

# Approximating Frequent Items in Asynchronous Data Stream over a Sliding Window

Ho-Leung Chan, Tak-Wah Lam\*, Lap-Kei Lee, and Hing-Fung Ting\*\*

Department of Computer Science, University of Hong Kong  
{hlchan,twlam,lklee,hfting}@cs.hku.hk

**Abstract.** In an asynchronous data stream, the data items may be out of order with respect to their original timestamps. This paper gives a space-efficient data structure to maintain such a data stream so that it can approximate the frequent item set over a sliding time window with sufficient accuracy. Prior to our work, Cormode *et al.* [3] have the best solution, with space complexity  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}) \min\{\log W, \frac{1}{\varepsilon}\} \log U)$ , where  $\varepsilon$  is the given error bound,  $W$  and  $B$  are parameters of the sliding window, and  $U$  is the number of all possible item names. Our solution reduces the space to  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$ . We also unify the study of synchronous and asynchronous data stream by quantifying the delay of the data items. When the delay is zero, our solution matches the space complexity of the best solution to the synchronous data streams [8].

## 1 Introduction

Identifying frequent items in a massive data stream has many applications in data mining and network monitoring, and the problem has been studied extensively [8, 7, 9, 6, 5, 1]. Recent interest has been shifted from the statistics of the whole data stream to that of a sliding window of recent data [1, 8]. In most applications, the amount of data in a window is gigantic when compared with the amount of memory in the processing units, and it is impossible to store all the data and to find the exact frequent items. Existing research has focused on finding space-efficient data structures to support finding approximate frequent items. The key concern is how to minimize the space so as to achieve a given level of accuracy.

**Asynchronous data stream.** Most of the previous work on data streams assume that items in a data stream are synchronous in the sense that the order of their arrivals is the same as the order of their creations. This synchronous model is however not suitable to applications that are distributed in nature. For example, in a sensor network, the sink collects data transmitted from sensors over a large area, and the data transmitted from different sensors would suffer different delay. It is possible that an item created at time  $t$  at a certain

---

\* T.W. Lam is partially supported by GRF Grant HKU 713909E.

\*\* H.F. Ting is partially supported by GRF Grant HKU 716307E.

sensor may arrive at the sink later than an item created after  $t$  at another sensor. From the sink's viewpoint, items in the data stream are out of order with respect to their creation times. Yet the statistics to be computed are usually based on the creation times. More specifically, an *asynchronous data stream* (or equivalently, *out-of-order data stream*) [11, 2, 3] can be considered as a sequence  $\langle (a_1, d_1), (a_2, d_2), (a_3, d_3), \dots \rangle$ , where  $a_i$  is the name of a data item chosen from a fixed universe  $\Sigma$ , and  $d_i$  is an integer timestamp recording the creation time of this item. Items arriving at the data stream are in arbitrary order regarding their timestamps. Furthermore, it is possible that more than one data item has the same timestamp.

**Previous work on approximating frequent items.** Consider a data stream and, in particular, those data items whose timestamps fall into the last  $W$  time units ( $W$  is the size of the sliding window). An item (precisely, an item name) is said to be a frequent item if its count (i.e., the number of occurrences) exceeds a certain required threshold of the total item count. Arasu and Manku [1] were the first to study the data structures for computing approximate frequent items over a sliding window under the synchronous model (in which data items arrive in non-decreasing order of timestamps). The space complexity is  $O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon} \log(\varepsilon B))$ , where  $\varepsilon$  is a user-specified error bound and  $B$  is the maximum number of items with timestamps falling into the same sliding window. Their work was later improved by Lee and Ting [8] to  $O(\frac{1}{\varepsilon} \log(\varepsilon B))$  space. Recently, Cormode *et al.* [3] has initiated the study of frequent items under the asynchronous model, and gave a solution with space complexity  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W})) \min\{\log W, \frac{1}{\varepsilon}\} \log U$ , where  $U = |\Sigma|$  is the number of possible item names.

The earlier work on asynchronous data stream focused on a relatively simpler problem called basic counting [11, 2].<sup>1</sup> In the same paper, Cormode *et al.* [3] improved the space complexity of basic counting to  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$ . Notice that under the synchronous model, the best data structure requires  $O(\frac{1}{\varepsilon} \log(\varepsilon B))$  space [4]. It is believed that there is roughly a gap of  $\log W$  between the synchronous model to the asynchronous model. Yet, for frequent items, the asynchronous result of Cormode *et al.* [3] requires space way bigger than that of the best synchronous result (which is  $O(\frac{1}{\varepsilon} \log(\varepsilon B))$ ) [8]. This motivates us to study better solutions for approximating frequent items in the asynchronous model.

**Formal definition of approximate frequent item set.** Consider an asynchronous data stream  $\langle (a_1, d_1), (a_2, d_2), \dots \rangle$ . The current time is defined to be the maximum timestamp over all items received thus far. If the current time is  $t$ , a sliding window of size  $W$  covers the time interval  $[t - W + 1, t]$ . For any time interval  $I$  and any data item  $a$ , let  $f_a(I)$  denote the frequency of item  $a$  in interval  $I$ , i.e., the number of items named  $a$  in the data stream with timestamps falling into  $I$ . Let  $\Sigma$  be the set of all possible item names. Define  $f_*(I) = \sum_{a \in \Sigma} f_a(I)$  to be the total number of all items in the stream with timestamps within  $I$ .

<sup>1</sup> It is worth-mentioning that with respect to a sliding time window, the problem of approximating the basic counting can be reduced to the problem of approximating the frequent item set problem.

**Table 1.** The space and time complexity for finding  $\varepsilon$ -approximate frequent item set in a sliding time window. Results from this paper are marked with [†].

	Space (words)	Update time	Query time
Synchronous [8]	$O(\frac{1}{\varepsilon} \log(\varepsilon B))$	$O(\frac{1}{\varepsilon} \log(\varepsilon B))$	$O(\frac{1}{\varepsilon})$
Asynchronous [3]	$O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}) \cdot \min\{\log W, \frac{1}{\varepsilon}\} \log U)$	$O(\log(\frac{\varepsilon B}{\log W}) \log W \log \log U)$	linear to space
Asynchronous [†]	$O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$	$O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log \log W))$	$O(\frac{1}{\varepsilon} \log W + \log \log(\varepsilon B))$
Asynchronous with tardiness $\tau$ [†]	$O(\frac{1}{\varepsilon} \log \tau \log(\frac{\varepsilon B}{\log \tau}))$	$O(\log(\frac{\varepsilon B}{\log \tau})(\frac{1}{\varepsilon} + \log \log W))$	$O(\frac{1}{\varepsilon} \log \tau + \log \log(\varepsilon B))$

Given a user-specified error bound  $\varepsilon$  and a window size  $W$ , we want to maintain a data structure to answer any  $\varepsilon$ -approximate frequent item set query in the form  $(\phi, W')$ , where  $\phi \geq \varepsilon$  and  $W' \leq W$ . The answer to such a query is a set  $S$  of item names defined as follows. Let  $t$  be the current time, and let  $I = [t - W' + 1, t]$  be the current window.

- (i)  $S$  contains every item  $a$  whose frequency in interval  $I$  is at least  $\phi f_*(I)$  (i.e.,  $f_a(I) \geq \phi f_*(I)$ ); and
- (ii) For any item  $a$  in  $S$ , its frequency in interval  $I$  is at least  $(\phi - \varepsilon) f_*(I)$  (i.e.,  $f_a(I) \geq (\phi - \varepsilon) f_*(I)$ ).

For example, assume  $\varepsilon = 1\%$ , then the query  $(10\%, 10000)$  would return all items whose frequencies are each at least 10% of the total item count in the last 10000 time units, plus possibly some items with frequency at least 9% of the total count. The set  $S$  is also called the  $\varepsilon$ -approximate  $\phi$ -frequent item set.

**Our contribution.** Our main result is a space-efficient data structure for computing an  $\varepsilon$ -approximate frequent item set; it uses  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words, where  $B$  is the maximum number of items with timestamp in the same window of  $W$  time units. Our memory usage is much smaller than that of the algorithm in [3], i.e.,  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}) \min\{\log W, \frac{1}{\varepsilon}\} \log U)$  words. We have also improved the time complexity of the update and query operations. See Table 1 for a comparison. Interestingly, the space complexity for finding approximate frequent items can now match the best space requirement for the relatively simpler problem of approximating basic counting (i.e.,  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$ ) [3].

In the asynchronous model, if a data item has a delay more than  $W$  time units, it can be immediately discarded when it arrives at the data stream. On the other hand, in many applications, the delay is relatively small when compared with the window size. This motivates us to extend the asynchronous model to consider data items that have a bounded delay. We say that an asynchronous data stream has tardiness  $\tau$  if a data item created at time  $d$  must arrive at the data stream no later than a data item created at time  $d + \tau$ . If we set  $\tau = 0$ , the model becomes the synchronous model. If we consider  $\tau = W$ , this is the asynchronous model studied above. In general, for any  $\tau \in [0, W]$ , we adapt our approximate

data structure for the frequent item set to occupy  $O(\frac{1}{\varepsilon} \log \tau \log(\frac{\varepsilon B}{\log \tau}))$  space. Note that our work implies the same space complexity as the best result under the synchronous model [8], but the underlying data structures and algorithms are more complicated than that in [8].

**Technical digest.** Our improved solution to the frequent item set problem stems from two new ideas, namely, a more versatile data structure called  $\lambda$ -counter for bookkeeping individual data items, and a technique of exploiting multi-resolution to optimize the space for organizing the  $\lambda$ -counters.

The  $\lambda$ -counter is a generalization of the data structure proposed by Lee and Ting [8], which estimates the number of an item within a sliding window under the synchronous model. Roughly speaking, the idea in [8] is to keep a sample of every  $\lambda$  item  $a$ 's received, for some constant  $\lambda$ . In an out-of-order stream, the order of timestamps is arbitrary, and it does not make sense to sample every  $\lambda$  item  $a$ 's in the stream. Our idea is to use an *interval splitting* technique, which dynamically (and erroneously) splits the current window into disjoint intervals of varying sizes but with similar number of items. An interesting point is that the splitting will inevitably introduce error, but we are able to keep the error in control. In Cormode *et al.*'s solution [3], they make use of the data structure q-digest (first proposed in [10]) for a similar purpose. Both q-digest and  $\lambda$ -counter allow efficient insertion of a new item. But an obvious advantage of  $\lambda$ -counter is that it also allows the deletion of the latest item in  $O(1)$  time. The special deletion operation allows us to have a better and simpler way to organize the  $\lambda$ -counters for approximating frequent item set.

Based on the new  $\lambda$ -counters, it is not difficult to adapt Lee and Ting's synchronous data structure for approximating frequent item set [8] to the asynchronous model. The space complexity would be  $O(\frac{1}{\varepsilon}(\log W)B)$ . The extra factor  $B$  is due to the fact that the approximation has to be within an absolute error  $\varepsilon f_*(I)$ , where  $f_*(I)$  can be as small as one and as big as  $B$ . An useful observation here is that there is a more error-sensitive way to exploit the  $\lambda$ -counters such that if  $f_*(I)$  is restricted to be in the range  $[\ell, r]$ , then the space complexity for approximating frequent item set would be  $O(\frac{1}{\varepsilon}(\log W)\frac{B}{\ell})$ . We call such a structure the  $(Y, \delta)$ -collection (see the details in Section 3). Then we can adopt the "multi-resolution" idea of [1] to keep  $O(\log(\frac{\varepsilon B}{\log W}))$  data structures, each could estimate the frequent item set with sufficient accuracy when  $f_*(I)$  is in a particular range, and each requires only  $O(\frac{1}{\varepsilon} \log W)$  space.

**Organization of the paper.** Sections 2 and 3 present the data structures  $\lambda$ -counter and  $(Y, \delta)$ -collection, respectively; the latter gives good estimates on the frequency of any item, when the total frequency of all items in the window is bounded by some fixed value  $Y$ . Section 4 uses the multi-resolution technique to remove the restriction on total frequency, and Section 5 shows how this data structure can identify  $\varepsilon$ -approximate frequent item set using  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words of space. Finally, Section 6 extends the results for out-of-order stream with tardiness.

## 2 $\lambda$ -Counter: Estimating the Frequency of a Fixed Item

This section presents a new data structure  $\lambda$ -counter, where  $\lambda \geq 1$  is a parameter. Let  $W$  be the window size. Each  $\lambda$ -counter counts only one certain item, and it maintains an estimation for the number of this item within a sliding window of  $W'$  time units, for any  $W' \leq W$ . The absolute error in the estimation is at most  $2\lambda \log W$ . We first define a  $\lambda$ -counter and then present the analysis.

### 2.1 Definition of a $\lambda$ -Counter

A  $\lambda$ -counter  $C_a$  for an item  $a$  is simply a list of intervals

$$\langle [p_1, q_1], [p_2, q_2], [p_3, q_3], \dots, [p_k, q_k] \rangle, \quad (1)$$

where  $p_i \leq q_i$  for  $i = 1, \dots, k$ . We will maintain that  $p_1 < q_k$ , and the intervals in the list form a partition of  $[p_1, q_k]$  (i.e.,  $p_{i+1} = q_i + 1$  for any  $1 \leq i < k$ ); we say that  $C_a$  *monitors* the interval  $[p_1, q_k]$ . Each interval  $[p_i, q_i]$  in the list is associated with a number  $e_a([p_i, q_i])$ , which is an estimate of  $f_a([p_i, q_i])$ , i.e., the number of item  $a$  in  $[p_i, q_i]$ . Initially, the list has only one interval  $[1, W]$ , and  $e_a([1, W]) = 0$ . A  $\lambda$ -counter is updated and answers queries as follows.

**Updating a  $\lambda$ -counter.** A  $\lambda$ -counter  $C_a$  is updated only when an item  $a$  arrives. Suppose a new item  $(a, t)$  arrives. Let the list of  $C_a$  be the one shown in (1) for some  $k \geq 1$ . Then we update the list according to the three cases below.

**Case 1:**  $t < p_1$ . We ignore the new item in this case.

**Case 2:**  $p_1 \leq t \leq q_k$ . In this case, let  $[p_i, q_i]$  be the unique interval in the list with  $t \in [p_i, q_i]$ . We increase its estimate  $e_a([p_i, q_i])$  by 1.

Then, if  $p_i \neq q_i$  and  $e_a([p_i, q_i]) = 2\lambda$ , we do the following extra step. Let  $m = \lfloor (p_i + q_i)/2 \rfloor$ . We replace the interval  $[p_i, q_i]$  by the two intervals  $[p_i, m]$  and  $[m + 1, q_i]$ . Then we set both  $e_a([p_i, m])$  and  $e_a([m + 1, q_i])$  to  $\lambda$ .

**Case 3:**  $q_k < t$ . In this case, we have two subcases.

1. If  $q_k < t \leq q_k + W$ , then append the interval  $[q_k + 1, q_k + W]$  to the list and set its estimate to 1.
2. If  $q_k + W < t$ , then find the unique integer  $\ell$  such that  $t \in [(\ell - 1)W + 1, \ell W]$ , and append to the list the two intervals  $[(\ell - 2)W + 1, (\ell - 1)W]$  and  $[(\ell - 1)W + 1, \ell W]$  with estimates 0 and 1, respectively.

After the above updates, we remove all expired intervals, i.e., intervals  $[p, q]$  in the list with  $q \leq t_{\text{cur}} - W$ , where  $t_{\text{cur}}$  is the current time, i.e., the largest timestamp of items received so far.

**Estimation by a  $\lambda$ -counter.** Let  $t_{\text{cur}}$  be the current time and consider any time  $t_{\text{cur}} - W + 1 \leq t \leq t_{\text{cur}}$ . We estimate the number of item  $a$  in the interval  $[t, t_{\text{cur}}]$  by summing up the estimates of all intervals  $[p_i, q_i]$  in the list of  $C_a$  such that  $t \leq q_i$ . That is, the estimate is

$$E_a(t) = \sum \{e_a([p_i, q_i]) \mid [p_i, q_i] \text{ is in list of } C_a \text{ and } t \leq q_i\} \quad (2)$$

## 2.2 Analysis of the $\lambda$ -Counter

Without loss of generality, we assume that the first item or query arrives no earlier than time  $W$ . We first state a fact which can be proved easily by induction.

**Fact 1.** *At any time  $t_{\text{cur}} \geq W$ , suppose  $C_a$  contains a list as shown in (1) for some  $k \geq 1$ . Then,  $p_1 \leq t_{\text{cur}} - W + 1 \leq t_{\text{cur}} \leq q_k$ , and  $q_k$  is a multiple of  $W$ .*

It is not hard to see the memory requirement of a  $\lambda$ -counter  $C_a$  (Lemma 1), since  $[p_1, q_k]$  has length  $O(W)$  and the total number of item  $a$  in this interval is  $O(B)$ , where  $B$  is the maximum number of items with timestamp in a window.

**Lemma 1.** *The memory usage of a  $\lambda$ -counter  $C_a$  is  $O(\frac{B}{\lambda})$  words.*

It is straightforward to see that each update takes  $O(\log(\frac{B}{\lambda}))$  time (by binary search on the list) and each query takes  $O(\frac{B}{\lambda})$  time. It remains to show that  $C_a$  gives estimates within the stated error bound. Suppose  $C_a$  contains a list as shown in (1) for some  $k \geq 1$ . We focus on some time  $t \in [t_{\text{cur}} - W + 1, t_{\text{cur}}] \subseteq [p_1, q_k]$ . For such  $t$ , there is a unique interval  $I_c = [p_c, q_c]$  in the list of  $C_a$  such that  $t \in I_c$ ; we call  $I_c$  the *critical interval* for  $t$ . We can check that Equation (2) is equivalent to

$$E_a(t) = \sum_{i=c, \dots, k} e_a([p_i, q_i])$$

The following technical lemma analyzes how well  $E_a(t)$  estimates  $f_a([t, q_k])$ . We say that an update is *critical* to  $t$  if it splits the critical interval for  $t$  (i.e., Case 2, and the extra step is executed on the critical interval for  $t$ ). The proof of the lemma is a tedious verification of Inequality (3) and will be given in full paper.

**Lemma 2.** *Let  $I = [p_1, q_k]$  be the interval monitored by  $C_a$ . Consider any time  $t \in [t_{\text{cur}} - W + 1, t_{\text{cur}}]$ . Let  $I_c$  be the critical interval for  $t$  and let  $n_c$  be the total number of updates made to  $C_a$  so far that are critical to  $t$ . Then,*

$$E_a(t) - \min\{e_a(I_c), 2\lambda\} - \lambda n_c \leq f_a([t, q_k]) \leq E_a(t) + 2\lambda n_c. \quad (3)$$

We can apply Lemma 2 to obtain our main theorem, as follows. Note that if  $W < 4$ , we can use three counters to maintain the exact count of an item  $a$ .

**Theorem 1.** *For  $W \geq 4$  and any  $W' \leq W$ , the estimate  $E_a(t_{\text{cur}} - W' + 1)$  returned by  $C_a$  has absolute error at most  $2\lambda \log W$ .*

*Proof.* Observe that no item with a timestamp larger than  $t_{\text{cur}}$  arrives yet, so  $f_a([t_{\text{cur}} - W' + 1, t_{\text{cur}}]) = f_a([t_{\text{cur}} - W' + 1, q_k])$ . Then by Lemma 2, we have

$$E_a(t_{\text{cur}} - W' + 1) - 2\lambda - \lambda n_c \leq f_a([t_{\text{cur}} - W' + 1, t_{\text{cur}}]) \leq E_a(t_{\text{cur}} - W' + 1) + 2\lambda n_c$$

where  $n_c$  is the number of updates critical to  $t_{\text{cur}} - W' + 1$ . Since  $W \geq 4$ ,  $2\lambda \leq \lambda \log W$ . Note that  $n_c \leq \log W$  because  $t_{\text{cur}} - W' + 1$  is initially in an interval of size at most  $W$  and each critical update reduces the size of the critical interval by half. Since the interval length is at least 1, the theorem follows.  $\square$

### 3 $(Y, \delta)$ -Collection: Estimating the Frequency of Every Item When Total Frequency Is at Most $Y$

Let  $Y$  be a fixed constant and  $\delta$  be an error bound. This section presents a data structure  $(Y, \delta)$ -collection, which maintains an estimation of the frequency of any item within a sliding window of  $W$  time units. We will define a  $(Y, \delta)$ -collection and show that the absolute error of an estimation is at most  $\delta Y$ , when the total frequency of all items within the window is at most  $Y$ . In the next section, we will extend the data structure to remove this restriction.

A  $(Y, \delta)$ -collection is a collection of at most  $\frac{2^4}{\delta} \lambda$ -counters where  $\lambda = \frac{\delta Y}{10 \log W}$ . Below we assume that  $W \geq 4$ .<sup>2</sup> Initially, the collection has no counter.

#### 3.1 Updating a $(Y, \delta)$ -Collection

When an item  $(a, t)$  arrives, we update as follows.

**Case 1.** If the collection has a  $\lambda$ -counter  $C_a$  for  $a$ , we update  $C_a$  by  $(a, t)$ . If after the update, we have either

- (i) the total number of intervals of all  $\lambda$ -counters in the collection is greater than  $(\frac{82}{\delta}) \log W$ , or
- (ii) the total estimate of all intervals of all  $\lambda$ -counters in the collection is greater than  $\frac{58}{10} Y$ ,

then we find, from the lists of intervals of all  $\lambda$ -counters, the interval  $[p, q]$  with the smallest  $q$ , and discard it from the collection.

**Case 2.** Suppose the collection does not have a  $\lambda$ -counter for  $a$ . If the total number of  $\lambda$ -counters is smaller than  $\frac{2^4}{\delta}$ , we create a new  $\lambda$ -counter for  $a$ , and update it by  $(a, t)$ . Otherwise, we do nothing about  $(a, t)$ . Instead, we perform a “decrement” operation to every  $\lambda$ -counter  $C_x$  in the collection:

**Decrement of  $C_x$ .** We find the last interval  $[p, q]$  in  $C_x$  (i.e., the one with the largest  $q$ ) with a positive estimate  $e_x([p, q])$ . Then, we decrease this estimate  $e_x([p, q])$  by 1. If the estimate becomes zero, and  $q$  is not a multiple of  $W$ , and there is another interval  $[q + 1, r]$  with estimate zero, then we combine the two intervals  $[p, q]$  and  $[q + 1, r]$  into one interval  $[p, r]$  with estimate  $e_x([p, r]) = 0$ .

For ease of reference, we call this step a batch-of-decrement step. After it, we remove all  $\lambda$ -counters which do not have any interval with positive estimate.

By the update operation, we have some simple fact and observation about a  $(Y, \delta)$ -collection. Consider any  $\lambda$ -counter  $C_a$  in the  $(Y, \delta)$ -collection. Suppose that  $C_a$  is monitoring the list of intervals  $\langle [p_1, q_1], [p_2, q_2], \dots, [p_{k-1}, q_{k-1}], [p_k, q_k] \rangle$ . The following fact is easy to verify.

**Fact 2.** *With respect to the  $\lambda$ -counter  $C_a$ , there are at most two intervals in its list with estimates smaller than  $\lambda$ .*

<sup>2</sup> If  $W < 4$ , we can replace each  $\lambda$ -counter for some item  $a$  with  $W$  counters to maintain the exact count of  $a$  in the window. Such  $(Y, \delta)$ -collection using exact counters can be analyzed similarly to that using  $\lambda$ -counters.

In Case 1 of the update operation, we discard some interval if (i) the total number of intervals is greater than  $(\frac{82}{\delta}) \log W$  or (ii) the total estimate of all intervals is greater than  $\frac{58}{10}Y$ . The following lemma asserts that no “important” interval is discarded if the total frequency of all items within the window is at most  $Y$ . Let  $I_{\text{cur}} = [t_{\text{cur}} - W + 1, t_{\text{cur}}]$ . Note that  $\Sigma$  is the set of possible items and  $f_*(I_{\text{cur}}) = \sum_{a \in \Sigma} f_a(I_{\text{cur}})$ .

**Lemma 3.** *Suppose that we have discarded at or before  $t_{\text{cur}}$  an interval  $[p, q]$  from the  $(Y, \delta)$ -collection with  $q \geq t_{\text{cur}} - W + 1$ . Then  $f_*(I_{\text{cur}}) > Y$ .*

*Proof.* Consider the moment of the removal. We claim that

$$\sum_{C_a \text{ in the collection}} E_a(t_{\text{cur}} - W + 1) > \frac{58}{10}Y.$$

It is obviously true if the removal is triggered by Case 1(ii). Suppose the removal is triggered by Case 1(i). Then there are totally more than  $(\frac{82}{\delta}) \log W$  intervals in the collection. Since we remove the interval  $[p, q]$  with the smallest  $q$ , and  $q \geq t_{\text{cur}} - W + 1$ , we conclude that all these intervals  $[x, y]$  have  $y \geq t_{\text{cur}} - W + 1$ . Together with the fact that there are at most  $\frac{24}{\delta} \lambda$ -counters and each of them has at most two intervals with estimates smaller than  $\lambda$  (Fact 2), we conclude

$$\sum_{C_a \text{ in the collection}} E_a(t_{\text{cur}} - W + 1) > \lambda(\frac{82}{\delta} \log W - 2(\frac{24}{\delta})) \geq \lambda(\frac{82}{\delta} - \frac{24}{\delta}) \log W = \frac{58}{10}Y$$

If there is no batch-of-decrement step, by Theorem 1, the total estimate is at most

$$\sum_{C_a \text{ in the collection}} (f_a(I_{\text{cur}}) + 2\lambda \log W) \leq f_*(I_{\text{cur}}) + \frac{24}{\delta}(2\lambda \log W).$$

Otherwise, suppose there is a batch-of-decrement step. We first claim that Theorem 1 still holds. To see this, consider the decrement of some  $\lambda$ -counter  $C_x$ , which essentially deletes the latest item  $x$  in the stream. If the decrement combines two intervals  $[p, q]$  and  $[q + 1, r]$  into an interval  $[p, r]$ , then the number of critical update to any time  $t \in [p, r]$  can be reset to 0 (because  $e_a([p, r]) = 0$  and any item  $x$  with timestamps in  $[p, r]$  has been deleted). Furthermore, the condition that  $q$  is not a multiple of  $W$  guarantees  $[p, r]$  has size at most  $W$ . Thus, we still have Theorem 1. The decrement operations will only make the estimates smaller and thus the above inequality still holds even when there are decrements. Combining the above two inequalities and recall that  $\lambda = \frac{\delta Y}{10 \log W}$ , the lemma follows.  $\square$

### 3.2 Estimation by a $(Y, \delta)$ -Counter

Given any  $W' \leq W$ , let  $I_{W'} = [t_{\text{cur}} - W' + 1, t_{\text{cur}}]$ . The  $(Y, \delta)$ -collection gives an estimate  $\text{Est}_a(I_{W'})$  of  $f_a(I_{W'})$  for every item  $a$  as follows.

- If the  $(Y, \delta)$ -collection has a  $\lambda$ -counter  $C_a$  for item  $a$ , then  $\text{Est}_a(I_{W'})$  is equal to the estimate  $E_a(t_{\text{cur}} - W' + 1)$  of  $C_a$ ;
- Otherwise,  $\text{Est}_a(I_{W'}) = 0$ .

We can analyze the accuracy of the estimate given by a  $(Y, \delta)$ -collection, as follows. Note that  $I_{\text{cur}} = [t_{\text{cur}} - W + 1, t_{\text{cur}}]$ .

**Lemma 4.** *Suppose that  $f_*(I_{\text{cur}}) \leq Y$ . Then, for any item  $a$  and any  $W' \leq W$ , we have*

$$f_a(I_{W'}) - \delta Y \leq \text{Est}_a(I_{W'}) \leq f_a(I_{W'}) + \delta Y.$$

*Proof.* We first derive an upper bound on the number of batch-of-decrement steps performed during  $I_{\text{cur}}$ . Note that at time  $t_{\text{cur}} - W$ , the total estimate of all intervals in the  $(Y, \delta)$ -collection is at most  $\frac{58}{10}Y$ . Together with the fact that  $f_*(I_{\text{cur}}) \leq Y$  items arrived during  $I_{\text{cur}}$ , the total units that can be decreased by the batch-of-decrement steps performed during  $I_{\text{cur}}$  is at most  $\frac{68}{10}Y$ . Since each batch-of-decrement step decreases  $\frac{24}{\delta} + 1$  units (each for a distinct item), we conclude that the number of batch-of-decrement steps performed during  $I_{\text{cur}}$  is at most  $(\frac{68}{10}Y)/(\frac{24}{\delta} + 1) < \frac{68}{240}\delta Y$ .

We can now analyze the accuracy of the estimate  $\text{Est}_a(I_{W'})$ .

- If there is no counter in the  $(Y, \delta)$ -collection for  $a$ , then  $\text{Est}_a(I_{W'}) = 0$ . Note that  $f_a(I_{W'})$  is at most  $\frac{68}{240}\delta Y$  because we need a batch-of-decrement step to take away a unit for item  $a$  and there are at most  $\frac{68}{240}\delta Y$  such steps. Thus,  $|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{68}{240}\delta Y < \delta Y$ .
- Suppose that there is a  $\lambda$ -counter  $C_a$  for  $a$ . Since  $f_*(I_{\text{cur}}) \leq Y$ , Lemma 3 asserts that no interval  $[p, q]$  with  $q \geq t_{\text{cur}} - W + 1$  is removed. Thus, we can apply Equation (2) in Section 2.1 to determine  $E_a(t_{\text{cur}} - W + 1)$ . If there is no batch-of-decrement step, by Theorem 1, we have

$$|\text{Est}_a(I_{W'}) - f_a(I_{W'})| = |E_a(t_{\text{cur}} - W + 1) - f_a(I_{W'})| \leq 2\lambda \log W.$$

The decrement operations will introduce an additional error of at most  $\frac{68}{240}\delta Y$  and thus we have

$$|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq 2\lambda \log W + \frac{68}{240}\delta Y = 2\left(\frac{\delta Y}{10 \log W}\right) \log W + \frac{68}{240}\delta Y < \delta Y. \square$$

## 4 Estimating the Frequency of Every Item

In this section, we build a data structure  $D_\varepsilon$  using  $(Y, \delta)$ -collections and show that  $D_\varepsilon$  can give good estimates without any restriction on the total frequency. We estimate the frequency of any item within a sliding window of the last recent  $W'$  time units, for any  $W' \leq W$ .

**Data structure  $D_\varepsilon$ .** Let  $k$  be the smallest integer such that  $2^k \geq \frac{\log W}{\varepsilon}$ . The data structure  $D_\varepsilon$  comprises the following:

- a  $(Y, \frac{\varepsilon}{4})$ -collection for each  $Y = 2^k, 2^{k+1}, \dots, 2^h$  where  $h = \lceil \log 4B \rceil$ ;
- the  $2^k$  items with the largest timestamps among all items received so far;
- the data structure by Cormode *et al.* [3] which uses  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words of space and allows us to find an estimate  $\text{Est}_*(I_{W'})$  of  $f_*(I_{W'})$  for any  $W' \leq W$  such that

$$|\text{Est}_*(I_{W'}) - f_*(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'}).$$

The following theorem analyzes the accuracy of the estimate given by  $D_\varepsilon$  as well as the memory usage, update time and query time of  $D_\varepsilon$ . Recall that for any  $W' \leq W$ ,  $I_{W'} = [t_{\text{cur}} - W' + 1, t_{\text{cur}}]$ .

**Theorem 2.** *Given any  $W' \leq W$  at the query time, the data structure  $D_\varepsilon$  can give an estimate  $\text{Est}_a(I_{W'})$  for any item  $a$  such that*

$$|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'}),$$

using  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words of space. Furthermore, an update upon the arrival of a new item takes  $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log \log W))$  time, and a query takes  $O(\frac{1}{\varepsilon} \log W + \log \log(\varepsilon B))$  time.

*Proof.* To find an estimate of  $f_a(I_{W'})$  for any item  $a$ , we first get an estimate  $\text{Est}_*(I_{W'})$  using the data structure in [3]. Let  $Y^* = 2^i$  be the unique integer with  $\frac{\text{Est}_*(I_{W'})}{1-\varepsilon/4} \leq Y^* < \frac{2\text{Est}_*(I_{W'})}{1-\varepsilon/4}$ . Note that  $f_*(I_{W'}) \leq \frac{\text{Est}_*(I_{W'})}{1-\varepsilon/4} \leq Y^* \leq \frac{2\text{Est}_*(I_{W'})}{1-\varepsilon/4} \leq \frac{2(1+\varepsilon/4)f_*(I_{W'})}{1-\varepsilon/4} \leq 4f_*(I_{W'}) \leq 4B$ .

If  $Y^* \leq 2^k$ , then  $f_*(I_{W'}) \leq Y^* \leq 2^k$  and we can obtain the exact value of  $f_*(I_{W'})$  from the sequence containing at most  $2^k$  items received thus far with the largest timestamps. Therefore, we have  $\text{Est}_a(I_{W'}) = f_a(I_{W'})$  and hence  $|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'})$ .

Now, suppose  $Y^* > 2^k$ . Since  $f_*(I_{W'}) \leq Y^*$ , by Lemma 4, we can find an estimate  $\text{Est}_a(I_{W'})$  using the  $(Y^*, \frac{\varepsilon}{4})$ -collection such that  $|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{\varepsilon}{4} Y^*$ . Since  $Y^* \leq 4f_*(I_{W'})$ , we have  $|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \varepsilon f_*(I_{W'})$ .

**Memory usage.** A  $(Y, \frac{\varepsilon}{4})$ -collection has  $O(\frac{4 \log W}{\varepsilon})$  intervals, each of which together with its estimate needs  $O(1)$  words. Since the number of  $(Y, \frac{\varepsilon}{4})$ -collections are  $O(h-k) = O(\log(\frac{\varepsilon B}{\log W}))$ , the total space used by the collections are  $O(\frac{1}{\varepsilon} \log W \cdot \log(\frac{\varepsilon B}{\log W}))$  words. Together with the space of the data structure in [3], the total space is  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words.

**Update and Query time.** Consider an update upon the arrival of a new item. For each  $(Y, \frac{\varepsilon}{4})$ -collection, if the new item causes a batch-of-decrement step, it takes  $O(\frac{1}{\varepsilon})$  time. Otherwise, we need to update a  $\lambda$ -counter in the  $(Y, \frac{\varepsilon}{4})$ -collection, in which we need to update an interval in the list of the  $\lambda$ -counter. We can locate the interval to be updated using binary search; it takes  $O(\log(\frac{1}{\varepsilon} \log W)) = O(\log(\frac{1}{\varepsilon}) + \log \log W)$  time. Since there are  $O(\log(\frac{\varepsilon B}{\log W}))$   $(Y, \frac{\varepsilon}{4})$ -collections, the update time required by the collections are  $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log \log W))$  time. Furthermore, it takes  $O(\log(\frac{1}{\varepsilon} \log W))$  time to update the list of the  $2^k$  items. Together with the update time of the data structure in [3], which is  $O(\log(\frac{\varepsilon B}{\log W}) \cdot \log \log W)$ , the total update time is  $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log \log W))$ . Upon a query, it takes  $O(1)$  time to identify which of  $(Y, \frac{\varepsilon}{4})$ -collection or the list of the  $2^k$  items to be used for estimation; in either case, it takes  $O(\frac{1}{\varepsilon} \log W)$  time to obtain the estimate. Together with the query time of the data structure in [3], which is  $O(\frac{1}{\varepsilon} \log W + \log \log(\varepsilon B))$ , the theorem follows.  $\square$

## 5 Finding $\varepsilon$ -Approximate $\phi$ -Frequent Item Set

In this section, we apply the data structure in Section 4 to find  $\varepsilon$ -approximate  $\phi$ -frequent item set  $S$  within a sliding window covering the last recent  $W'$  time units for any window size  $W' \leq W$  given at the query time. Recall that  $I_{W'} = [t_{\text{cur}} - W' + 1, t_{\text{cur}}]$ . The set  $S$  needs to satisfy the following two requirements:

- (i)  $S$  contains every item  $a$  with  $f_a(I_{W'}) \geq \phi f_*(I_{W'})$ , and
- (ii) For any item  $b \in S$ ,  $f_b(I_{W'}) \geq (\phi - \varepsilon) f_*(I_{W'})$ .

To find such set, we maintain the following:

- Our data structure  $D_{\frac{\varepsilon}{4}}$ , which enables us to find, for any item  $a$ , an estimate  $\text{Est}_a(I_{W'})$  of  $f_a(I_{W'})$  such that  $|\text{Est}_a(I_{W'}) - f_a(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'})$ .
- The data structure by Cormode *et al.* [3] that enables us to find an estimate  $\text{Est}_*(I_{W'})$  of  $f_*(I_{W'})$  such that  $|\text{Est}_*(I_{W'}) - f_*(I_{W'})| \leq \frac{\varepsilon}{4} f_*(I_{W'})$ .

The following theorem suggests how to find an  $\varepsilon$ -approximate  $\phi$ -frequent item set, and states the space and time complexity of the data structure.

**Theorem 3.** *Let  $\epsilon \in (0, 1)$  be the error bound. With respect to the above data structure, given any threshold  $\phi \in [\varepsilon, 1]$  and any window size  $W' \leq W$  at the query time, the set*

$$S = \{a \mid \text{Est}_a(I_{W'}) \geq (\phi - \frac{\varepsilon}{2}) \text{Est}_*(I_{W'})\}$$

*is an  $\varepsilon$ -approximate  $\phi$ -frequent item set. It uses space  $O(\frac{1}{\varepsilon} \log W \log(\frac{\varepsilon B}{\log W}))$  words, an update takes  $O(\log(\frac{\varepsilon B}{\log W})(\frac{1}{\varepsilon} + \log \log W))$  time, and a query takes  $O(\frac{1}{\varepsilon} \log W + \log \log(\varepsilon B))$  time.*

*Proof.* By the estimate guarantees of the two data structures and the fact that  $\phi \leq 1$ , it can be verified that any item  $a$  with  $f_a(I_{W'}) \geq \phi f_*(I_{W'})$  is in  $S$ , and any item  $a \in S$  has  $f_a(I_{W'}) \geq (\phi - \varepsilon) f_*(I_{W'})$ . Thus,  $S$  is an  $\varepsilon$ -approximate  $\phi$ -frequent item set. The space and time complexity follow directly from Theorem 2.  $\square$

## 6 Extension for a Stream with Bounded Tardiness

Recall that in an out-of-order data stream with *tardiness*  $\tau \in [0, W]$ , any item  $(a, d)$  arriving at time  $t_{\text{cur}}$  satisfies  $d \geq t_{\text{cur}} - \tau$ ; intuitively, it guarantees that the delay of any item is at most  $\tau$ . In this section, we sketch the idea for extending our data structure to take advantage of this small delay guarantee to reduce the space requirement.

**Modification to  $\lambda$ -counter.** We say that an interval  $[p, q]$  is *short* if its length is no more than  $\tau + 1$  (i.e.,  $p - q \leq \tau$ ); otherwise it is *long*. To take advantage of the small delay guarantee, we make some minor modification to the implementation of  $\lambda$ -counter. Consider any  $\lambda$ -counter  $C_a$ . When an item  $(a, d)$  arrives at time  $t_{\text{cur}}$ , we update  $C_a$  in exactly same way as in the original implementation except for the case:  $d$  belongs to some *long* interval  $[p, q]$  in  $C_a$  and  $e_a([p, q]) = 2\lambda - 1$ . Then, we perform the update differently as follows:

**Clean-up step.** If  $q > t_{\text{cur}}$ , replace  $[p, q]$  by the two intervals  $[p, t_{\text{cur}}]$  and  $[t_{\text{cur}} + 1, q]$ , and set their estimates to  $2\lambda - 1$  and 0, respectively.

**Divide step.** Let  $s = \min\{q, t_{\text{cur}}\}$ . If  $p < t_{\text{cur}} - \tau$ , we replace  $[p, s]$  by  $[p, t_{\text{cur}} - \tau - 1]$  and  $[t_{\text{cur}} - \tau, s]$ ; otherwise, we replace  $[p, s]$  by  $[p, m]$  and  $[m + 1, s]$  where  $m = \lfloor \frac{p+s}{2} \rfloor$ . Then, we set the estimates of the two new intervals to  $\lambda$ .

Note that if we add  $[t_{\text{cur}} + 1, q]$  to the list of  $C_a$  in the clean-up step, then before the clean-up,  $[p, q]$  must be the last interval in the list. The clean-up will not increase the error; for any  $t \in [p, q]$ , if  $t \leq t_{\text{cur}}$ , then  $f_a([t, q]) = f_a([t, t_{\text{cur}}])$  and in our construction,  $E_a(t)$  does not change. Furthermore, note that for any  $t \in [p, q]$  where  $t > t_{\text{cur}}$ ,  $t$  will be in the new interval  $[t_{\text{cur}} + 1, q]$  after the clean-up, and in such case, the number of critical updates to  $t$  is reset to 0 (because  $E_a(t) = f_a([t, q]) = 0$  and the estimate for this interval is accurate).

**Lemma 5.** *Suppose that the data stream has tardiness  $\tau \in [0, W]$ . With respect to the  $\lambda$ -counter, for any  $W' \leq W$ , our new update operation guarantees that  $|E_a(t_{\text{cur}} - W' + 1) - f_a(I_{W'})| \leq 2\lambda(\log(\tau + 1) + 2)$ .*

*Proof.* It can be verified that the modified implementation still satisfies Lemma 2. To prove the error bound, it suffices to prove that for any  $t$ , the number of updates that are critical to  $t$  is at most  $(\log(\tau + 1) + 1)$ .

Suppose that initially,  $t \in [p, q]$ . If  $[p, q]$  are short, there will be at most  $\log(\tau + 1)$  critical updates to  $t$ . Suppose  $[p, q]$  is long. If  $t > t_{\text{cur}}$ , the clean-up step will add the new interval  $R = [t_{\text{cur}} + 1, q]$ , and the number of critical updates to  $t$  is reset to 0. In such case, we can treat it as if  $t$  was initially in  $R$  without any update yet, and we can repeat the argument by replacing  $[p, q]$  by  $R$ .

Suppose  $t \leq t_{\text{cur}}$ . A divide step can split  $[p, s]$  into two new intervals and  $t$  is in one of them, say  $[x, y]$ . It can be verified that either (i)  $[x, y]$  is short, and there will be at most  $\log(\tau + 1)$  additional critical updates to  $t$ , or (ii)  $y < t_{\text{cur}} - \tau$  and there will be no more update to  $[x, y]$  because the tardiness is  $\tau$ . Thus, the total number of critical updates to  $t$  is at most  $\log(\tau + 1) + 1$ .  $\square$

**Finding frequent item set.** By setting  $\lambda = \frac{\delta Y}{10(\log(\tau + 1) + 2)}$  and using the same analysis in previous sections, we have the following theorem.

**Theorem 4.** *There is a data structure that finds  $\varepsilon$ -approximate  $\phi$ -frequent item set for data streams with tardiness  $\tau$  using  $O(\frac{1}{\varepsilon} \log \tau \log(\frac{\varepsilon B}{\log \tau}))$  words of space. Furthermore, an update takes  $O(\log(\frac{\varepsilon B}{\log \tau})(\frac{1}{\varepsilon} + \log \log \tau))$  time and a query takes  $O(\frac{1}{\varepsilon} \log \tau + \log \log(\varepsilon B))$  time.*

## References

1. Arasu, A., Manku, G.: Approximate counts and quantiles over sliding windows. In: Proc. PODS, pp. 286–296 (2004)
2. Busch, C., Tirthapua, S.: A deterministic algorithm for summarizing asynchronous streams over a sliding window. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 465–476. Springer, Heidelberg (2007)

3. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: Proc. PODS, pp. 89–98 (2008)
4. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. *SIAM Journal on Computing* 31(6), 1794–1813 (2002)
5. Demaine, E., Lopez-Ortiz, A., Munro, J.: Frequency estimation of internet packet streams with limited space. In: Möhring, R.H., Raman, R. (eds.) *ESA 2002*. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
6. Karp, R., Shenker, S., Papadimitriou, C.: A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Systems* 28(1), 51–55 (2003)
7. Lee, L.K., Ting, H.F.: Maintaining significant stream statistics over sliding windows. In: Proc. SODA, pp. 724–732 (2006)
8. Lee, L.K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proc. PODS, pp. 290–297 (2006)
9. Misra, J., Gries, D.: Finding repeated elements. *Science of Computer Programming* 2(2), 143–152 (1982)
10. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: new aggregation techniques for sensor networks. In: Proc. SenSys, pp. 239–249 (2004)
11. Tirthapura, S., Xu, B., Busch, C.: Sketching asynchronous streams over a sliding window. In: *PODC*, pp. 82–91 (2006)