and operations: Our framework DUET and operations in Section 3 can be directly used in this application. Before querying, the conditional statement in Algorithm 4 is set as $\widehat{f}_x \geq \phi_2 N$ *and* $\widehat{f}_{x \triangleleft y} \geq \phi_1 \widehat{f}_x$.

5.2 TH

Problem definition: Given a stream of quadratic elements, find all $(x \triangleleft y)$s that satisfy that f_x ranks in top-k and $f_{x \triangleleft y} \geq \phi_1 f_x$.

Data structures and operations: In this application, the sketch in DFilter can be any data structure to find top-k items. Typical sketches include CM (CU) sketch with a heap [26], and Stream-Summary updated by Space-Saving. Besides, the filter and STable keep unchanged. For the insertion process, we still insert items and quadratic elements as in Algorithm 1. For the query process, we replace $\widehat{f}_x$ = QuerySketch(x) in Line 5 of Algorithm 4 with $\widehat{f}_x = \mathcal{S}(x)$. Here, $\mathcal{S}(x)$ returns $\widehat{f}_x$ if x is recorded in the sketch as one of the top-k items, or returns 0 if not. Meanwhile, the conditional statement is set as $\widehat{f}_x > 0$ *and* $\widehat{f}_{x \triangleleft y} \geq \phi_1 \widehat{f}_x$.

5.3 PP

Problem definition: Given a stream of quadratic elements divided into T equal-sized time periods, find all $(x \triangleleft y)$s that satisfy $p_x \geq \phi_2 T$ and $p_{x \triangleleft y} \geq \phi_1 p_x$.

Data structures and operations: One simple way to remove duplicates is to add a Bloom Filter (BF) [42, 43] on the top of DUET. The rationale is to regard the items and quadratic elements that occur in each period as a set and encode them in BF. By membership query for each item and element, we can judge whether it appears for the first time in the current period or not. If yes, we store it into DUET and perform update operations. However, the encoding of BF is based on hashing, which introduces extra computation overhead and slows down the processing speed.

To overcome the shortcomings of BF, we use a CU sketch as the sketch in DFilter, and add a flag bit [23] to each counter of DUET to indicate whether it has been updated in each period, which also achieves the purpose of removing duplicates. At the beginning of each period, all flag bits are initialized to 0. For the insertion process, we check the related flag bit before each increment operation in Algorithm 1. If and only if the flag bit is 0, we perform the increment and reset it as 1. Before querying similar to Algorithm 4, we replace $\widehat{f}_{x \triangleleft y}$ and $\widehat{f}_x$ with $\widehat{p}_{x \triangleleft y}$ and $\widehat{p}_x$, respectively, and set the conditional statement as $\widehat{p}_x \geq \phi_2 T$ *and* $\widehat{p}_{x \triangleleft y} \geq \phi_1 \widehat{p}_x$.

6 EXPERIMENTAL RESULTS

In this section, we conduct experiments to validate the performance of DUET. We firstly introduce the experimental setup in Section 6.1. Then, we compare our algorithms based on DUET with prior algorithms on four data sets for application HH in Section 6.2. Similarly, we evaluate the performance of DUET for application TH and PP in Section 6.3 and 6.4, respectively.

6.1 Experimental Setup

Implementation: We implement our algorithm and the comparison algorithms in C++ and conduct all the experiments on a machine with 6-core processors (Intel Core i7-8700 CPU @3.20GHz). The source codes of all algorithms are available at [44].

Data Sets: We use the four data sets in our experiments as below.

(1) **CAIDA1 and CAIDA2:** We use anonymized passive traffic traces from CAIDA's passive monitors in 2019 [45], which includes traces from the 'equinix-nyc' high-speed monitor. Particularly, we choose 'equinix-nyc.dirA.20190117-125910' (named CAIDA1, which contains 10M packets, including 163,078 distinct items and 623,981 distinct quadratic elements) and 'dirB.20190117-130100' (named CAIDA2, which contains 10M packets, including 326,551 distinct items and 848,638 distinct quadratic elements) for evaluation. The format of data is (SOURCE_IP ◁ DESTINATION_IP).

(2) **Reddit:** We use the data set from the social network – Reddit Hyperlink Network [46] and name it Reddit. This subreddit-to-subreddit hyperlink network is extracted from the posts that create hyperlinks from one subreddit to another, which has 286,561 records in total, including 27,777 distinct items and 137,805 distinct quadratic elements. The format of data is (SOURCE_SUBREDDIT ◁ TARGET_SUBREDDIT).

(3) **Stack:** We use the data set from Stack Overflow Temporal Network [47] and name it Stack. This is a network of interactions on the stack exchange website, which contains 10M records, including 519,167 distinct items and 4,655,787 distinct quadratic elements. The format of data is (SRC ◁ TGT).

Metrics: Our experiments involve the following five metrics. Note that due to space limitations, we only show some of them for each of the three applications as described in Section 6.2-6.4.

(1) **Throughput:** The throughput is defined as the number of quadratic elements inserted per second.

(2) **Precision:** The precision is defined as $\frac{|\Gamma|}{|\Upsilon|}$, where Γ and Υ are the set of correct quadratic elements we find and the set of all the quadratic elements we find, respectively.

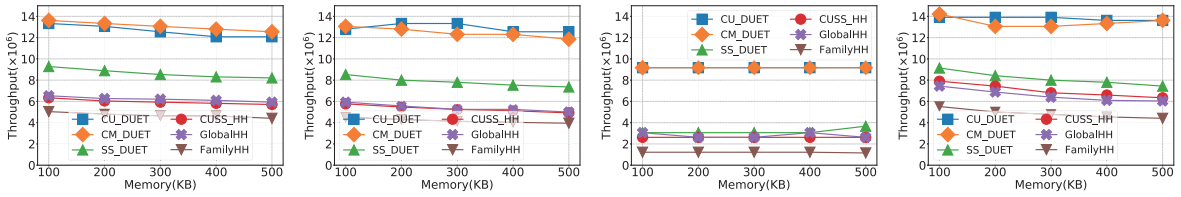
(3) **Recall:** The recall is defined as $\frac{|\Gamma|}{|\Lambda|}$, where Λ is the set of all the correct quadratic elements.

(4) **F1 score:** The F1 score is the harmonic mean of precision and recall, which is given by $\frac{2 * precision * recall}{precision + recall}$.

(5) **Average Relative Error (ARE):** The ARE is defined as $\frac{1}{|\Gamma|} \sum_{(x_i \triangleleft y_i) \in \Gamma} \left| \widehat{f}_{x_i \triangleleft y_i} - f_{x_i \triangleleft y_i} \right| / f_{x_i \triangleleft y_i}$. Especially, for the application PP, we replace f with p.

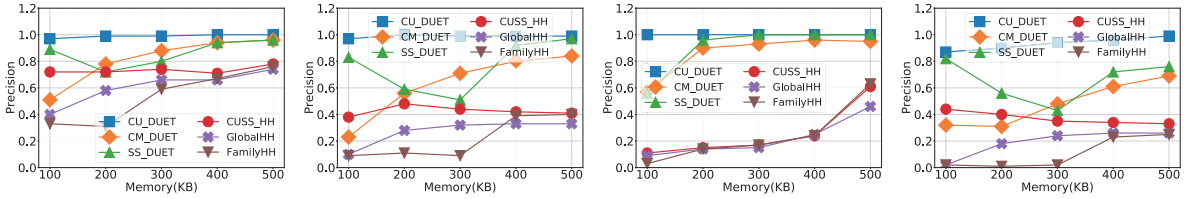
6.2 Experiments for Application HH

In this section, we validate the performance of DUET for application HH. Specifically, we first show the performance comparison between DUET and prior algorithms on different data sets. Then, we show the effects of parameters ϕ_1 and ϕ_2 on the performance of all the algorithms used in HH.



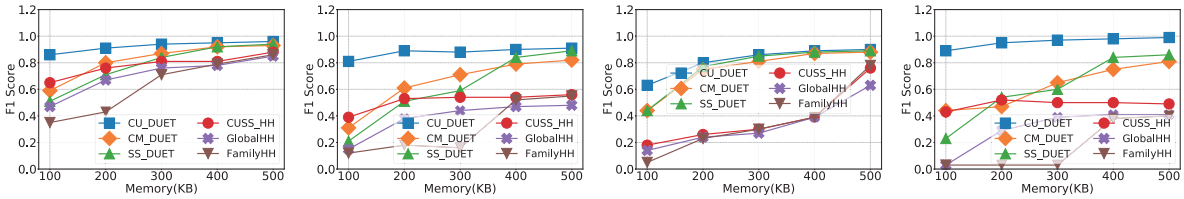
(a) CAIDA1. (b) CAIDA2. (c) Reddit. (d) Stack.

Figure 6: Throughput of the six algorithms for application HH on the four data sets.



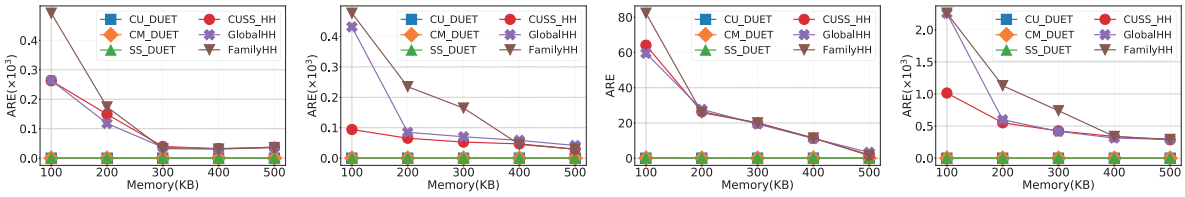
(a) CAIDA1. (b) CAIDA2. (c) Reddit. (d) Stack.

Figure 7: Precision of the six algorithms for application HH on the four data sets.



(a) CAIDA1. (b) CAIDA2. (c) Reddit. (d) Stack.

Figure 8: F1 score of the six algorithms for application HH on the four data sets.



(a) CAIDA1. (b) CAIDA2. (c) Reddit. (d) Stack.

Figure 9: ARE of the six algorithms for application HH on the four data sets.

Table 3: Algorithms used in application HH.

Algorithms	Data Structures
CU_DUET	DUET with CUS [19]
CM_DUET	DUET with CMS [17]
SS_DUET	DUET with SS [48]
CUSS_HH	CUS + SS
GlobalHH [39]	CMS + SS
FamilyHH [32, 39]	SS + SS

6.2.1 ***Algorithms and Their Settings.*** We first show all the algorithms used in experiments and their associated data structures in Table 3, and introduce them in the following:

- **CU_DUET, CM_DUET, and SS_DUET**. All these three algorithms are based on our framework DUET, and they set the sketch in DFilter as CU sketch (CUS) [19], CM sketch (CMS) [17], and Stream-Summary (SS) [48], respectively.
- **CUSS_HH**. This algorithm uses CUS to approximately estimate the frequency of items and uses SS to track frequent quadratic elements.
- **GlobalHH**. The original GlobalHH [39] uses an SS to track frequent quadratic elements, and assumes that there is enough memory to fully record all items, which is not realistic for data streams. To tackle that, we use CMS to approximately estimate the frequency of items.
- **FamilyHH**. The key idea of both FamilyHH [39] and CSS-CHH [32] is to use two SS instances to track frequent items and quadratic elements separately. We refer to these two algorithms collectively as FamilyHH.

6.2.2 ***Performance Comparison on Different Data Sets.*** We vary the memory from 100KB to 500KB and conduct experiments on four different data sets. To compare the performance of all the

algorithms used in HH, we choose four metrics: throughput, precision, F1 score, and ARE, and the experimental results are displayed in Fig. 6-9. In particular, we divide all the six algorithms into three groups according to the data structure for recording items for comparison. That is, CU_DUET and CUSS_HH are a group; CM_DUET and GloablHH are a group; SS_DUET and FamilyHH are a group.

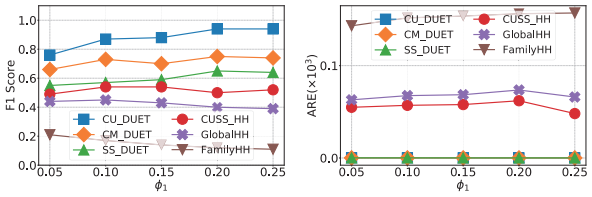
Throughput (Fig. 6): On average, the throughput of CU_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 2.1, 2.4, 3.5, and, 2.0 times higher than that of CUSS_HH, respectively; the throughput of CM_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 2.1, 2.3, 3.3, and, 2.0 times higher than that of GlobalHH, respectively; the throughput of SS_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.8, 2.0, 2.7, and, 1.5 times higher than that of FamilyHH, respectively.

Precision (Fig. 7): On average, the precision of CU_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.3, 2.3, 3.9, and, 2.4 times higher than that of CUSS_HH, respectively; the precision of CM_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.3, 2.3, 4.0, and, 2.4 times higher than that of GlobalHH, respectively; the precision of SS_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.6, 3.5, 3.7, and, 7.5 times higher than that of FamilyHH, respectively.

F1 score (Fig. 8): On average, the F1 score of CU_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.2, 1.7, 2.1, and, 2.0 times higher than that of CUSS_HH, respectively; the F1 score of CM_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.2, 1.7, 2.2, and, 2.0 times higher than that of GlobalHH, respectively; the F1 score of SS_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1.2, 2.0, 2.2, and, 3.7 times higher than that of FamilyHH, respectively.

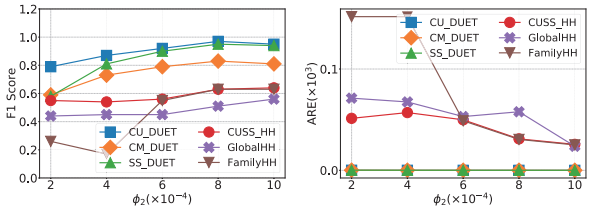
ARE (Fig. 9): On average, the ARE of CU_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 2480.0, 722.7, 326.0, and, 8262.2 times lower than that of CUSS_HH, respectively; the ARE of CM_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 1876.4, 1526.3, 288.6, and, 8754.2 times lower than that of GlobalHH, respectively; the ARE of SS_DUET on CAIDA1, CAIDA2, Reddit, and Stack is around 2732.7, 1828.8, 294.1, and, 9187.6 times lower than that of FamilyHH, respectively.

Analysis: Firstly, it is obvious that the algorithms based on DUET have higher throughput and lower ARE than the other algorithms by comparing each group of algorithms. The reason is that the cardinality of quadratic elements is much larger than the number of counters in SS, and using SS without filtering will cause lots of eviction operations to lower the throughput and increase the estimation error. Secondly, the algorithms based on DUET have higher F1 scores than the other algorithms. The reason is that the special quadratic elements found by CUSS_HH, GlobalHH, and FamilyHH need to satisfy more strict conditions than our problem definition, for that they first need to be frequent enough with respect to all quadratic elements to be recorded in SS. Thirdly, CU_DUET has the best performance on accuracy among the three algorithms based on DUET. The reason is that the conservative updating greatly improves the accuracy of the estimation of items. For SS, it needs to allocate space to record the ID of items. Therefore, SS_DUET has a lower F1 score than CM_DUET when the memory is tight. Besides, the precision of SS_DUET decreases as the memory increases on some data sets. The reason is that when the memory is very tight,



(a) F1 score vs. ϕ_1. (b) ARE vs. ϕ_1.

Figure 10: Experimental results with varying ϕ_1 for application HH.



(a) F1 score vs. ϕ_2. (b) ARE vs. ϕ_2.

Figure 11: Experimental results with varying ϕ_2 for application HH.

SS can record a few items with very high frequencies accurately. As the memory increases, SS overestimates more items, which leads to a decrease in precision.

6.2.3 ***Effects of Parameters ϕ_1 and ϕ_2.*** Recall that for application HH, there are two user-defined parameters ϕ_1 and ϕ_2 that directly effect the estimated results. To explore the effect of the parameter ϕ_1, we fix all the other parameters of the basic framework and set it from 0.05 to 0.25. Analogously, for the parameter ϕ_2, we vary it from 2×10^{-4} to 10^{-3}. Due to the space limitation, we focus on two metrics: F1 score, and ARE, and merely show the related results of CAIDA2 in Fig. 10-11.

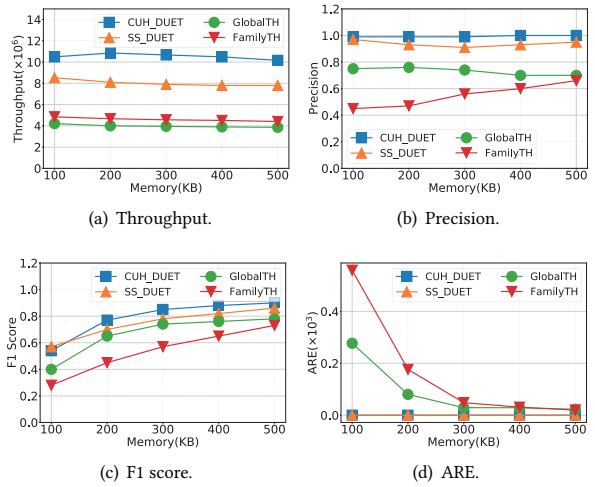
Firstly, it is clear that the algorithms based on DUET always have higher F1 scores and lower ARE than the other three algorithms under all the settings of ϕ_1 and ϕ_2. Secondly, we analyze why all the algorithms based on DUET have certain performance improvements when ϕ_1 or ϕ_2 increases. The reason is that when ϕ_1 or ϕ_2 increases, the smallest frequency of special quadratic elements increases. Note that the quadratic elements with higher frequencies have higher probabilities to be recorded, which leads to higher accuracy when ϕ_1 or ϕ_2 increases. Next, we analyze why F1 scores of CUSS_HH, FamilyHH, and GlobalHH decrease, as is shown in Fig. 10(a). The reason is that SS overestimates quadratic elements, and some normal quadratic elements are misreported as special quadratic elements. As ϕ_1 increases, the number of true special quadratic elements decreases, and the probability of misreporting increases. Generally, the advantages of the algorithms based on DUET are not effected by the settings of ϕ_1 or ϕ_2.

6.3 Experiments for Application TH

In this section, we validate the performance of DUET for application TH. Specifically, we first introduce the settings of all the algorithms used in TH, and show the performance comparison between DUET and prior algorithms. Then, we show the effect of parameter k on all the algorithms.

Table 4: Algorithms used in application TH.

Algorithms	Data Structures
CUH_DUET	DUET with CUS + heap [26]
SS_DUET	DUET with SS
GlobalTH	CUS + heap + SS
FamilyTH	SS + SS



(a) Throughput. (b) Precision.

(c) F1 score. (d) ARE.

Figure 12: Experimental results for application TH.

6.3.1 ***Algorithms and Their Settings.*** All the algorithms used in application TH and their associated data structures are shown in Table 4, and we describe them in the following:

- **CUH_DUET and SS_DUET**. Both of these algorithms are based on our framework DUET, and they set the sketch in DFilter as CUS + heap and SS, respectively.
- **GlobalTH**. This algorithm uses CUS + heap to maintain top-k items and SS to approximately record quadratic elements.
- **FamilyTH**. FamilyHH can be directly used in this application TH. To distinguish it from the previous experiments, we rename it FamilyTH.

6.3.2 ***Performance Comparison.*** To compare the performance of all the algorithms used in TH, we choose four metrics: throughput, precision, F1 score, and ARE, and illustrate the experimental results on CAIDA1 in Fig. 12. Similar to HH, we vary the memory from 100KB to 500KB and divide all these four algorithms into two groups for comparison. That is, CUH_DUET and GlobalTH are a group; SS_DUET and FamilyTH are a group.

Throughput (Fig. 12(a)): The throughput of CUH_DUET is around 2.6 times higher than that of GlobalTH; the throughput of SS_DUET is around 1.8 times higher than that of FamilyTH.

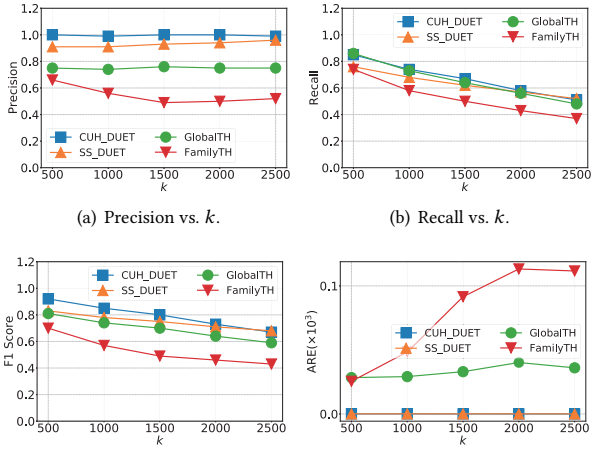
Precision (Fig. 12(b)): The precision of CUH_DUET is around 1.4 times higher than that of GlobalTH; the precision of SS_DUET is around 1.7 times higher than that of FamilyTH.

F1 score (Fig. 12(c)): CUH_DUET has the highest F1 score while FamilyTH has the lowest, and the F1 score of CUH_DUET is around 1.5 times higher than that of FamilyTH.

ARE (Fig. 12(d)): The ARE of CUH_DUET is around 1328.8 times lower than that of GlobalTH; the ARE of SS_DUET is around 2383.2 times lower than that of FamilyTH.

Analysis: Firstly, the speed gap of CUH_DUET and SS_DUET in this application is smaller than that of CU_DUET and SS_DUET in application HH. Besides, the F1 score of SS_DUET is higher than that of CUH_DUET when the memory usage is less than 150KB. The reason is that CUH_DUET needs to allocate a part of the memory to a heap to maintain top-k items. Secondly, the precision of CUH_DUET and SS_DUET is much higher than that of FamilyTH and GlobalTH. The reason is that the quadratic elements found by FamilyTH and GlobalTH need to satisfy more strict conditions than our definition, for that they need to be frequent enough with respect to all the quadratic elements in data streams. Thirdly, the ARE of CUH_DUET and SS_DUET is much lower than that of FamilyTH and GlobalTH. The reason is that DUET filters out a part of quadratic elements and estimates special quadratic elements more accurately.

6.3.3 ***Effect of Parameter k.*** To explore the effect of k, we fix all the other parameters of DUET and vary k from 500 to 2500, and show the experimental results in Fig. 13. Firstly, CU_DUET and SS_DUET have higher precision, higher F1 score, and lower ARE compared with the other two algorithms under all the settings of k. Besides, CU_DUET and SS_DUET always have higher recall than GlobalTH and FamilyTH, respectively, except for that when $k = 500$. Secondly, the performance of all the algorithms degrades as k increases. The reason is that the memory usage is fixed, but the number of special quadratic elements that need to be found increases as k becomes larger. In general, the advantages of the algorithms based on DUET are not effected by the settings of k.



(a) Precision vs. k. (b) Recall vs. k.

(c) F1 score vs. k. (d) ARE vs. k.

Figure 13: Experimental results with varying k for application TH.

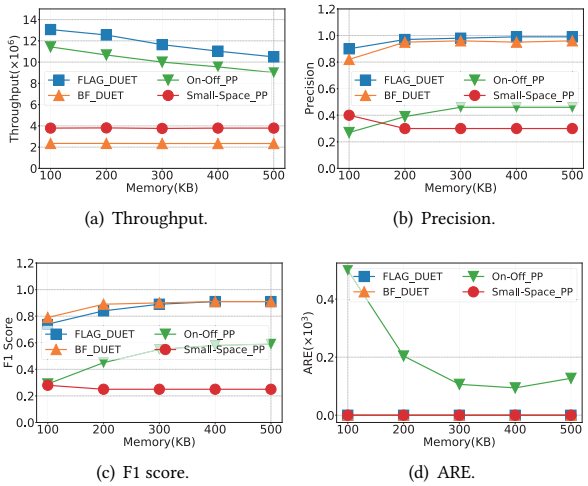
6.4 Experiments for Application PP

In this section, we validate the performance of DUET for application PP. Specifically, we first introduce the settings of DUET and prior algorithms used in PP. Then, we show the performance comparison between DUET and prior algorithms.

6.4.1 ***Algorithms and Their Settings.*** As no prior work can be directly applied to this application, we propose several comparison algorithms based on existing works. All the algorithms used in PP and their associated data structures are shown in Table 5, and we describe them in the following:

Table 5: Algorithms used in application PP.

Algorithms	Data Structures
FLAG_DUET	Flag bits + DUET
BF_DUET	BF [42] + DUET
On-Off_PP	PE [23] + FPI [23]
Small-Space_PP	2 × small space [25]



(a) Throughput. (b) Precision.

(c) F1 score. (d) ARE.

Figure 14: Experimental results for application PP.

- **FLAG_DUET**. We implement the algorithm mentioned in Section 5.3, which sets the sketch as a CU sketch and adds a flag bit to each counter of DUET.
- **BF_DUET**. We also implement the naive method mentioned in Section 5.3, which adds a Bloom Filter on the top of DUET to remove duplicates.
- **On-Off_PP**. The key idea of On-Off_PP is to record items and quadratic elements separately. It uses PE [23] to estimate the persistence of items and FPI [23] to find persistent quadratic elements.
- **Small-Space_PP**. The key idea of Small-Space_PP is to use two small space [25] instances to maintain persistent items and quadratic elements separately.

6.4.2 ***Performance Comparison.*** To compare the performance of all the algorithms, we choose four metrics: throughput, precision, F1 score, and ARE. We vary the memory from 100KB to 500KB and show the experimental results on CAIDA1 in Fig. 14.

Throughput (Fig. 14(a)): The throughput of FLAG_DUET is the highest, and it is around 5.0, 1.2, and 3.1 times higher than that of BF_DUET, On-Off_PP, and Small-Space_PP, respectively.

Precision (Fig. 14(b)): FLAG_DUET and BF_DUET have higher precision than the other two algorithms. Specifically, the precision of FLAG_DUET is around 2.4 and 3.0 times higher than that of On-Off_PP and Small-Space_PP, respectively.

F1 score (Fig. 14(c)): FLAG_DUET and BF_DUET have higher F1 scores than the other two algorithms. Specifically, the F1 score of FLAG_DUET is around 1.7 and 3.4 times higher than that of On-Off_PP and Small-Space_PP, respectively.

ARE (Fig. 14(d)): The ARE of On-Off_PP is highest, and the ARE of the other three algorithms is basically the same.

Analysis: Firstly, Small-Space_PP has the worst performance in terms of the first three metrics, but its ARE is not large. The reason is that small space can accurately estimate the quadratic elements or items with a large persistence under our memory settings. However, those quadratic elements only occupy a small part of all true special quadratic elements, which results in a low F1 score. Secondly, BF_DUET and FLAG_DUET have comparable performance in terms of accuracy. Nevertheless, as the hash calculations increase the overhead of updating, the throughput of BF_DUET is lower than that of FLAG_DUET. Thirdly, comparing On-Off_PP with FLAG_DUET (both of which use flags for indicating), we can find that benefit from our framework, FLAG_DUET has better performance.

7 DISCUSSION

In this section, we briefly discuss some practical extensions of our framework DUET.

Sliding window case: Our proposed framework DUET processes tasks in a fixed time window. However, in some scenarios like real-time monitoring, recent data streams always outweigh outdated streams along the time dimension. This requires the extension of our framework to the sliding window model [49]. To this end, we incorporate the sliding sketch model presented in [50] by dividing each of counters in DUET into several time zones. When inserting a quadratic element, we reuse the original method to update the latest time zone of each part. Moreover, we apply the scanning approach [50] to the DFilter and STable to delete the outdated information. For query, we also follow the original method to query the latest time zone in DUET.

Distributed scenarios: We also extend DUET to distributed processing scenarios. Take the application HH as an example. We deploy DUET with the same parameter setting in $\mathcal{N}$ distributed nodes, and an STable in the centralized controller. Suppose that the stream of each item passes through $\mathcal{P}$ ($\mathcal{P} \leq \mathcal{N}$) nodes, and uniformly distributes its quadratic elements to $\mathcal{P}$ nodes. Besides, denote the stream size as N.

In each node n_i, DUET finds the local quadratic elements that satisfy $Fun(x) \geq \phi_2 \frac{N}{\mathcal{P}}$ and $Fun((x \triangleleft y)) \geq \phi_1 Fun(x)$, and returns these quadratic elements and their frequencies in a set $\mathcal{E}_i$. At the end of such processing, the sketch $\mathcal{K}_i$ in DFilter and the set $\mathcal{E}_i$ are sent to the controller. In the controller, the STable records quadratic elements in each $\mathcal{E}_i$. Meanwhile, the corresponding positions of each $\mathcal{K}_i$ are simply added up to get the final sketch $\mathcal{K}$, which is used to query the frequency of items. Finally, we execute Query in Algorithm 4 to obtain the special quadratic elements.

8 CONCLUSION

In this paper, we study the problem of finding special quadratic elements in data streams. We propose a new framework DUET, which is generic and efficient, and apply it into three important applications related to our problem, HH, TH, and PP. We conduct theoretical analysis to obtain the error bound of DUET and measure the performance of DUET by five metrics on four data sets. Our experimental results show that DUET can achieve up to 3.5 times higher throughput and reduce the average relative error by three orders of magnitude compared with prior algorithms. Besides, we discuss some extensions of DUET to more practical scenarios.

REFERENCES

[1] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *Proceedings of International Conference on Collaboration Technologies and Systems*, pages 42–47. IEEE, 2013.

[2] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of ACM Conference on SIGCOMM*, pages 503–514. ACM, 2014.

[4] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of ACM Conference on SIGCOMM*, pages 254–265. ACM, 2011.

[5] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of ACM Conference on SIGCOMM*, pages 357–371. ACM, 2018.

[6] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of ACM Conference on SIGCOMM*, pages 85–98. ACM, 2017.

[7] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. Network intrusion detection. *Network*, 8(3):26–41, 1994.

[8] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, 2009.

[9] Qun Huang, Patrick PC Lee, and Yungang Bao. SketchLearn: relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of ACM Conference on SIGCOMM*, pages 576–590. ACM, 2018.

[10] Tong Yang, Haowei Zhang, Dongsheng Yang, Yucheng Huang, and Xiaoming Li. Finding significant items in data streams. In *Proceedings of International Conference on Data Engineering*, pages 1394–1405. IEEE, 2019.

[11] Jing Cao, Yu Jin, Aiyou Chen, Tian Bu, and Zhi-Li Zhang. Identifying high cardinality internet hosts. In *Proceedings of International Conference on Computer Communications*, pages 810–818. IEEE, 2009.

[12] Lu Tang, Qun Huang, and Patrick PC Lee. SpreadSketch: Toward invertible and network-wide detection of superspreaders. In *Proceedings of International Conference on Computer Communications*, pages 1608–1617. IEEE, 2020.

[13] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of ACM Conference on SIGCOMM*, pages 234–247. ACM, 2003.

[14] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of ACM Conference on SIGCOMM*, pages 101–114. ACM, 2016.

[15] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of ACM Conference on SIGCOMM*, pages 334–350. ACM, 2019.

[16] Lu Tang, Qun Huang, and Patrick PC Lee. MV-Sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *Proceedings of International Conference on Computer Communications*, pages 2026–2034. IEEE, 2019.

[17] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[18] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM Conference on SIGCOMM*, pages 323–336. ACM, 2002.

[19] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[20] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of International Conference on Database Theory*, pages 398–412. Springer, 2005.

[21] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of International Conference on Very Large Databases*, pages 346–357. Elsevier, 2002.

[22] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*, pages 2584–2593. ACM, 2018.

[23] Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. On-off sketch: a fast and accurate sketch on persistence. *Proceedings of the VLDB Endowment*, 14(2):128–140, 2020.

[24] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and Yuankun Zhong. Finding persistent items in data streams. *Proceedings of the VLDB Endowment*, 10(4):289–300, 2016.

[25] Bibudh Lahiri, Srikanta Tirthapura, and Jaideep Chandrashekar. Space-efficient tracking of persistent items in a massive data stream. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(1):70–92, 2014.

[26] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. WavingSketch: An unbiased and generic sketch for finding top-k items in data streams. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*, pages 1574–1584. ACM, 2020.

[27] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Faster and more accurate measurement through additive-error counters. In *Proceedings of International Conference on Computer Communications*, pages 1251–1260. IEEE, 2020.

[28] Stuart Staniford, James A Hoagland, and Joseph M McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[29] Advanced persistent threat. http://www.usenix.org/event/lisa09/tech/slides/daly.pdf.

[30] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of International Workshop on Recent Advances in Intrusion Detection*, pages 326–345. Springer, 2009.

[31] Nicole Immorlica, Kamal Jain, Mohammad Mahdian, and Kunal Talwar. Click fraud resistant methods for learning click-through rates. In *Proceedings of International Workshop on Internet and Network Economics*, pages 34–45. Springer, 2005.

[32] Italo Epicoco, Massimo Cafaro, and Marco Pulimeno. Fast and accurate mining of correlated heavy hitters. *Data Mining and Knowledge Discovery*, 32(1):162–186, 2018.

[33] Marco Pulimeno, Italo Epicoco, Massimo Cafaro, Catiuscia Melle, and Giovanni Aloisio. Parallel mining of correlated heavy hitters on distributed and shared-memory architectures. In *Proceedings of International Conference on Big Data*, pages 5111–5118. IEEE, 2018.

[34] Bibudh Lahiri and Srikanta Tirthapura. Finding correlated heavy-hitters over data streams. In *Proceedings of International Performance Computing and Communications Conference*, pages 307–314. IEEE, 2009.

[35] Jayadev Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.

[36] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of ACM Conference on SIGCOMM*, pages 267–280. ACM, 2010.

[37] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 561–575. ACM, 2018.

[38] Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Optimal elephant flow detection. In *Proceedings of International Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[39] Katsiaryna Mirylenka, Themis Palpanas, Graham Cormode, and Divesh Srivastava. Finding interesting correlations with conditional heavy hitters. In *Proceedings of International Conference on Data Engineering*, pages 1069–1080. IEEE, 2013.

[40] Amin Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

[41] Bobhash. http://burtleburtle.net/bob/hash/evahash.html.

[42] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[43] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[44] Source codes of DUET. https://github.com/callitwhatyouwannt13/DUET_Code.

[45] The caida anonymized internet traces. https://www.caida.org/data/passive/passive_2016_dataset.xml.

[46] Social network: Reddit hyperlink network. http://snap.stanford.edu/data/soc-RedditHyperlinks.html.

[47] Stack overflow temporal network. http://snap.stanford.edu/data/sx-stackoverflow.html.

[48] The source codes of HeavyKeeper. https://github.com/papergitkeeper/heavy-keeper-project.

[49] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[50] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *Proceedings of International Conference on Knowledge Discovery & Data Mining*, pages 1015–1025, 2020.