Algorithms **2011**, *3*, 200-222; doi:10.3390/a4030200

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Approximating Frequent Items in Asynchronous Data Stream over a Sliding Window

Hing-Fung Ting [1,*], Lap-Kei Lee [2], Ho-Leung Chan [1] and Tak-Wah Lam [1]

[1] Department of Computer Science, University of Hong Kong, Pokfulam, Hong Kong, China; E-Mails: hlchan@cs.hku.hk (H.-L.C.); twlam@cs.hku.hk (T.-W.L.)

[2] MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation), Department of Computer Science, Aarhus University, Aarhus C DK-8000, Denmark; E-Mail: lklee@madalgo.au.dk

* Author to whom correspondence should be addressed; E-Mail: hfting@cs.hku.hk; Tel.:+852-2859-8944; Fax: +852-2549-7908.

Received: 23 June 2011; in revised form: 23 June 2011 / Accepted: 10 September 2011 / Published: 22 September 2011

Abstract: In an asynchronous data stream, the data items may be out of order with respect to their original timestamps. This paper studies the space complexity required by a data structure to maintain such a data stream so that it can approximate the set of frequent items over a sliding time window with sufficient accuracy. Prior to our work, the best solution is given by Cormode *et al.* [1], who gave an $O\big(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W})\min\{\log W, \frac{1}{\epsilon}\}\log|U|\big)$-space data structure that can approximate the frequent items within an ϵ error bound, where W and B are parameters of the sliding window, and U is the set of all possible item names. We gave a more space-efficient data structure that only requires $O\big(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W})\log\log W\big)$ space.

Keywords: asynchronous data streams; frequent items; sliding window; space complexity

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 [2–5]. Recent interest has been shifted from the statistics of the whole data stream to that of a sliding window of recent data [6–9]. In most applications, the amount of data in a window is gigantic when compared with the amount of memory available in the processing units. It is impossible to store all the data and then find the exact frequent items. Existing research has focused on designing space-efficient data structures to support finding the *approximate* frequent items. The key concern is how to minimize the space so as to achieve a required level of accuracy.

1.1. 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 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* (a.k.a. *out-of-order data stream*) [1,10,11] can be considered as a sequence $(a_1, t_1), (a_2, t_2), (a_3, t_3), \ldots$, where a_i is the name of a data item chosen from a fixed universe U, and t_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, and it is possible that more than one data item has the same timestamp.

1.2. 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 (or 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 [6] were the first to study approximating frequent items over a sliding window under the synchronous model, in which data items arrive in non-decreasing order of timestamps. The space complexity of their data structure is $O(\frac{1}{\epsilon}(\log \frac{1}{\epsilon})^2 \log(\epsilon B))$, where ϵ 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 [7] to $O(\frac{1}{\epsilon} \log(\epsilon B))$ space. Recently, Cormode *et al.* [1] initiated the study of frequent items under the asynchronous model, and gave a solution with space complexity $O(\frac{1}{\epsilon} \log W \log(\frac{\epsilon B}{\log W}) \min\{\log W, \frac{1}{\epsilon}\} \log |U|)$, where U is the set of possible item names. Later, Cormode *et al.* [12] gave a hashing-based randomized solution using $O(\frac{1}{\epsilon^2} \log |U|)$ space. The space complexity is quadratic in $\frac{1}{\epsilon}$, which is less preferred, but that is a general solution that can solve other problems like finding the sum and quantiles.

The earlier work on asynchronous data stream focused on a relatively simpler problem called ϵ-approximate basic counting [10,11]. Cormode *et al.* [1] improved the space complexity of basic counting to $O(\frac{1}{\epsilon} \log W \log(\frac{\epsilon B}{\log W}))$. Notice that under the synchronous model, the best data structure requires $O(\frac{1}{\epsilon} \log(\epsilon B))$ space [9]. 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.* [1] has space complexity way bigger than that of the best synchronous result, which is

$O(\frac{1}{\epsilon}\log(\epsilon B))$ [7]. This motivates us to study more space-efficient solutions for approximating frequent items in the asynchronous model.

1.3. Formal Definition of Approximate Frequent Item Set

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 arrived items named a with timestamps falling into I. Define $f_*(I) = \sum_{a\in U} f_a(I)$ to be the total number of all arrived items with timestamps within I.

Given a user-specified error bound ϵ and a window size W, we want to maintain a data structure to answer any ϵ-approximate frequent item set query for any sub-window (specified at query time), which is in the form (ϕ, W') where $\phi \in [\epsilon, 1]$ is the required threshold and $W' \leq W$ is the sub-window size. Suppose that $\tau_{\mathtt{cur}}$ is the current time. The answer to such a query is a set S of item names satisfying the following two conditions:

(C1) S contains every item a whose frequency in interval $I = [\tau_{\mathtt{cur}} - W' + 1, \tau_{\mathtt{cur}}]$ is at least $\phi f_*(I)$, *i.e.*, $f_a(I) \geq \phi f_*(I)$.

(C2) For any item a in S, its frequency in interval I is at least $(\phi - \epsilon) f_*(I)$, *i.e.*, $f_a(I) \geq (\phi - \epsilon) f_*(I)$.

The set S is also called an ϵ-approximate ϕ-frequent item set. For example, assume $\epsilon = 1\%$, then the query $(10\%, 10,000)$ would return all items whose frequencies in the last $10,000$ time units are each at least 10% of the total item count, plus possibly some other items with frequency at least 9% of the total count.

1.4. Our Contribution

This paper gives a more space-efficient data structure for answering any ϵ-approximate frequent item set query. Our data structure uses $O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W})\log\log W)$ words, which is significantly smaller than the one given by Cormode *et al.* [1] (see Table 1). Furthermore, this space complexity is larger than the best synchronous solution by only a factor of $O(\log W \log\log W)$, which is close to the expected gap of $O(\log W)$. Similar to existing data structures for this problem, it takes time linear to the data structure's size to answer an ϵ-approximate frequent item set query. Furthermore, it takes $O(\log(\frac{\epsilon B}{\log W})(\log\frac{1}{\epsilon} + \log\log W))$ time to modify the data structure for a new data item. Occasionally, we might need to clean up some old data items that are no longer significant to the approximation; in the worst case, this takes time linear to the size of the data structure, and thus is no bigger than the query time. As a remark, the solution of Cormode *et al.* [1] requires $O(\log(\frac{\epsilon B}{\log W})\log W \log\log|U|)$ time for an update.

Table 1. The space complexity for answering ϵ-approximate frequent item set query in a sliding time window. Results from this paper are marked with [†]. Note that we assume $B \geq \frac{1}{\epsilon}\log W$; otherwise, we can always store all items in the window for exact answer, using $O(\frac{1}{\epsilon}\log W)$ words. Similarly, for the result with tardiness, we assume $B \geq \frac{1}{\epsilon}\log d_{\max}$.

	Space Complexity (words)
Synchronous [7]	$O(\frac{1}{\epsilon}\log(\epsilon B))$
Asynchronous [1]	$O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W}) \min\{\log W, \frac{1}{\epsilon}\} \log \lvert U \rvert)$
Asynchronous [†]	$O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W}) \log\log W)$
Asynchronous with tardiness [†]	$O(\frac{1}{\epsilon}\log d_{\max} \log(\frac{\epsilon B}{\log d_{\max}}) \log\log d_{\max})$

In the asynchronous model, if a data item has a delay more than W time units, it can be discarded immediately when it arrives. In many applications, the delay is usually small. 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 $d_{\max}$ if a data item created at time t must arrive at the stream no later than time $t + d_{\max}$. If we set $d_{\max} = 0$, the model becomes the synchronous model. If we allow $d_{\max} \geq W$, this is in essence the asynchronous model studied above. We adapt our data structure to take advantage of small tardiness such that when $d_{\max}$ is small, it uses smaller space (see Table 1) and support faster update time (which is $O(\log(\frac{\epsilon B}{\log d_{\max}})(\log\frac{1}{\epsilon} + \log\log d_{\max})))$. In particular, when $d_{\max} = \Theta(1)$, the size and update time of our data structure match those of the best data structure for synchronous data stream.

Remark. This paper is a corrected version of a paper with the same title in WAOA 2009 [13]; in particular, the error bound on the estimates was given incorrectly before and is fixed in this version.

1.5. Technical Digest

To solve the frequent item set problem, we need to estimate the frequency of any item with relative error $\epsilon f_*(I)$ where $I = [\tau_{\texttt{cur}} - W + 1, \tau_{\texttt{cur}}]$ is the interval covered by the sliding window. To this end, we first propose a simple data structure for estimating the frequency of a fixed item over the sliding window. Then, we adapt a technique of Misra and Gries [14] to extend our data structure to handle any item. The result is an $O(f_*(I))/\lambda)$-space data structure that allows us to obtain an estimate for any item with an error bound of about $\lambda \log W$. Here λ is a design parameter. To ensure $\lambda \log W$ to be no greater than $\epsilon f_*(I)$, we should set $\lambda \leq \epsilon f_*(I)/\log W$. Since $f_*(I)$ can be as small as $\Theta(\frac{1}{\epsilon}\log W)$ (the case for smaller $f_*(I)$ can be handled by brute-force), we need to be conservative and set λ to some constant. But then the size of the data structure can be $\Theta(B)$ because $f_*(I)$ can be as large as B. To reduce space, we introduce a multi-resolution approach. Instead of using one single data structure, we maintain a collection of $O(\log B)$ copies of our data structure, each uses a distinct, carefully chosen parameter λ so that it could estimate the frequent item set with sufficient accuracy when $f_*(I)$ is in a particular range. The resulting data structure uses $O(\frac{1}{\epsilon}\log W \log B)$ space.

Unfortunately, a careful analysis of our data structure reveals that in the worst case, it can only guarantee estimates with an error bound of $\epsilon f_*(H \cup I)$ where $H = [\tau_{\texttt{cur}} - 2W + 1, \tau_{\texttt{cur}} - W]$, not the required $\epsilon f_*(I)$. The reason is that the error of its estimates over I depend on the number of updates made during I, and unlike synchronous data stream, this number for asynchronous data stream can be significantly larger than $f_*(I)$. For example, at time $\tau_{\texttt{cur}} - W + 1$, there may still be many new items (a, u) with timestamps $u \in H$, for which we must update our data structure to get good estimates when the sliding window is at earlier positions. Indeed, the number of updates during I can be as large as $f_*(H \cup I)$, and this gives an error bound of $\epsilon f_*(H \cup I)$.

To reduce the error bound to $\epsilon f_*(I)$, we introduce a novel algorithm to split the data structure into independent smaller ones at appropriate times. For example, at time $\tau_{\texttt{cur}} - W + 1$, we can split our data structure into two smaller ones D_H and D_I, and we will only update D_H for items (a, u) with $u \in H$ and update D_I for those with $u \in I$. Then, when we need to find an estimate on I at time $\tau_{\texttt{cur}}$, we only need to consult D_I, and the number of updates made to it is $f_*(I)$. In this paper, we develop sophisticated procedures to decide when and how to split the data structure so as to enable us to get good enough estimates when sliding window moves continuously. The resulting data structure has size $O(\frac{1}{\epsilon}(\log W)^2 \log(\frac{\epsilon B}{\log W}))$. Then, we further make the data structure adaptive to the input size, allowing us to reduce the space to $O(\frac{1}{\epsilon}(\log\log W)\log W \log(\frac{\epsilon B}{\log W}))$.

2. Preliminaries

Our data structures for the frequent item set problem depends on data structures for the following two related data stream problems. Let $0 < \epsilon < 1$ be any real number, and $\tau_{\texttt{cur}}$ be the current time.

- The ϵ-*approximate basic counting* problem asks for data structure that allows us to obtain, for any interval $I = [\tau_{\texttt{cur}} - W' + 1, \tau_{\texttt{cur}}]$ where $W' \leq W$, an estimate $\hat{f}_*(I)$ of $f_*(I)$ such that $|\hat{f}_*(I) - f_*(I)| \leq \epsilon f_*(I)$.
- The ϵ-*approximate counting* problem asks for data structure that allows us to obtain, for any item a and any interval $I = [\tau_{\texttt{cur}} - W' + 1, \tau_{\texttt{cur}}]$ where $W' \leq W$, an estimate $\hat{f}_a(I)$ of $f_a(I)$ such that $|\hat{f}_a(I) - f_a(I)| \leq \epsilon f_*(I)$.

As mentioned in Section 1, Cormode *et al.* [1] gave an $O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W}))$-space data structure $\mathcal{B}_\epsilon$ for solving the ϵ-approximate basic counting problem. In this paper, we give an $O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W})\log\log W)$-space data structure $\mathcal{D}_\epsilon$ for solving the harder ϵ-approximate counting problem. The theorem below shows how to use these two data structures to answer ϵ-approximate frequent item set query.

Theorem 1 *Let $\epsilon_o = \epsilon/4$. Given $\mathcal{B}_{\epsilon_o}$ and $\mathcal{D}_{\epsilon_o}$, we can answer any ϵ-approximate frequent item set query. The total space required is $O(\frac{1}{\epsilon}\log W \log(\frac{\epsilon B}{\log W})\log\log W)$.*

Proof The space requirement is obvious. Consider any ϵ-approximate frequent item set query (ϕ, W') where $\epsilon \leq \phi \leq 1$ and $W' \leq W$. Let $I = [\tau_{\texttt{cur}} - W' + 1, \tau_{\texttt{cur}}]$. Since $\epsilon_o = \epsilon/4$, the estimates given by $\mathcal{B}_{\epsilon_o}$ satisfy $|\hat{f}_*(I) - f_*(I)| \leq \frac{\epsilon}{4} f_*(I)$, and for any item a, the estimates given by $\mathcal{D}_{\epsilon_o}$ satisfy $|\hat{f}_a(I) - f_a(I)| \leq \frac{\epsilon}{4} f_*(I)$. To answer the query (ϕ, W'), we return the set

$$S_\phi = \{a \mid \hat{f}_a(I) \geq (\phi - \tfrac{\epsilon}{2})\hat{f}_*(I)\}$$

which satisfies the required conditions (C1) and (C2) because

- for any item a with $f_a(I) \geq \phi f_*(I)$, $\hat{f}_a(I) \geq f_a(I) - \frac{\epsilon}{4} f_*(I) \geq (\phi - \frac{\epsilon}{4}) f_*(I) \geq (\phi - \frac{\epsilon}{4})(\frac{1}{1+\frac{\epsilon}{4}})\hat{f}_*(I) \geq (\phi - \frac{\epsilon}{4})(1 - \frac{\epsilon}{4})\hat{f}_*(I) \geq (\phi - \frac{\epsilon}{2})\hat{f}_*(I)$, and $a \in S_\phi$; thus (C1) is satisfied, and
- for every $a \in S_\phi$, we have $f_a(I) \geq \hat{f}_a(I) - \frac{\epsilon}{4} f_*(I) \geq (\phi - \frac{\epsilon}{2})\hat{f}_*(I) - \frac{\epsilon}{4} f_*(I) \geq (\phi - \frac{\epsilon}{2})(1 - \frac{\epsilon}{4}) f_*(I) - \frac{\epsilon}{4} f_*(I) \geq (\phi - \epsilon) f_*(I)$; thus (C2) is satisfied.

The building block of $\mathcal{D}_\epsilon$ is a data structure that counts items over some fixed interval (instead of the sliding window). For any interval $I = [\ell_I, r_I]$ of size W, Theorem 4 in Section 4 gives a data structure $\mathcal{D}_{I,\epsilon}$ that uses $O(\frac{1}{\epsilon} \log W \log(\frac{\epsilon B}{\log W}) \log\log W)$ space, supports $O(\log(\frac{\epsilon B}{\log W}) \cdot (\log \frac{1}{\epsilon} + \log\log W))$ update time, and enables us to obtain, for any item a and any time $t \in I$, an estimate $\hat{f}_a([t, r_I])$ of $f_a([t, r_I])$ such that

$$|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq \epsilon f_*([t, r_I]) \tag{1}$$

Given $\mathcal{D}_{I_1,\epsilon}, \mathcal{D}_{I_2,\epsilon}, \ldots$ where $I_i = [(i-1)W + 1, iW]$, we can obtain, for any $W' \leq W$, an estimate $\hat{f}_a([s, \tau_{\text{cur}}])$ of $f_a([s, \tau_{\text{cur}}])$ where $s = \tau_{\text{cur}} - W' + 1$ as follows.

- Let I_i and I_{i+1} be the intervals such that $[s, \tau_{\text{cur}}] \subset I_i \cup I_{i+1}$.
- Use $\mathcal{D}_{I_i,\epsilon}$ to get an estimate $\hat{f}_a([s, iW])$ of $f_a([s, iW])$, and $\mathcal{D}_{I_{i+1},\epsilon}$ an estimate $\hat{f}_a([iW+1, (i+1)W])$ of $f_a([iW+1, (i+1)W])$.
- Our estimate $\hat{f}_a([s, \tau_{\text{cur}}]) = \hat{f}_a([s, iW]) + \hat{f}_a([iW+1, (i+1)W])$.

By Equation (1), we have

$$|\hat{f}_a([s, iW]) - f_a([s, iW])| \leq \epsilon f_*([s, iW]) \tag{2}$$

and

$$|\hat{f}_a([iW+1, (i+1)W]) - f_a([iW+1, (i+1)W])| \leq \epsilon f_*([iW+1, (i+1)W]) \tag{3}$$

Observe that any item that arrives at or before the current time τ_{cur} must have timestamp no greater than τ_{cur}; hence $f_a([iW+1, (i+1)W]) = f_a([iW+1, \tau_{\text{cur}}])$ and $f_*([iW+1, (i+1)W]) = f_*([iW+1, \tau_{\text{cur}}])$, and Equation (3) is equivalent to

$$|\hat{f}_a([iW+1, (i+1)W]) - f_a([iW+1, \tau_{\text{cur}}])| \leq \epsilon f_*([iW+1, \tau_{\text{cur}}]) \tag{4}$$

Adding Equations (2) and (4), we conclude $|\hat{f}_a([s, \tau_{\text{cur}}]) - f_a([s, \tau_{\text{cur}}])| \leq \epsilon f_*([s, \tau_{\text{cur}}])$, as required.

Our data structure $\mathcal{D}_\epsilon$ is just the collection of $\mathcal{D}_{I_1,\epsilon}, \mathcal{D}_{I_2,\epsilon}, \ldots$. Note that we only need to physically store in $\mathcal{D}_\epsilon$ the data structures $\mathcal{D}_{I_i,\epsilon}$ and $\mathcal{D}_{I_{i+1},\epsilon}$ where $[\tau_{\text{cur}} - W + 1, \tau_{\text{cur}}] \subseteq I_i \cup I_{i+1}$. The intervals of the earlier ones will no longer be covered by the sliding window and the corresponding $\mathcal{D}_{I,\epsilon}$'s can be thrown away. Together with Theorem 4, we have the following theorem.

Theorem 2 *The data structure $\mathcal{D}_\epsilon$ solves the ϵ-approximate counting problem. The space usage is $O(\frac{1}{\epsilon} \log W \log(\frac{\epsilon B}{\log W}) \log\log W)$ and it supports $O(\log(\frac{\epsilon B}{\log W}) \cdot (\log \frac{1}{\epsilon} + \log\log W))$ update time.*

3. A Simple Data Structure For Frequency Estimation

Let $I = [\ell_I, r_I]$ be any interval of size W. To simplify notation, we assume that W is a power of 2, so that $\log W$ is an integer and we can avoid the floor or the ceiling functions. In this section, we describe a simple data structure $\mathcal{C}_{I,\lambda,\kappa}$ that enables us to obtain, for any item a, a good estimate of a's frequency over I. The parameters λ and κ determine its accuracy and space usage. However, its accuracy is not enough for answering any ϵ-approximate frequent item set query. We will explain how to improve the accuracy in the next section.

Roughly speaking, $\mathcal{C}_{I,\lambda,\kappa}$ is a set of queues $Q^a_{I,\lambda}$, *i.e.*, $\mathcal{C}_{I,\lambda,\kappa} = \{Q^a_{I,\lambda} \mid a \in U\}$. For an item a, the queue $Q^a_{I,\lambda}$ keeps track of the occurrences of a in I. Each node N in $Q^a_{I,\lambda}$ is associated with an interval $i(N)$, a value $v(N)$, and a debit $d(N)$; $v(N)$ counts the number of arrived items (a, u) with $u \in i(N)$, and $d(N)$ is for implementing a space reduction technique. Initially, $Q^a_{I,\lambda}$ has only one node N with $i(N) = I$, and $v(N) = d(N) = 0$. In general, $Q^a_{I,\lambda}$ is a queue $\langle N_1, N_2, \ldots, N_k\rangle$ of nodes whose intervals form a partition of I, *i.e.*,

$$\langle i(N_1), i(N_2), \ldots, i(N_k)\rangle = \langle [p_1, q_1], [p_2, q_2], \ldots, [p_k, q_k]\rangle$$

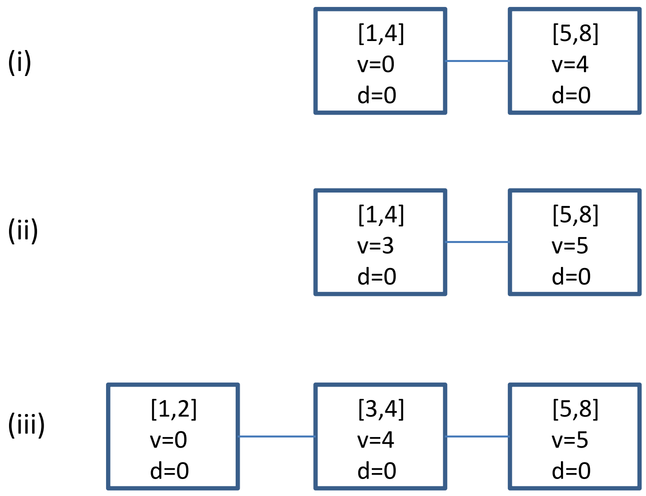
where $q_{i-1} + 1 = p_i \leq q_i$ and $\bigcup_{1\leq i\leq k}[p_i, q_i] = I$. When an item (a, u) with $u \in I$ arrives, we update $Q^a_{I,\lambda}$ as follows.

$Q^a_{I,\lambda}$.Debit( )

```
1: find the unique node $N$ in $Q^a_{I,\lambda}$ with $u \in i(N) = J = [p, q]$;
2: increase the value of $N$ by 1, i.e., $v(N) = v(N) + 1$;
3: if ($|J| > 1$ and $\lambda$ units have been added to $v(N)$ since $J$ is assigned to $i(N)$) then
4:     /* refine $J$ */
5:     create a new node $N'$ and insert it to the left of $N$;
6:     let $i(N') = [p, m]$, $i(N) = [m + 1, q]$ where $m = \lfloor (p + q)/2 \rfloor$;
7:     let $v(N') = 0$ and $d(N') = 0$;
8:     /* we make no change to $v(N)$ and $d(N)$ */
9: end if
```

Figure 1 gives an example on how $Q^a_{I,\lambda}$ is updated using the procedure.

Figure 1. Suppose that $\lambda = 4$. (i) shows the queue $Q^a_{I,\lambda}$ before the arrivals of items $(a,1),(a,2),(a,3),(a,8)$; (ii) is the resulting queue after the updates for these items; (iii) shows that after the arrival of another item $(a,1)$, the first node in (ii) is updated and refined.



Obviously, a direct implementation of $\mathcal{C}_{I,\lambda,\kappa}$ uses too much space. We now extend a technique of Misra and Gries [14] to reduce the space requirement. For any $Q^a_{I,\lambda}$, we say that $Q^a_{I,\lambda}$ is *trivial* if the queue contains only a single node N with (i) $i(N) = I$, and (ii) $v(N) = d(N) = 0$. Every queue in $\mathcal{C}_{I,\lambda,\kappa}$ is trivial initially. The key for reducing the space complexity of $\mathcal{C}_{I,\lambda,\kappa}$ is to maintain the following invariant throughout the execution:

$(*)$ There are at most κ non-trivial queues in $\mathcal{C}_{I,\lambda,\kappa}$.

We call κ the *capacity* of $\mathcal{C}_{I,\lambda,\kappa}$. The invariant helps us save space because we do not need to store trivial queues physically in memory. To maintain $(*)$, each queue $Q^a_{I,\lambda}$ supports the following procedure, which is called only when $v(Q^a_{I,\lambda})$, the total values of the nodes in $Q^a_{I,\lambda}$, is strictly greater than $d(Q^a_{I,\lambda})$, the total debits of the nodes in $Q^a_{I,\lambda}$.

```
$Q^a_{I,\lambda}$.Debit( )
1: if ($v(Q^a_{I,\lambda}) \le d(Q^a_{I,\lambda})$) then
2:     return error;
3: else
4:     find an arbitrary node $N$ of $Q^a_{I,\lambda}$ with $v(N) > d(N)$;
5:     /* such a node must exist because $v(Q^a_{I,\lambda}) > d(Q^a_{I,\lambda})$ */
6:     $d(N) = d(N) + 1$;
7: end if
```

Note from the implementation of Debit( ) that $v(Q^a_{I,\lambda})$ is always no smaller than $d(Q^a_{I,\lambda})$, and for each node N of $Q^a_{I,\lambda}$, $v(N) \ge d(N)$. Furthermore, if $v(Q^a_{I,\lambda}) = d(Q^a_{I,\lambda})$, then $v(N) = d(N)$ for every node N in $Q^a_{I,\lambda}$. To maintain $(*)$, $\mathcal{C}_{I,\lambda,\kappa}$ processes a newly arrived item (a,u) with $u \in I$ as follows.

$\mathcal{C}_{I,\lambda,\kappa}$.Process($(a,u)$)

```
1: update $Q^a_{I,\lambda}$ by calling $Q^a_{I,\lambda}$.Update($(a,u)$);
2: if (after the update the number of non-trivial queues becomes $\kappa$) then
3:     for each $Q^x_{I,\lambda}$ with $v(Q^x_{I,\lambda}) > d(Q^x_{I,\lambda})$ do $Q^x_{I,\lambda}$.Debit();
4:     for each non-trivial queues $Q^x_{I,\lambda}$ with $v(Q^x_{I,\lambda}) = d(Q^x_{I,\lambda})$ do
5:         delete all nodes of $Q^x_{I,\lambda}$ and make it a trivial queue;
6:     /* Note that each deleted node $N$ satisfies $v(N) = d(N)$. */
7: end if
```

It is easy to see that Invariant $(*)$ always holds: Initially the number m of non-trivial queues is zero, and m increases only when Process((a,u)) is on some trivial $Q^a_{I,\lambda}$; in such case $v(Q^a_{I,\lambda})$ becomes 1 and $d(Q^a_{I,\lambda})$ remains 0. If m becomes κ after this increase, we will debit, among other queues, $Q^a_{I,\lambda}$ and its $d(Q^a_{I,\lambda})$ becomes 1 too. It follows that $v(Q^a_{I,\lambda}) = d(Q^a_{I,\lambda})$, and Lines 4–5 will make $Q^a_{I,\lambda}$ trivial and m becomes less than κ again.

We are now ready to define $\mathcal{C}_{I,\lambda,\kappa}$'s estimate $\hat{f}_a([t,r_I])$ of $f_a([t,r_I])$ and analyze its accuracy. We need some definitions. For any interval $J = [p,q]$ and any $t \in I$, we say that J *covers* t if $t \in [p,q]$, is *to the right* of t if $t < p$, and is *to the left* of t otherwise. For any item a and any $t \in I = [\ell_I, r_I]$, $\mathcal{C}_{I,\lambda,\kappa}$'s estimate of $f_a([t,r_I])$ is

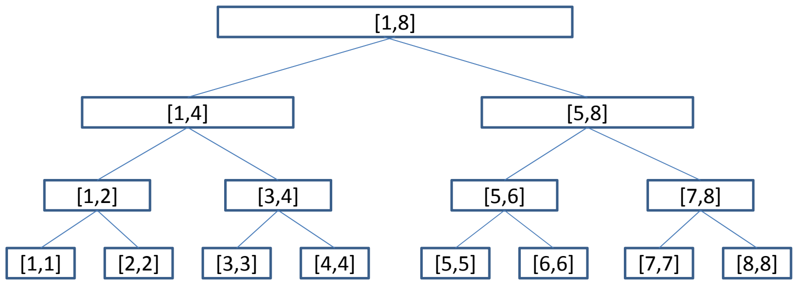
> $\hat{f}_a([t,r_I])$ = the value sum of the nodes N currently in $Q^a_{I,\lambda}$ whose $i(N)$ covers or is to the right of t.

For example, in Figure 1, after the update of the last item $(a,1)$, we can obtain the estimate $\hat{f}_a([2,8]) = 0 + 4 + 5 = 9$.

Given any node N of $Q^a_{I,\lambda}$, we say that N is monitoring a over J, or simply N is monitoring J if $i(N) = J$. Note that a node may monitor different intervals during different periods of execution, and the size of these intervals are monotonically decreasing. Observe that although there are about $W^2/2$ possible sub-intervals of size-W interval I, there are only about $2W$ of them that would be monitored by some nodes: there is only one such interval of size W, namely $I = [\ell_I, r_I]$, which gives birth to two such intervals of size $W/2$, namely $[\ell_I, m]$ and $[m+1, r_I]$ where $m = \lfloor(\ell_I + r_I)/2\rfloor$, and so on. We call these $O(W)$ intervals *interesting intervals*. For any two interesting intervals J and H such that $J \subset H$, we say that J is a *descendant* of H, and H is an *ancestor* of J. Figure 2 shows all the interesting intervals for $I = [1,8]$, as well as their ancestor-descendant relationship. The following important fact is easy to verify by induction.

Fact 1 *Any two interesting intervals J and H do not cross, although one can contain another, i.e., either $J \subset H$, or $H \subset J$, or $J \cap H = \emptyset$. Furthermore, any interesting interval has at most* $\log W$ *ancestors.*

Figure 2. Interesting intervals for $I = [1, 8]$.



For any node N, let $\mathcal{I}(N)$ be the set of intervals that have been monitored by N so far. The following fact can be verified from the update procedure.

Fact 2 *Consider a node N in $Q^a_{I,\lambda}$, where $i(N) = J$.*

- *If J covers or is to the right of t, then all intervals in $\mathcal{I}(N)$ cover or are to the right of t.*
- *If J is to the left of t, then all intervals in $\mathcal{I}(N)$ are to the left of t.*

We say that N *covers or is to the right of* t if the intervals in $\mathcal{I}(N)$ cover or are to the right of t; otherwise, N is *to the left of* t. For any queue $Q^a_{I,\lambda}$, let $\mathsf{alive}(Q^a_{I,\lambda})$ be the set of nodes currently in $Q^a_{I,\lambda}$, $\mathsf{dead}(Q^a_{I,\lambda})$ be those nodes of $Q^a_{I,\lambda}$ that have already been deleted (because of Line 5 of the procedure Process( )), and $\mathsf{node}(Q^a_{I,\lambda}) = \mathsf{alive}(Q^a_{I,\lambda}) \cup \mathsf{dead}(Q^a_{I,\lambda})$. Note that the estimate $\hat{f}_a([t, r_I])$ is the value sum of the nodes in $\mathsf{alive}(Q^a_{I,\lambda})$ that cover or are to the right of t. For simplicity, we need to express it more succinctly. Let

$$\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa}) = \bigcup \{\mathsf{alive}(Q^a_{I,\lambda}) \mid Q^a_{I,\lambda} \in \mathcal{C}_{I,\lambda,\kappa}\}$$

be the set of nodes currently in $\mathcal{C}_{I,\lambda,\kappa}$. Define $\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})$ and $\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})$ similarly. For any item a and any subset $X \subseteq \mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})$, let X^a be the set of nodes in X that are monitoring a (and thus are the nodes from $Q^a_{I,\lambda}$). For any $t \in I$, let $X_{\geq t}$ denote the set of nodes in X that cover or are to the right of t. Define $v(X) = \sum_{N \in X} v(N)$ and $d(X) = \sum_{N \in X} d(N)$. Then, $\hat{f}_a([t, r_I])$ can be expressed as follows:

$$\hat{f}_a([t, r_I]) = v(\mathsf{alive}(Q^a_{I,\lambda})_{\geq t}) = v(\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t})$$

The following theorem analyzes its accuracy, as well as gives the size of $\mathcal{C}_{I,\lambda,\kappa}$.

Lemma 3 *For any $t \in I$, $f_a([t, r_I]) - \frac{1}{\kappa} f_*(I) \leq \hat{f}_a([t, r_I]) \leq f_a([t, r_I]) + \lambda \log W$. Furthermore, $\mathcal{C}_{I,\lambda,\kappa}$ has size $O(f_*(I)/\lambda + \kappa)$ words.*

Proof Recall that $\hat{f}_a([t, r_I]) = v(\mathsf{alive}(Q^a_{I,\lambda})_{\geq t})$. Consider any node $N \in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}$. Note that $v(N) = \sum_{J \in \mathcal{I}(N)} v_{\mathsf{add}}(N, J)$ where $v_{\mathsf{add}}(N, J)$ is the value added to $v(N)$ during the period when $i(N) = J$. By Fact 2, we can divide it as $v(N) = \sum\{v_{\mathsf{add}}(N, J) \mid J \text{ covers } t\} + \sum\{v_{\mathsf{add}}(N, J) \mid J \text{ is to the right of } t\}$. It follows that

$$\begin{aligned} v(\mathsf{alive}(Q^a_{I,\lambda})_{\geq t}) &= \textstyle\sum_{N \in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}} v(N) \\ &= \textstyle\sum_{N \in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}} \sum\{v_{\mathsf{add}}(N, J) \mid J \text{ covers } t\} + \\ &\qquad \textstyle\sum_{N \in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}} \sum\{v_{\mathsf{add}}(N, J) \mid J \text{ is to the right of } t\} \end{aligned} \quad (5)$$

Note that $\sum_{N\in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}} \sum \{v_{\mathsf{add}}(N,J) \mid J \text{ is to the right of } t\} \leq f_a([t,r_I])$, because if an arrived item (a,u) causes an increase of $v_{\mathsf{add}}(N,J)$ for some J that is to the right of t, then u must be in $[t,r_I]$. By Equation (5), to show the second inequality of the lemma, it suffices to show that $S_o = \sum_{N\in \mathsf{alive}(Q^a_{I,\lambda})_{\geq t}} \sum \{v_{\mathsf{add}}(N,J) \mid J \text{ covers } t\} = v_{\mathsf{add}}(N_1,J_1) + v_{\mathsf{add}}(N_2,J_2) + \cdots + v_{\mathsf{add}}(N_k,J_k)$ is no greater than $\lambda \log W$, as follows.

Without loss of generality, suppose $|J_1| \geq |J_2| \geq \cdots \geq |J_k|$. It can be verified that once an interval J is assigned to a node, it will not be assigned to other nodes; thus the J_i's are distinct. Furthermore, note that for $1 \leq i < k$, $J_k \subset J_i$ because (i) t is in both J_i and J_k; (ii) J_k is the smallest interval; and (iii) interesting intervals do not cross; thus J_k is a descendant of J_i, and together with Fact 1, $k \leq \log W$. By Line 3 of the procedure Update( ), $v_{\mathsf{add}}(N_i,J_i) \leq \lambda$ for $1 \leq i \leq k$. It follows that $S_o \leq \lambda \log W$.

For the first inequality of the lemma, it is clearer to use $\hat{f}_a([t,r_I]) = v(\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t})$. Note that every arrived item (a,u) with $u \in [t,r_I]$ increments the value of some node in $\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}$; thus $f_a([t,r_I]) \leq v(\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t})$ and

$$f_a([t,r_I]) - v(\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) \leq v(\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) - v(\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) = v(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t})$$

From Lines 4–6 of the procedure Process( ), when we delete a node N, $v(N) = d(N)$. Hence, $v(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) = d(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t})$, which is equal to the total number of debit operations made to these dead nodes. Since whenever we make a debit operation to $Q^a_{I,\lambda}$, we will make a debit operation to $\kappa - 1$ other queues,

$$\kappa \cdot d(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) \leq d(\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})) \leq v(\mathsf{node}(\mathcal{C}_{I,\lambda,\kappa})) = f_*(I) \tag{6}$$

In summary, we have $f_a([t,r_I]) - \hat{f}_a([t,r_I]) = f_a([t,r_I]) - v(\mathsf{alive}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) \leq v(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) = d(\mathsf{dead}(\mathcal{C}_{I,\lambda,\kappa})^a_{\geq t}) \leq f_*(I)/\kappa$, and the first inequality of the lemma follows.

For the space, we say that a node is born-rich if it is created because of Line 5 of the procedure Update( ) (and thus has λ items under its belt); otherwise it is born-poor. Obviously, there are at most $f_*(I)/\lambda$ born-rich nodes. For born-poor nodes, we need to store at most κ of them because every queue has one born-poor node (the rightmost one), and we only need to store at most κ non-trivial queues; the space bound follows.

If we set $\lambda = \lambda_i = \epsilon 2^i / \log W$ and $\kappa = \frac{1}{\epsilon}$, then Lemma 3 asserts that $\mathcal{C}_{I,\lambda,\kappa} = \mathcal{C}_{I,\lambda_i,\frac{1}{\epsilon}}$ is an $O(\frac{f_*(I)}{\epsilon 2^i} \log W + \frac{1}{\epsilon})$-space data structure that enables us to obtain, for any item $a \in U$ and any timestamp $t \in I$, an estimate $\hat{f}_a([t,r_I])$ that satisfies

$$f_a([t,r_I]) - \epsilon f_*(I) \leq \hat{f}_a([t,r_I]) \leq f_a([t,r_I]) + \epsilon 2^i$$

If $f_*(I)$ does not vary too much, we can determine the i such that $f_*(I) \approx 2^i$, and $\mathcal{C}_{I,\lambda_i,\frac{1}{\epsilon}}$ is an $O(\frac{1}{\epsilon}\log W)$ space data structure that guarantees an error bound of $O(\epsilon f_*(I))$. However, this approach has two obvious shortcomings:

(1) $f_*(I)$ may vary from some small value to a value as large as B, the maximum number of items falling in a window of size W; hence, there may not be any fixed i that always satisfies $f_*(I) \approx 2^i$.

(2) To estimate $f_a([t,r_I])$, we need an error bound of $\epsilon f_*([t,r_I])$, not $\epsilon f_*(I)$.

We will explain how to overcome these two shortcomings in the next section.

4. Our Data Structure for ϵ-Approximate Counting

The first shortcoming of the approach given in Section 3 is easy to overcome: a natural idea is to maintain $\mathcal{C}_{I,\lambda_i,\frac{1}{\epsilon}}$ for different λ_i to handle different possible values of $f_*(I)$. The second shortcoming is more fundamental. To overcome it, we need to modify $\mathcal{C}_{I,\lambda,\kappa}$ substantially. The result is a new and complicated data structure $\mathcal{D}^Y_{I,\epsilon}$, where Y is an integer determining the accuracy. As asserted in Theorem 7 below, this data structure uses $O(\frac{1}{\epsilon}\log W \log\log W)$ space, supports $O(\log\frac{1}{\epsilon} + \log\log W)$ update time, and for any $t \in I$, it offers the following special guarantee:

- When $f_*([t, r_I]) \leq Y$, $\mathcal{D}^Y_{I,\epsilon}$ can return, for any item a, an estimate $\hat{f}_a([t, r_I])$ of $f_a([t, r_I])$ such that $|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq \epsilon Y$.
- When $f_*([t, r_I]) > Y$, $\mathcal{D}^Y_{I,\epsilon}$ does not have any error bound on its estimate $\hat{f}_a([t, r_I])$.

Before giving the details of $\mathcal{D}^Y_{I,\epsilon}$, let us explain how to use it to build the data structure $\mathcal{D}_{I,\epsilon}$ mentioned in Section 2 for the ϵ-approximate counting problem. To build $\mathcal{D}_{I,\epsilon}$, we need another $O(\frac{1}{\epsilon}\log W \log\frac{\epsilon B}{\log W})$-space data structure $\mathcal{B}_{I,\epsilon}$, which is a simple adaption of the data structure $\mathcal{B}_\epsilon$ of Cormode *et al.* [1] for the ϵ-approximate basic counting problem; $\mathcal{B}_{I,\epsilon}$ enables us to find, for any $t \in I$, an estimate $\hat{f}_*([t, r_I])$ of $f_*([t, r_I])$ such that

$$f_*([t, r_I]) \leq \hat{f}_*([t, r_I]) \leq (1+\epsilon) f_*([t, r_I]) \tag{7}$$

$\mathcal{B}_{I,\epsilon}$ is implemented as follows. During execution, we maintain the data structure $\mathcal{B}_{\epsilon/4}$ of Cormode *et al.* to count the items in the sliding window. When $\tau_{\texttt{cur}} = r_I$, we duplicate $\mathcal{B}_{\epsilon/4}$ and get $\mathcal{B}'$. Then, $\mathcal{B}'$ is updated as if $\tau_{\texttt{cur}}$ was fixed at r_I. To get the estimate $\hat{f}_*([t, r_I])$, we first obtain an estimate f' of $f_*([t, r_I])$ from $\mathcal{B}'$, which satisfies $|f' - f_*([t, r_I])| \leq \frac{\epsilon}{4} f_*([t, r_I])$. Then, $\hat{f}_*([t, r_I]) = \frac{1}{1-\epsilon/4} f'$. It can be verified that $\hat{f}_*([t, r_I])$ satisfies Equation (7). Our data structure $\mathcal{D}_{I,\epsilon}$ is composed of (i) $\mathcal{B}_{I,\epsilon}$, and (ii) $\mathcal{D}^{2^i}_{I,\epsilon/4}$ for each integer i from $\log(\frac{1}{\epsilon}\log W) + 1$ to $\log B$. It also maintains a brute-force $O(\frac{1}{\epsilon}\log W)$-space data structure for remembering the $\frac{1}{\epsilon}\log W$ items (a, u) with the largest $u \in I$; this brute-force data structure will be used for finding $\hat{f}_a([t, r_I])$ only when $f_*([t, r_I]) \leq \frac{1}{\epsilon}\log W$.

Theorem 4

(i) The data structure $\mathcal{D}_{I,\epsilon}$ has size $O\big(\frac{1}{\epsilon}(\log\log W)(\log W)\log(\frac{\epsilon B}{\log W})\big)$ words, and supports $O\big((\log\frac{1}{\epsilon} + \log\log W)\log(\frac{\epsilon B}{\log W})\big)$ update time.

(ii) Given $\mathcal{D}_{I,\epsilon}$, we can find, for any $a \in \Sigma$ and $t \in I$, an estimate of $\hat{f}_a([t, r_I])$ of $f_a([t, r_I])$ such that $|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq \epsilon f_*([t, r_I])$.

Proof Statement (i) is straightforward because there are $\log B - \log(\frac{1}{\epsilon}\log W)$ different $\mathcal{D}^Y_{I,\epsilon}$, each has size $O(\frac{1}{\epsilon}(\log\log W)\log W)$ and takes $O(\log\frac{1}{\epsilon} + \log\log W)$ time for an update. For Statement (ii), we describe how to get the estimate and analyze its accuracy.

First, we use $\mathcal{B}_{I,\epsilon}$ to get the estimate $\hat{f}_*([t, r_I])$. If $\hat{f}_*([t, r_I]) \leq \frac{1}{\epsilon}\log W$, then $f_*([t, r_I]) \leq \hat{f}_*([t, r_I]) \leq \frac{1}{\epsilon}\log W$ and we can use the brute-force data structure to find $f_a([t, r_I])$ exactly. Otherwise, we determine the i with $2^{i-1} < \hat{f}_*([t, r_I]) \leq 2^i$. Note that

- $i \geq \log(\frac{1}{\epsilon}\log W) + 1$ and we have the data structure $\mathcal{D}^{2^i}_{I,\frac{\epsilon}{4}}$, and

- $f_*([t, r_I]) \leq \hat{f}_*([t, r_I]) \leq 2^i$.

We use $\mathcal{D}^{2^i}_{I,\frac{\epsilon}{4}}$ to obtain an estimate $\hat{f}_a([t, r_I])$ with $|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq (\frac{\epsilon}{4})2^i$. By Equation (7), $2^{i-1} < \hat{f}_*([t, r_I]) \leq (1+\epsilon)f_*([t, r_I])$. Combining the two inequalities we have

$$|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq 2(\tfrac{\epsilon}{4})(2^{i-1}) < 2(\tfrac{\epsilon}{4})(1+\epsilon)f_*([t, r_I]) \leq \epsilon f_*([t, r_I])$$

We now describe the construction of $\mathcal{D}^Y_{I,\epsilon}$. First, we describe an $O(\frac{1}{\epsilon}(\log W)^2)$-space version of the data structure. Then, we show in the next section how to reduce the space to $O(\frac{1}{\epsilon}\log\log W \log W)$. In our discussion, we fix $\lambda = \epsilon Y/\log W$ and $\kappa = \frac{4}{\epsilon}\log W$.

Initially, $\mathcal{D}^Y_{I,\epsilon}$ is just the data structure $\mathcal{C}_{I,\lambda,\kappa}$. By Lemma 3, we know that its size is $O(\frac{f_*(I)}{\lambda} + \kappa) = O(\frac{f_*(I)}{\epsilon Y}\log W + \frac{1}{\epsilon}\log W)$, which is $O(\frac{1}{\epsilon}\log W)$ when $f_*(I) \leq Y$. However, it is much larger than $\frac{1}{\epsilon}\log W$ when $f_*(I) \gg Y$, and to maintain small space usage in such case, we trim $\mathcal{C}_{I,\lambda,\kappa}$ by throwing away a significant number of nodes. This is acceptable because $\mathcal{C}_{I,\lambda,\kappa}$ only guarantees good estimates for those $t \in I$ with $f_*([t, r_I]) \leq Y$. The trimming process is rather tricky. The natural idea of throwing away all the nodes to the left of t when we find $f_*([t, r_I]) > Y$ does not work because the resulting data structure may return estimates with error larger than the required ϵY bound. For example, let $I = [1, W]$. For each item $a_i \in \{a_1, a_2, \ldots, a_{\kappa-1}\}$, there are $m = Y/\kappa$ copies of $(a_i, t+1)$ arrive at time $W + t$ for every $t \in [0, W-1]$. Also, there are m copies of (a, W) arrive at time $W + t$ for every $t \in [0, W-1]$. Hence, at each time $W + t$, there are $m\kappa = Y$ items with timestamps in $[t, W]$ arrives, m items for each of the κ item name in $\{a, a_1, \ldots, a_{\kappa-1}\}$. We are interested in the accuracy of the estimate $\hat{f}_a([W, W])$. It can be verified that at each time $W + t$, Lines 4–5 of the procedure Process( ) will eventually trivialize $Q^a_{I,\lambda}$ and thus $\hat{f}_a([W, W]) = 0$. Since $f_a([W, W]) = (t+1)m$, $|\hat{f}_a([W, W]) - f_a([W, W])| = (t+1)m$. When $t = 2\epsilon Y/m - 1$, the absolute error is $2\epsilon Y$ which is larger than the required error bound ϵY.

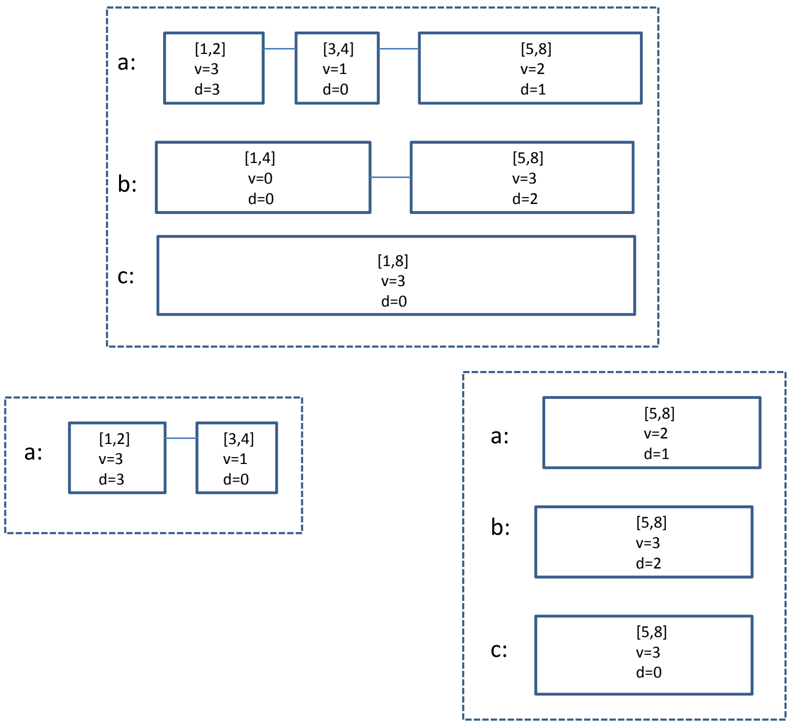
To describe the right trimming procedure, we need some basic operations. Consider any $\mathcal{C}_{J,\lambda,\kappa}$ where $J = [p, q]$. The following operation splits $\mathcal{C}_{J,\lambda,\kappa}$ into two smaller data structures $\mathcal{C}_{J_\ell,\lambda,\kappa}$ and $\mathcal{C}_{J_r,\lambda,\kappa}$ where $J_\ell = [p, m]$ and $J_r = [m+1, q]$ with $m = \lfloor (p+q)/2 \rfloor$.

```
$\mathcal{D}^Y_{I,\epsilon}$.Split($\mathcal{C}_{J,\lambda,\kappa}$)
1:  for each non-trivial queue $Q^a_{J,\lambda} \in \mathcal{C}_{J,\lambda,\kappa}$ do
2:    if ($Q^a_{J,\lambda}$ has only one node $N$ monitoring the whole interval $J$) then
3:      /* refine $J$ */
4:      insert a new node $N'$ immediately to the left of $N$ with $v(N') = d(N') = 0$;
5:      $i(N') = J_\ell$, and $i(N) = J_r$;
6:    end if
7:    divide $Q^a_{J,\lambda}$ into two sub-queues $Q^a_{J_\ell,\lambda}$ and $Q^a_{J_r,\lambda}$ where
8:      $Q^a_{J_\ell,\lambda}$ contains the nodes monitoring some sub-intervals of $J_\ell$, and
9:      $Q^a_{J_r,\lambda}$ contains those monitoring some sub-intervals of $J_r$;
10:   put $Q^a_{J_\ell,\lambda}$ in $\mathcal{C}_{J_\ell,\lambda,\kappa}$ and $Q^a_{J_r,\lambda}$ in $\mathcal{C}_{J_r,\lambda,\kappa}$.
11: end for
12: /* For a trivial $Q^a_{J,\lambda}$, its two children in $\mathcal{C}_{J_\ell,\lambda,\kappa}$ and $\mathcal{C}_{J_r,\lambda,\kappa}$ are also trivial. */
```

We say that $\mathcal{C}_{J_\ell,\lambda,\kappa}$ and $\mathcal{C}_{J_r,\lambda,\kappa}$ are the left and right child of $\mathcal{C}_{J,\lambda,\kappa}$, respectively. Figure 3 gives an example of Split($\mathcal{C}_{[1,8],\lambda,\kappa}$), the split of $\mathcal{C}_{[1,8],\lambda,\kappa}$, which has three non-trivial queues $Q^a_{I,\lambda}$, $Q^b_{I,\lambda}$ and

$Q^c_{I,\lambda}$, into $\mathcal{C}_{[1,4],\lambda,\kappa}$ and $\mathcal{C}_{[5,8],\lambda,\kappa}$. Note that the queues for b and c in $\mathcal{C}_{[1,4],\lambda,\kappa}$ are trivial and we have not stored them.

Figure 3. Split of $\mathcal{C}_{[1,8],\lambda,\kappa}$.



Using Split( ), we can trim, for example, $\mathcal{C}_{[p,p+1],\lambda,\kappa}$ into $\mathcal{C}_{[p+1,p+1],\lambda,\kappa}$ as follows: Split $\mathcal{C}_{[p,p+1],\lambda,\kappa}$ into $\mathcal{C}_{[p,p],\lambda,\kappa}$ and $\mathcal{C}_{[p+1,p+1],\lambda,\kappa}$, and throw away $\mathcal{C}_{[p,p],\lambda,\kappa}$. The following recursive procedure LeftRefine( ) generalizes this idea for larger J: Given $\mathcal{C}_{J,\lambda,\kappa} = \mathcal{C}_{[p,q],\lambda,\kappa}$, it returns a list $\langle \mathcal{C}_{J_0,\lambda,\kappa}, \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa}\rangle$ where the J_i's form a partition of $[p,q]$, and $J_0 = [p,p]$. Throwing away $\mathcal{C}_{J_0,\lambda,\kappa}$, and the remaining $\mathcal{C}_{J_i,\lambda,\kappa}$'s all together monitor $[p+1,q]$.

```
$\mathcal{D}^Y_{I,\epsilon}$.LeftRefine ($\mathcal{C}_{[p,q],\lambda,\kappa}$)
1: **if** ($|[p,q]| = |[p,p]| = 1$) **then**
2:     return $\langle \mathcal{C}_{[p,p],\lambda,\kappa} \rangle$;
3: **else**
4:     split $\mathcal{C}_{[p,q],\lambda,\kappa}$ into its left child $\mathcal{C}_{[p,m],\lambda,\kappa}$ and right child $\mathcal{C}_{[m+1,q],\lambda,\kappa}$
5:     /* where $m = \lfloor (p+q)/2 \rfloor$ */;
6:     $L$ = LeftRefine($\mathcal{C}_{[p,m],\lambda,\kappa}$);
7:     suppose $L = \langle \mathcal{C}_{J_0,\lambda,\kappa}, \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_k,\lambda,\kappa} \rangle$;
8:     return $\langle \mathcal{C}_{J_0,\lambda,\kappa}, \ldots, \mathcal{C}_{J_k,\lambda,\kappa}, \mathcal{C}_{[m+1,q],\lambda,\kappa} \rangle$;
9: **end if**
```

For example, LeftRefine($\mathcal{C}_{[1,8],\lambda,\kappa}$) gives us the list $\langle \mathcal{C}_{[1,1],\lambda,\kappa}, \mathcal{C}_{[2,2],\lambda,\kappa}, \mathcal{C}_{[3,4],\lambda,\kappa}, \mathcal{C}_{[5,8],\lambda,\kappa}\rangle$. Note that $J_0 = [p,p]$ because the recursion stops only when $|[p,q]| = 1$. The list returned by LeftRefine($\mathcal{C}_{[p,q],\lambda,\kappa}$) has another useful property, which we describe below.

Given $L = \langle \mathcal{C}_{Z_1,\lambda,\kappa}, \ldots, \mathcal{C}_{Z_k,\lambda,\kappa} \rangle$, we say that L is an *interesting-partition* covering the interval J if (i) the Z_i's are all interesting intervals and form a partition of J; and (ii) for $1 \le i < k$, Z_i is to the left of Z_{i+1}, and $|Z_i| \le \frac{1}{2}|Z_{i+1}|$. The fact below can be verified by induction on the length of the list returned by LeftRefine( ).

Fact 3 *Let J be an interesting interval, and $L = \langle \mathcal{C}_{J_0,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ be the list returned by LeftRefine($\mathcal{C}_{J,\lambda,\kappa}$). Then, the list $\langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ (i.e., the list obtained by throwing away the head $\mathcal{C}_{J_0,\lambda,\kappa}$ of L) is an interesting-partition covering $[p+1, q]$.*

For example, if $[1, 8]$ is an interesting interval, then the list $\langle \mathcal{C}_{[2,2],\lambda,\kappa} \mathcal{C}_{[3,4],\lambda,\kappa}, \mathcal{C}_{[5,8],\lambda,\kappa} \rangle$ obtained by throwing away the first element $\mathcal{C}_{[1,1],\lambda,\kappa}$ from LeftRefine($\mathcal{C}_{[1,8],\lambda,\kappa}$) is an interesting-partition covering $[2, 8]$.

We now give details of $\mathcal{D}^Y_{I,\epsilon}$. Initially, it is the interesting-partition $\langle \mathcal{C}_{I,\lambda,\kappa} \rangle$ covering the whole interval $I = [\ell_I, r_I]$. Throughout the execution, we maintain the following invariant:

$(**)$ $\mathcal{D}^Y_{I,\epsilon}$ is an interesting-partition covering some $[p, r_I] \subseteq I$.

When $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ is covering $[p, r_I]$, it only guarantees good estimates of $f_a([t, r_I])$ for $t \in [p, r_I]$, and this estimate is obtained by

$$\hat{f}_a([t, r_I]) = v(\mathsf{alive}(\mathcal{C}_{J_h,\lambda,\kappa})^a_{\ge t}) + \textstyle\sum_{h+1 \le i \le m} v(\mathsf{alive}(\mathcal{C}_{J_i,\lambda,\kappa})^a)$$

(or equivalently, $\hat{f}_a([t, r_I]) = v(\mathsf{alive}(Q^a_{J_h,\lambda})_{\ge t}) + \sum_{h+1 \le i \le m} v(\mathsf{alive}(Q^a_{J_i,\lambda}))$, where J_h is the interval in $\{J_1, J_2, \ldots, J_m\}$ that covers t. When an item (a, u) with $u \in [p, r_I]$ arrives, we find the unique $\mathcal{C}_{J_i,\lambda,\kappa}$ in $\mathcal{D}^Y_{I,\epsilon}$ where $u \in J_i$, update it by calling $\mathcal{C}_{J_i,\lambda,\kappa}$.Process($(a, u)$). Note that this update has no effect on the other $\mathcal{C}_{J,\lambda,\kappa}$ in $\mathcal{D}^Y_{I,\epsilon}$.

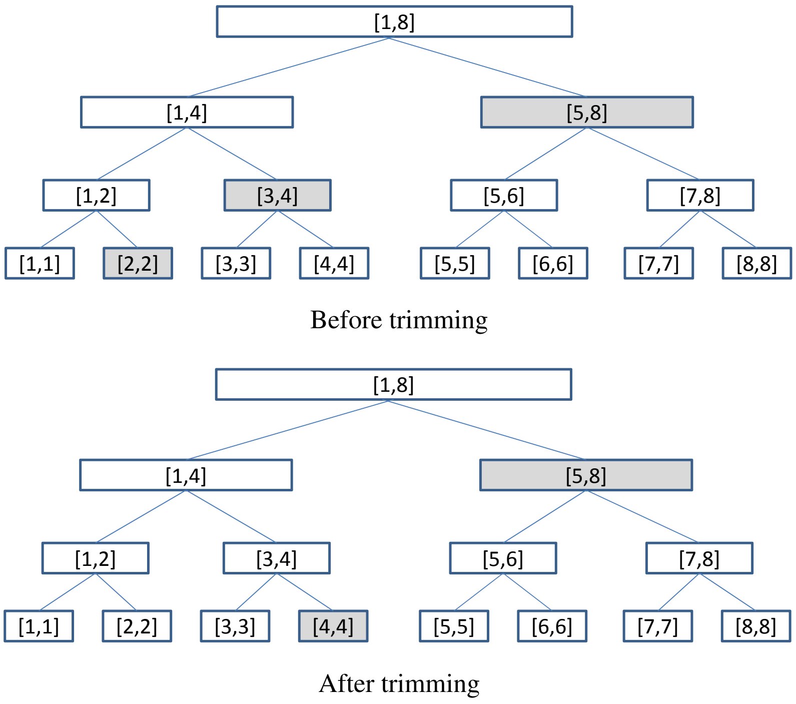
During execution, we also keep track of the largest timestamp $p_{\mathtt{max}} \in I$ such that the estimate $\hat{f}_*([p_{\mathtt{max}}, r_I])$ given by $\mathcal{B}_{I,\epsilon}$ is greater than $(1+\epsilon)Y$ (which implies $f_*([p_{\mathtt{max}}, r_I]) > Y$ because of Equation (7)). As soon as $p_{\mathtt{max}}$ falls in the interval covered by $\mathcal{D}^Y_{I,\epsilon}$, we use the following procedure to trim $\mathcal{D}^Y_{I,\epsilon}$ to cover the smaller interval $[p_{\mathtt{max}} + 1, r_I]$.

Suppose that $L = \langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ is an interesting-partition covering $[p, r_I]$, and $t \in [p, r_I]$. Trim(L, t) constructs an interesting-partition covering $[t+1, r_I]$ recursively as follows.

```
$\mathcal{D}^Y_{I,\epsilon}$.Trim($L$, $t$)
1:  find the unique $\mathcal{C}_{J_i,\lambda,\kappa}$ in $L$ such that $t \in J_i$;
2:  $L'$ =LeftRefine($\mathcal{C}_{J_i,\lambda,\kappa}$);
3:  suppose $L' = \langle \mathcal{C}_{K_0,\lambda,\kappa}, \mathcal{C}_{K_1,\lambda,\kappa}, \ldots, \mathcal{C}_{K_\ell,\lambda,\kappa} \rangle$;
4:  **if** ($K_0 = [t, t]$) **then**
5:      return $\langle \mathcal{C}_{K_1,\lambda,\kappa}, \ldots, \mathcal{C}_{K_\ell,\lambda,\kappa}, \mathcal{C}_{J_{i+1},\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$;
6:      /* *i.e.*, throw away $\mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_{i-1},\lambda,\kappa}$ and $\mathcal{C}_{K_0,\lambda,\kappa}$, */
7:      /* and return an interesting-partition covering $[t+1, r_I]$. */
8:  **else**
9:      return Trim($\langle \mathcal{C}_{K_1,\lambda,\kappa}, \ldots, \mathcal{C}_{K_\ell,\lambda,\kappa}, \mathcal{C}_{J_{i+1},\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle, t$).
10:     /* throw away $\mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_{i-1},\lambda,\kappa}$ and $\mathcal{C}_{K_0,\lambda,\kappa}$. */
11: **end if**
```

For example, Figure 4 shows that when $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{[2,2],\lambda,\kappa}, \mathcal{C}_{[3,4],\lambda,\kappa}, \mathcal{C}_{[5,8],\lambda,\kappa} \rangle$, Trim$(\mathcal{D}^Y_{I,\epsilon}, 3)$ return $\langle \mathcal{C}_{[4,4],\lambda,\kappa}, \mathcal{C}_{[5,8],\lambda,\kappa} \rangle$. Based on Fact 3, it can be verified inductively that after $\mathcal{D}^Y_{I,\epsilon} \leftarrow$ Trim$(\mathcal{D}^Y_{I,\epsilon}, p_{\texttt{max}})$, the new $\mathcal{D}^Y_{I,\epsilon}$ is an interesting-partition covering $[p_{\texttt{max}} + 1, r_I]$; Invariant $(**)$ is preserved. In the rest of this section, we analyze the size of $\mathcal{D}^Y_{I,\epsilon}$ and the accuracy of its estimates.

Figure 4. Trim$(\langle \mathcal{C}_{[2,2],\lambda,\kappa}, \mathcal{C}_{[3,4],\lambda,\kappa}, \mathcal{C}_{[5,8],\lambda,\kappa} \rangle, 3)$.



Let ALL be the set of all $\mathcal{C}_{J,\lambda,\kappa}$'s that ever exist, *i.e.*, if $\mathcal{C}_{J,\lambda,\kappa} \in$ ALL, then either (i) it is currently in $\mathcal{D}^Y_{I,\epsilon}$; or (ii) it has been in $\mathcal{D}^Y_{I,\epsilon}$ some time earlier in the execution, but is thrown away during some trimming of $\mathcal{D}^Y_{I,\epsilon}$. For any $p \in I$, define

$$\text{ALL}_{\geq p} = \left\{ \mathcal{C}_{J,\lambda,\kappa} \mid \mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}, \text{ and } J \text{ covers or is to the right of } p \right\}$$

Let $v_{\texttt{add}}(\mathcal{C}_{J,\lambda,\kappa})$ be the total value added to the nodes of $\mathcal{C}_{J,\lambda,\kappa}$ during its lifespan. We now derive an upper bound on $\sum_{\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}} v_{\texttt{add}}(\mathcal{C}_{J,\lambda,\kappa})$, which is crucial for getting a tight error bound on the accuracy of $\mathcal{D}^Y_{I,\epsilon}$'s estimates.

Recall that initially $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{I,\lambda,\kappa} \rangle$ and thus $\mathcal{C}_{I,\lambda,\kappa} \in$ ALL. For any other $\mathcal{C}_{J,\lambda,\kappa} \in$ ALL, $\mathcal{C}_{J,\lambda,\kappa}$ must be a child of some $\mathcal{C}_{H,\lambda,\kappa} \in$ ALL (*i.e.*, $\mathcal{C}_{J,\lambda,\kappa}$ is obtained from Split$(\mathcal{C}_{H,\lambda,\kappa})$). Given $\mathcal{C}_{J,\lambda,\kappa}$ and $\mathcal{C}_{H,\lambda,\kappa}$, we say that $\mathcal{C}_{J,\lambda,\kappa}$ is a descendant of $\mathcal{C}_{H,\lambda,\kappa}$, and $\mathcal{C}_{H,\lambda,\kappa}$ is an ancestor of $\mathcal{C}_{J,\lambda,\kappa}$, if either (i) $\mathcal{C}_{J,\lambda,\kappa}$ is a child of $\mathcal{C}_{H,\lambda,\kappa}$, or (ii) it is a child of some of $\mathcal{C}_{H,\lambda,\kappa}$'s descendants. Note that the original $\mathcal{C}_{I,\lambda,\kappa}$ is an ancestor of every $\mathcal{C}_{J,\lambda,\kappa} \in$ ALL, and in general, any $\mathcal{C}_{H,\lambda,\kappa} \in$ ALL is an ancestor of every $\mathcal{C}_{J,\lambda,\kappa} \in$ ALL with $J \subset H$. We have the following lemma. (Note that we are abusing the notation here and regard $\mathcal{D}^Y_{I,\epsilon}$ as a set.)

Lemma 5 *Suppose that* $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ *is covering* $[p, r_I]$. *Let* $anc(\mathcal{D}^Y_{I,\epsilon}) = anc(\langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle)$ *be the set* $\left\{ \mathcal{C}_{H,\lambda,\kappa} \mid \mathcal{C}_{H,\lambda,\kappa} \text{ is an ancestor of some } \mathcal{C}_{J_i,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon} \right\}$. *Then,*

(1) $\text{ALL}_{\geq p} \subseteq \mathcal{D}^Y_{I,\epsilon} \cup anc(\mathcal{D}^Y_{I,\epsilon})$,

(2) $v_{\mathtt{add}}(\mathcal{C}_{J,\lambda,\kappa}) \leq (1+\epsilon)Y$ *for any* $\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}$*, and*
(3) $|\mathcal{D}^Y_{I,\epsilon} \cup \mathit{anc}(\mathcal{D}^Y_{I,\epsilon})| \leq 2\log W$.

Therefore, $\sum\{v_{\mathtt{add}}(\mathcal{C}_{J,\lambda,\kappa}) \mid \mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}\} \leq 2(1+\epsilon)Y\log W$.

Proof For (1), it suffices to prove that for any $\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}$, $\mathcal{C}_{J,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{D}^Y_{I,\epsilon})$. By definition, J covers or is to the right of p; thus $J \cap (J_1 \cup \cdots \cup J_m) = J \cap [p, r_I] \neq \emptyset$. Since the intervals are interesting and do not cross, there is an $1 \leq i \leq m$ such that either (i) $J = J_i$, and thus $\mathcal{C}_{J,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon}$, or (ii) $J_i \subset J$, which implies $\mathcal{C}_{J,\lambda,\kappa}$ is an ancestor of $\mathcal{C}_{J_i,\lambda,\kappa}$, *i.e.*, $\mathcal{C}_{J,\lambda,\kappa} \in \text{anc}(\mathcal{D}^Y_{I,\epsilon})$. (It is not possible that $J \subset J_i$; otherwise $\mathcal{C}_{J_i,\lambda,\kappa}$ would have been split and should not be in the current $\mathcal{D}^Y_{I,\epsilon}$. Hence, $\mathcal{C}_{J,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{D}^Y_{I,\epsilon})$.

To prove (2), suppose that $J = [x,y]$ and $v_{\mathtt{add}}(\mathcal{C}_{J,\lambda,\kappa})$ has just reached $(1+\epsilon)Y$. This implies $f_*([x,r_I]) \geq (1+\epsilon)Y$, and so does its estimate $\hat{f}_*([x,r_I])$ given by $\mathcal{B}_{I,\epsilon}$ (as $f_*([x,r_I]) \leq \hat{f}_*([x,r_I])$, by Equation (7)). Then, the procedure Trim( ) will be called and $\mathcal{C}_{J,\lambda,\kappa}$ will be either thrown away or split, and no more value can be added to $\mathcal{C}_{J,\lambda,\kappa}$. It follows that $v_{\mathtt{add}}(\mathcal{C}_{J,\lambda,\kappa}) \leq (1+\epsilon)Y$.

For (3), recall that $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa}, \mathcal{C}_{J_2,\lambda,\kappa}, \ldots \mathcal{C}_{J_m,\lambda,\kappa}\rangle$. Among the intervals $J_1, \ldots, J_m$, interval J_1 is the leftmost interval and its left boundary $\ell_{J_1} = p$. We now prove that $\mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{D}^Y_{I,\epsilon}) = \mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})$ where $\text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})$ is the set of ancestors of $\mathcal{C}_{J_1,\lambda,\kappa}$. Then, together with the facts that $|\mathcal{D}^Y_{I,\epsilon}| \leq \log W$ (by Property (ii) of interesting-partition) and $|\text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})| \leq \log W$ (as each Split operation would reduce the size of interval by half), we have

$$|\mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{D}^Y_{I,\epsilon})| = |\mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})| \leq |\mathcal{D}^Y_{I,\epsilon}| + |\text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})| \leq 2\log W$$

To show $\mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{D}^Y_{I,\epsilon}) = \mathcal{D}^Y_{I,\epsilon} \cup \text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})$, it suffices to show that for any $\mathcal{C}_{H,\lambda,\kappa} \in \text{anc}(\mathcal{D}^Y_{I,\epsilon})$, $\mathcal{C}_{H,\lambda,\kappa} \in \text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})$. Since $\mathcal{C}_{H,\lambda,\kappa} \in \text{anc}(\mathcal{D}^Y_{I,\epsilon})$, it is the ancestor of some $\mathcal{C}_{J_i,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon}$. Thus $J_i = [\ell_{J_i}, r_{J_i}] \subset H = [\ell_H, r_H]$. Since $\mathcal{C}_{H,\lambda,\kappa}$ is already an ancestor, it no longer exists, and all the $\mathcal{C}_{J,\lambda,\kappa}$ to its left have been thrown away. Thus, $\mathcal{D}^Y_{I,\epsilon}$ has no $\mathcal{C}_{J,\lambda,\kappa}$ where J is to the right of ℓ_H. This implies $\ell_H \leq p = \ell_{J_1}$ and $\ell_H \leq \ell_{J_1} \leq r_{J_1} \leq r_{J_i} \leq r_H$. It follows that $J_1 \subset H$ and $\mathcal{C}_{H,\lambda,\kappa}$ is an ancestor of $\mathcal{C}_{J_1,\lambda,\kappa}$, *i.e.*, $\mathcal{C}_{H,\lambda,\kappa} \in \text{anc}(\mathcal{C}_{J_1,\lambda,\kappa})$.

We are now ready to analyze the accuracy of $\mathcal{D}^Y_{I,\epsilon}$'s estimates.

Theorem 6 *Suppose that* $\mathcal{D}^Y_{I,\epsilon}$ *is covering* $[p, r_I]$*. For any item* a *and any* $t \in [p, r_I]$*, the estimate* $\hat{f}_a([t,r_I])$ *of* $f_a([t,r_I])$ *obtained by* $\mathcal{D}^Y_{I,\epsilon}$ *satisfies* $|\hat{f}_a([t,r_I]) - f_a([t,r_I])| \leq \epsilon Y$*. Furthermore,* $\mathcal{D}^Y_{I,\epsilon}$ *uses* $O(\frac{1}{\epsilon}(\log W)^2)$ *space.*

Proof Let $\mathsf{alive}(\mathcal{D}^Y_{I,\epsilon})$ be the set of nodes currently in $\mathcal{D}^Y_{I,\epsilon}$, $\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})$ the set of those that were in $\mathcal{D}^Y_{I,\epsilon}$ earlier in the execution but have been deleted, and $\mathsf{node}(\mathcal{D}^Y_{I,\epsilon}) = \mathsf{alive}(\mathcal{D}^Y_{I,\epsilon}) \cup \mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})$. It can be verified that $\hat{f}_a([t,r_I]) = v(\mathsf{alive}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t})$. Below, we prove that

$$f_a([t,r_I]) - \tfrac{2(1+\epsilon)Y}{\kappa}\log W \leq v(\mathsf{alive}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}) \leq f_a([t,r_I]) + \lambda\log W \tag{8}$$

Recall that we fix $\lambda = \epsilon Y/\log W$ and $\kappa = \frac{4}{\epsilon}\log W$; the ϵY error bound follows.

The proof of the second inequality of Equation (8) is identical to that of Lemma 3, except that we replace all occurrences of $\mathcal{C}_{I,\lambda,\kappa}$ by $\mathcal{D}^Y_{I,\epsilon}$. The proof of the first inequality is also similar. We still have

$$f_a([t,r_I]) - v(\mathsf{alive}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}) \leq v(\mathsf{node}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}) - v(\mathsf{alive}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}) = v(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t})$$

which equals $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t})$. As in Lemma 3, we can derive the bound $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}) \leq \frac{1}{\kappa} v(\mathsf{node}(\mathcal{D}^Y_{I,\epsilon})) = \frac{1}{\kappa} f_*(I)$, but we can do better here.

Observe that for any node $N \in \mathsf{dead}((\mathcal{D}^Y_{I,\epsilon})^a_{\geq t})$, N can only be in those $\mathcal{C}_{J,\lambda,\kappa} \in \mathrm{ALL}_{\geq p}$ (because $t \in [p, r_I]$), and when we debit N, if it is in $\mathcal{C}_{J,\lambda,\kappa}$, then we debit $\kappa - 1$ other nodes in $\mathcal{C}_{J,\lambda,\kappa}$ monitoring $\kappa - 1$ items other than a. Thus, $\kappa \cdot d(\mathsf{dead}((\mathcal{D}^Y_{I,\epsilon})^a_{\geq t}))$ is no more than the total value available in the $\mathcal{C}_{J,\lambda,\kappa} \in \mathrm{ALL}_{\geq p}$, which is $\sum\{v_{\mathsf{add}}(\mathcal{C}_{J,\lambda,\kappa}) \mid \mathcal{C}_{J,\lambda,\kappa} \in \mathrm{ALL}_{\geq p}\}$. Together with Lemma 5 we conclude

$$\kappa \cdot d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq p}) \leq \sum\{v_{\mathsf{add}}(\mathcal{C}_{J,\lambda,\kappa}) \mid \mathcal{C}_{J,\lambda,\kappa} \in \mathrm{ALL}_{\geq p}\} \leq 2(1+\epsilon) Y \log W$$

and the first inequality of Equation (8) follows.

For the size of $\mathcal{D}^Y_{I,\epsilon}$, similar to the proof of Lemma 3, we can argue that the number of born-rich nodes is only $O(Y/\lambda) = O(\frac{1}{\epsilon}\log W)$, but the number of born-poor nodes can be much larger. A born-poor node of a non-trivial queue is created either when we increase the value of a trivial queue, or when we execute Lines 2–6 of procedure Split. It can be verified that every queue $Q^a_{I,\lambda}$ has at most one born-poor node, which is the rightmost node in $Q^a_{I,\lambda}$. Since there are $O(\log W)$ $\mathcal{C}_{J,\lambda,\kappa}$'s in $\mathcal{D}^Y_{I,\epsilon}$ and each has at most κ non-trivial queues, the number of born-poor nodes, and hence the size of $\mathcal{D}^Y_{I,\epsilon}$, is $O(\kappa \log W) = O(\frac{1}{\epsilon}(\log W)^2)$.

To reduce $\mathcal{D}^Y_{I,\epsilon}$'s size from $O(\frac{1}{\epsilon}(\log W)^2)$ to $O(\frac{1}{\epsilon}\log\log W \log W)$, we need to reduce the number of born-poor nodes; or equivalently, the number of non-trivial queues in $\mathcal{D}^Y_{I,\epsilon}$. In the next section, we give a simple idea to reduce the number of non-trivial queues and hence the size of $\mathcal{D}^Y_{I,\epsilon}$ to $O(\frac{1}{\epsilon}\log\log W \log W)$. In Section 6, we show how to further reduce the size by taking advantage of the tardiness of the data stream.

5. Reducing the Size of $\mathcal{D}^Y_{I,\epsilon}$

Our idea for reducing the size is simple; for every $\mathcal{C}_{J,\lambda,\kappa} \in \mathcal{D}^Y_{I,\epsilon}$, its capacity is no longer fixed at $\kappa = \frac{4}{\epsilon}\log W$; instead, we start with a much smaller capacity, namely $\frac{4}{\epsilon}\log\log W$, which is allowed to increase gradually during execution. To determine $\mathcal{C}_{J,\lambda,\kappa}$'s capacity, we use a variable to keep track of the number $\bar{f}_*(J)$ of items (a,u) with $u \in J$ that have arrived since $\mathcal{C}_{J,\lambda,\kappa}$'s creation. Let v_J be the total value of the nodes in $\mathcal{C}_{J,\lambda,\kappa}$ when it is created (v_J may not be zero if $\mathcal{C}_{J,\lambda,\kappa}$ is resulted from the splitting of its parent). The capacity of $\mathcal{C}_{J,\lambda,\kappa}$ is determined as follows.

When $\frac{(c-1)Y}{\log W} \leq v_J + \bar{f}_*(J) < \frac{cY}{\log W}$ for some integer $c \geq 1$, the capacity of $\mathcal{C}_{J,\lambda,\kappa}$ is $\kappa(c) = \frac{4c}{\epsilon}\log\log W$, *i.e.*, set $\kappa = \kappa(c)$ and allow $\kappa(c)$ non-trivial queues in $\mathcal{C}_{J,\lambda,\kappa}$.

Note that when we increase the capacity of $\mathcal{C}_{J,\lambda,\kappa}$ to $\kappa(c)$, we do not need to do anything, except that we allow more non-trivial queues (up to $\kappa(c)$) in the data structure. Also note that when $\mathcal{C}_{J,\lambda,\kappa}$ is created during the trimming process, its inherited capacity may be larger than the supposed capacity $\kappa(c)$; in such case, we simply debit every non-trivial queue until some queue $Q^x_{J,\lambda}$ has $v(Q^x_{J,\lambda}) = d(Q^x_{J,\lambda})$ and we execute Lines 4 and 5 of the procedure Process( ) to make this queue trivial. We repeat the process until the number of non-trivial queues is at most $\kappa(c)$. The following theorem asserts that $\mathcal{D}^Y_{I,\epsilon}$ maintains the accuracy of its estimates under this new implementation. It gives the revised size and the update time.

Theorem 7

(1) Suppose that $\mathcal{D}^Y_{I,\epsilon}$ is currently covering $[p, r_I]$. For any item $a \in \Sigma$ and any timestamp $t \in [p, r_I]$, the estimate $\hat{f}_a([t, r_I])$ of $f_a([t, r_I])$ obtained by the new $\mathcal{D}^Y_{I,\epsilon}$ satisfies $|\hat{f}_a([t, r_I]) - f_a([t, r_I])| \leq \epsilon Y$.

(2) $\mathcal{D}^Y_{I,\epsilon}$ has size $O(\frac{1}{\epsilon}(\log\log W)\log W)$, and supports $O(\log\frac{1}{\epsilon} + \log\log W)$ update time.

Proof Suppose that $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa(c_1)}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)} \rangle$. From the fact that we are using $\mathcal{C}_{J_i,\lambda,\kappa(c_i)}$ to monitor J_i we conclude $\frac{(c_i-1)Y}{\log W} \leq v_{J_i} + \bar{f}_*(J_i)$. It follows that $\sum_{1\leq i\leq m}\frac{c_iY}{\log W} \leq \sum_{1\leq i\leq m}(v_{J_i} + \bar{f}_*(J_i)) + \sum_{1\leq i\leq m}\frac{Y}{\log W}$, which is O(Y) because (i) $|\mathcal{D}^Y_{I,\epsilon}| = m = O(\log W)$ and (ii) $\sum_{1\leq i\leq m}(v_{J_i} + \bar{f}_*(J_i)) = O(Y)$ (otherwise $\mathcal{D}^Y_{I,\epsilon}$ would have been trimmed). Thus,

$$\textstyle\sum_{1\leq i\leq m} c_i = O(\log W) \tag{9}$$

For Statement (1), the analysis of the accuracy of $\hat{f}_a([t, r_I])$ is very similar to that of Theorem 6, except for the following difference: In the proof of Theorem 6, we show that $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq p}) \leq \frac{2(1+\epsilon)Y}{\kappa}\log W$, and since κ is fixed at $\frac{4}{\epsilon}\log W$, $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq p}) \leq \epsilon Y$. Here, we also prove that $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq p}) \leq \epsilon Y$, but we have to prove it differently because the capacities are no longer fixed.

As argued previously, any node in $\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})_{\geq p}$ is in some $\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}$. Below, we show that for any $\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}$, we can make at most $\frac{\epsilon Y}{2\log W}$ debit operations to the queue $Q^a_{J,\lambda}$ of $\mathcal{C}_{J,\lambda,\kappa}$ during its lifespan. Together with the fact that $|\text{ALL}_{\geq p}| \leq 2\log W$, we have $d(\mathsf{dead}(\mathcal{D}^Y_{I,\epsilon})^a_{\geq p}) \leq \epsilon Y$.

Consider any $\mathcal{C}_{J,\lambda,\kappa} \in \text{ALL}_{\geq p}$. Note that the smaller its capacity, the larger the number of debit operations can be made to the queue $Q^a_{J,\lambda}$ of $\mathcal{C}_{J,\lambda,\kappa}$. To maximize the number of debit operations made to $Q^a_{J,\lambda}$, suppose that $v_J = 0$ and thus $\mathcal{C}_{J,\lambda,\kappa}$ has the smallest capacity $\kappa(1)$ when it is created. Before increasing its capacity to $\kappa(2)$, $\mathcal{C}_{J,\lambda,\kappa}$ can make at most $\frac{1}{\kappa(1)} \cdot \frac{Y}{\log W}$ debit operations to $Q^a_{J,\lambda}$. Then, during the next $\frac{Y}{\log W}$ arrivals of items (a, u) with $u \in J$, $\frac{Y}{\log W} \leq v_J + \bar{f}_*(J) < \frac{2Y}{\log W}$, the capacity is $\kappa(2)$, and at most $\frac{1}{\kappa(2)} \cdot \frac{Y}{\log W}$ debit operations can be made to $Q^a_{J,\lambda}$. In general, during the period when $\frac{(c-1)Y}{\log W} \leq v_J + \bar{f}_*(J) < \frac{cY}{\log W}$, at most $\frac{1}{\kappa(c)} \cdot \frac{Y}{\log W}$ debit operations can be made to $Q^a_{J,\lambda}$. If the largest capacity is $\kappa(c_{\max})$, the total number of debit operations made to $Q^a_{J,\lambda}$ is at most

$$\frac{Y}{\log W}\left(\frac{1}{\kappa(1)} + \cdots + \frac{1}{\kappa(c_{\max})}\right) = \frac{\epsilon Y}{4(\log\log W)\log W}\left(1 + \frac{1}{2} + \cdots + \frac{1}{c_{\max}}\right) \leq \frac{\epsilon Y(\ln(c_{\max})+1)}{4(\log\log W)\log W}$$

which is smaller than $\frac{\epsilon Y}{2\log W}$ because by Equation (9), $c_{\max} = O(\log W)$, which implies $\ln(c_{\max}) + 1 \leq 2\log\log W$ (suppose that W is larger than some constant).

We now prove (2). Note that the total number of non-trivial queues in $\mathcal{D}^Y_{I,\epsilon}$, and hence the number of born-poor nodes, is at most $\sum_{1\leq i\leq m}\kappa(c_i) = \sum_{1\leq i\leq m}\frac{4c_i}{\epsilon}\log\log W$. By Equation (9), $\sum_{1\leq i\leq m} c_i = O(\log W)$, and it follows that the size of $\mathcal{D}^Y_{I,\epsilon}$ is $O(\frac{1}{\epsilon}\log\log W\log W)$.

For the update time, suppose that an item (a, u) arrives. We can find the $\mathcal{C}_{J_i,\lambda,\kappa}$ in $\mathcal{D}^Y_{I,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa} \rangle$ with $u \in J_i$ using $O(\log m) = O(\log\log W)$ time by querying a balanced search tree storing the J_i's. By hashing (e.g., Cuckoo hashing [15], which supports constant update and query time) we can locate the queue $Q^a_{J_i,\lambda} \in \mathcal{C}_{J_i,\lambda,\kappa}$ in constant time. Then, by consulting an auxiliary balanced search tree on the intervals monitored by the nodes of $Q^a_{J_i,\lambda}$, we can find and update the node N of $Q^a_{J_i,\lambda}$ with $u \in i(N)$ using $O(\log(Y/\lambda)) = O(\log\frac{1}{\epsilon} + \log\log W)$ time. At times we may also need to execute Lines 3 and 4 of the procedure Process( ), which debits all the non-trivial queues in $\mathcal{C}_{J_i,\lambda,\kappa}$. Using the de-amortizing technique given in [16], this step takes constant time.

Note that occasionally, we may also need to clean up $\mathcal{D}^Y_{I,\epsilon}$ by calling Trim( ); this step takes time linear to the size of $\mathcal{D}^Y_{I,\epsilon}$, which is $O(\frac{1}{\epsilon}\log\log W\log W)$.

6. Further Reducing the Size of $\mathcal{D}^Y_{I,\epsilon}$ for Streams with Small Tardiness

Recall that in an out-of-order data stream with *tardiness* $d_{\max} \in [0, W]$, any item (a, u) arriving at time $\tau_{\texttt{cur}}$ satisfies $u \geq \tau_{\texttt{cur}} - d_{\max}$; in other words, the delay of any item is guaranteed to be at most $d_{\max}$. This section extends $\mathcal{D}^Y_{I,\epsilon}$ to a data structure $\mathcal{E}^Y_{I,\epsilon}$ that takes advantage of this maximum delay guarantee to reduce the space usage. The idea is as follows. Since there is no new item with stamps smaller than $\tau_{\texttt{cur}} - d_{\max}$, we will not make any further change to those nodes to the of left $\tau_{\texttt{cur}} - d_{\max}$ and hence can consolidate these nodes to reduce space substantially. To handle those nodes with timestamps in $[\tau_{\texttt{cur}} - d_{\max}, \tau_{\texttt{cur}}]$, we use the data structure given in Section 5; since it is monitoring an interval of $d_{\max}$ instead of W, its size is $O(\frac{1}{\epsilon}(\log\log d_{\max})\log d_{\max})$ instead of $O(\frac{1}{\epsilon}(\log\log W)\log W)$.

To implement $\mathcal{E}^Y_{I,\epsilon}$, we need a new operation called *consolidate*. Consider any list of queues $\langle Q^a_{J_1,\lambda}, Q^a_{J_2,\lambda}, \ldots, Q^a_{J_m,\lambda}\rangle$, where $J_1, J_2, \ldots, J_m$ are ordered from left to right and form a partition of the interval $J_{1..m} = J_1 \cup \cdots \cup J_m$. We consolidate them into a single queue $Q^a_{J_{1..m},\lambda}$ as follows:

(1) Concatenate the queues into a single queue, in which the nodes preserve the left-right order.
(2) Starting from the leftmost node, check from left to right every node N in the queue, if N is not the rightmost node and $v(N) < \lambda$, merge it with the node N' immediately to its right, *i.e.*, delete N, set $v(N') = v(N) + v(N')$, $d(N') = d(N) + d(N')$ and $\mathcal{I}(N') = \mathcal{I}(N) \cup \mathcal{I}(N')$.

Note that after the consolidation, the resulting queue $Q^a_{J_{1..m},\lambda}$ has at most one node (the rightmost one) with value smaller than λ.

Given the list $\langle \mathcal{C}_{J_1,\lambda,\kappa(c_1)}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)}\rangle$, we consolidate them into $\mathcal{C}_{J_{1..m},\lambda,\frac{1}{\epsilon}}$ by first consolidating, for each item a, the queues $Q^a_{J_1,\lambda}, \ldots, Q^a_{J_m,\lambda}$ in $\mathcal{C}_{J_1,\lambda,\kappa(c_1)}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)}$ into the queue $Q^a_{J_{1..m},\lambda}$ and put it in $\mathcal{C}_{J_{1..m},\lambda,\frac{1}{\epsilon}}$. Then, we apply Lines 3–5 of procedure Process( ) repeatedly to reduce the number of non-trivial queues in the data structure to $\frac{1}{\epsilon}$.

We are now ready to describe how to extend $\mathcal{D}^Y_{I,\epsilon}$ to $\mathcal{E}^Y_{I,\epsilon}$. In our discussion, we fix $\lambda = \frac{\epsilon Y}{\log d_{\max}}$, and without loss of generality, we assume that $I = [1, W]$. Recall that $p_{\texttt{max}}$ denotes the largest timestamp in I such that $\hat{f}_*([p_{\texttt{max}}, r_I]) > (1+\epsilon)Y$ (which implies $f_*([p_{\texttt{max}}, r_I]) > Y$). We partition I into sub-windows $I_1, I_2, \ldots, I_m$, each of size $d_{\max}$ (*i.e.*, $I_i = [(i-1)d_{\max}, id_{\max}]$). We divide the execution into different periods according to $\tau_{\texttt{cur}}$, the current time.

- During the 1st period, when $\tau_{\texttt{cur}} \in [1, d_{\max}] = I_1$, $\mathcal{E}^Y_{I,\epsilon}$ simply is $\mathcal{D}^Y_{I_1,\epsilon}$.
- During the 2nd period, when $\tau_{\texttt{cur}} = I_2$, $\mathcal{E}^Y_{I,\epsilon}$ maintains $\mathcal{D}^Y_{I_2,\epsilon}$ in addition to $\mathcal{D}^Y_{I_1,\epsilon}$.
- During the 3rd period, when $\tau_{\texttt{cur}} \in I_3$, $\mathcal{E}^Y_{I,\epsilon}$ maintains $\mathcal{D}^Y_{I_3,\epsilon}$ in addition to $\mathcal{D}^Y_{I_2,\epsilon}$. Also, the $\mathcal{D}^Y_{I_1,\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa(c_1)}, \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)}\rangle$ is consolidated into $\mathcal{C}_{I_1,\lambda,\frac{1}{\epsilon}}$.
- In general, during the ith period, when $\tau_{\texttt{cur}} \in [(i-1)d_{\max}+1, id_{\max}] = I_i$, $\mathcal{E}^Y_{I,\epsilon}$ maintains $\mathcal{D}^Y_{I_{i-1},\epsilon}$ and $\mathcal{D}^Y_{I_i,\epsilon}$, and also $\mathcal{C}_{I_{1..i-2},\lambda,\frac{1}{\epsilon}}$ where $I_{1..i-2} = I_1 \cup I_2 \cup \cdots \cup I_{i-2}$. Observe that in this period, there is no item (a, u) with $u \in I_{1..i-2}$ arrives (because the tardiness is $d_{\max}$), and thus we do not need to update $\mathcal{C}_{I_{1..i-2},\lambda,\frac{1}{\epsilon}}$. However, we will keep throwing away any node N in $\mathcal{C}_{I_{1..i-2},\lambda,\frac{1}{\epsilon}}$ as soon as we know $i(N)$ is to the left of $p_{\texttt{max}}+1$.
- When entering the $(i+1)$st period, we do the followings: Keep $\mathcal{D}^Y_{I_i,\epsilon}$, create $\mathcal{D}^Y_{I_{i+1},\epsilon}$, merge $\mathcal{C}_{I_{1..i-2},\lambda,\kappa}$ with $\mathcal{D}^Y_{I_{i-1},\epsilon} = \langle \mathcal{C}_{J_1,\lambda,\kappa(c_1)} \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)}\rangle$, and then get $\mathcal{C}_{I_{1..i-1},\lambda,\frac{1}{\epsilon}}$ by consolidating $\langle \mathcal{C}_{I_{1..i-2},\lambda,\frac{1}{\epsilon}}, \mathcal{C}_{J_1,\lambda,\kappa(c_1)} \ldots, \mathcal{C}_{J_m,\lambda,\kappa(c_m)}\rangle$.

Given any $t \in [p_{\texttt{max}} + 1, r_I]$, the estimate of $f_a([t, r_I])$ given by $\mathcal{E}^Y_{I,\epsilon}$ is

$$\hat{f}_a([t, r_I]) = v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})$$

The following theorem gives the accuracy of $\hat{f}_a([t, r_I])$, $\mathcal{E}^Y_{I,\epsilon}$'s size and its update time.

Theorem 8

1. *For any $t \in [p_{\texttt{max}} + 1, r_I]$, the estimate $\hat{f}_a([t, r_I])$ given by $\mathcal{E}^Y_{I,\epsilon}$ satisfies*

$$f_a([t, r_I]) - 2\epsilon Y \leq \hat{f}_a([t, r_I]) \leq f_a([t, r_I]) + 2\epsilon Y$$

2. *$\mathcal{E}^Y_{I,\epsilon}$ has size $O(\frac{1}{\epsilon}(\log\log d_{\max}) \log d_{\max})$, and supports $O(\log \frac{1}{\epsilon} + \log\log d_{\max})$ update time.*

Proof Recall that I is partitioned into sub-intervals $I_1, I_2, \ldots, I_m$. Suppose that $t \in I_k$. Note that if we had not performed any consolidation,

$$v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) = v(\mathsf{alive}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq t}) + \textstyle\sum_{k+1 \leq i \leq m} v(\mathsf{alive}(\mathcal{D}^Y_{I_i,\epsilon})^a)$$

Note that for $k + 1 \leq i \leq m$, $v(\mathsf{alive}(\mathcal{D}^Y_{I_i,\epsilon})^a) \leq f_a(I_i)$, and for $v(\mathsf{alive}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq t})$, since $|I_k| = d_{\max}$, the same argument used in the proof of Lemma 3 gives us $v(\mathsf{alive}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq t}) \leq f_a([t, r_{I_k}]) + \lambda \log d_{\max}$. Hence

$$\begin{aligned} v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) &= v(\mathsf{alive}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq t}) + \textstyle\sum_{k+1 \leq i \leq m} v(\mathsf{alive}(\mathcal{D}^Y_{I_i,\epsilon})^a) \\ &\leq f_a([t, r_{I_k}]) + \lambda \log d_{\max} + \textstyle\sum_{k+1 \leq i \leq m} f_a(I_i) = f_a([t, r_I]) + \lambda \log d_{\max} \end{aligned} \quad (10)$$

The consolidation step may add errors to $v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})$. To get a bound on them, let $N_1, N_2, \ldots$ be the nodes for a in $\mathcal{E}^Y_{I,\epsilon}$, ordered from left to right. Suppose that $t \in N_h$. Note that

- the consolidation step will added at most λ units to $v(N_h)$ before we move on to consider the node immediately to its right, and
- for node N_i with $i \geq h + 1$, any node N that has been merged to N_i must be to the right of of N_h, and thus is to the right of t; it follows that N is contributing $v(N)$ to $v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})$ in Equation (10) and its merging will not make any change.

In conclusion, the consolidation steps introduce at most λ extra errors, and Equation (10) becomes $v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) \leq f_a([t, r_I]) + \lambda \log W + \lambda \leq f_a([t, r_I]) + 2\epsilon Y$, which is the second inequality of the lemma.

To prove the first inequality, suppose that we ask for the estimate $\hat{f}_a([t, r_I])$ during the ith period, when we have $\mathcal{C}_{I_{1..i-2},\lambda,\frac{1}{\epsilon}}$, $\mathcal{D}^Y_{I_{i-1},\epsilon}$ and $\mathcal{D}^Y_{I_i,\epsilon}$. Recall that $\mathcal{C}_{I_{1..i-2},\lambda,\epsilon}$ comes from consolidating $\mathcal{D}^Y_{I_1,\epsilon}, \mathcal{D}^Y_{I_2,\epsilon}, \ldots, \mathcal{D}^Y_{I_{i-2},\epsilon}$. As in all our previous analyses, we have

$$f_a([t, r_I]) - v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) \leq v(\mathsf{node}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) - v(\mathsf{alive}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) = d(\mathsf{dead}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})$$

(Note that the merging of nodes during consolidations would not take away any value). To get a bound on $d(\mathsf{dead}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})$, suppose that $p_{\texttt{max}} \in I_k$. Then, all the nodes to the left of I_k have been thrown away. Among $\mathcal{D}^Y_{I_k,\epsilon}, \mathcal{D}^Y_{I_{k+1},\epsilon}, \ldots, \mathcal{D}^Y_{I_m,\epsilon}$, only $\mathcal{D}^Y_{I_k,\epsilon}$ may have been trimmed. Note that

- $d(\mathsf{dead}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t}) \leq d(\mathsf{dead}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq p_{\max}}) + \sum_{k+1\leq \ell \leq m} d(\mathsf{dead}(\mathcal{D}^Y_{I_\ell,\epsilon})^a)$,
- as in the proof of Theorem 7, we can argue that $d(\mathsf{dead}(\mathcal{D}^Y_{I_k,\epsilon})^a_{\geq p_{\max}}) \leq \epsilon Y$, and
- for the other $\mathcal{D}^Y_{I_\ell,\epsilon}$, since their capacity is at least $\frac{1}{\epsilon}$

$$\textstyle\sum_{k+1\leq \ell \leq m} d(\mathsf{dead}(\mathcal{D}^Y_{I_\ell,\epsilon})^a) \leq \sum_{k+1\leq \ell\leq m} f_*(I_\ell)/(1/\epsilon) \leq \epsilon f_*([p_{\max}+1, r_I]) \leq \epsilon Y$$

Thus, $d(\mathsf{dead}(\mathcal{E}^Y_{I,\epsilon})^a_{\geq t})) \leq 2\epsilon Y$, and the first inequality follows.

For Statement (2), note that both $\mathcal{D}^Y_{I_{i-1},\epsilon}$ and $\mathcal{D}^Y_{I_i,\epsilon}$ have size $O(\frac{1}{\epsilon}\log\log d_{\max}\log d_{\max})$ (by Theorem 7, and $|I_{i-1}| = |I_i| = d_{\max}$), and for $\mathcal{C}_{J_{1..i-2},\lambda,\frac{1}{\epsilon}}$, it has size $O(Y/\lambda + \frac{1}{\epsilon}) = O(\frac{1}{\epsilon}\log d_{\max})$; thus the size of $\mathcal{E}^Y_{I,\epsilon}$ is $O(\frac{1}{\epsilon}\log\log d_{\max}\log d_{\max})$. For the update time, it suffices to note that it is dominated by the update times of $\mathcal{D}^Y_{I_{i-1},\epsilon}$ and $\mathcal{D}^Y_{I_i,\epsilon}$.

Acknowledgements

H.F. Ting is partially supported by the GRF Grant HKU-716307E; T.W. Lam is partially supported by the GRF Grant HKU-713909E.

References

1. Cormode, G.; Korn, F.; Tirthapura, S. Time-Decaying Aggregates in Out-of-Order Streams. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'08*, Vancouver, Canada, 9–11 June 2008; pp. 89–98.
2. Karp, R.; Shenker, S.; Papadimitriou, C. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.* **2003**, *28*, 51–55.
3. Demaine, E.; Lopez-Ortiz, A.; Munro, J. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th Annual European Symposium, ESA'07*, Rome, Italy, 17–21 September 2002; pp. 348–360.
4. Muthukrishnan, S. *Data Streams: Algorithms and Applications*; Now Publisher Inc.: Boston, MA, USA, 2005.
5. Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'02*, Madison, WI, USA, 3–5 June 2002; pp. 1–16.
6. Arasu, A.; Manku, G. Approximate Counts and Quantiles over Sliding Windows. In *Proceedings of the 23th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'04*, Paris, France, 14–16 June 2004; pp. 286–296.
7. Lee, L.K.; Ting, H.F. A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. In *Proceedings of the PODS*, June 26–28, 2006, Chicago, Illinois, USA; pp. 290–297.
8. Lee, L.K.; Ting, H.F. Maintaining Significant Stream Statistics over Sliding Windows. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'06*, Miami, FL, USA, 22–26 January 2006; pp. 724–732.
9. Datar, M.; Gionis, A.; Indyk, P.; Motwani, R. Maintaining stream statistics over sliding windows. *SIAM J. Comput.* **2002**, *31*, 1794–1813.

10. Tirthapura, S.; Xu, B.; Busch, C. Sketching Asynchronous Streams over a Sliding Window. In *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC'06*, Denver, CO, USA, 23–26 July 2006; pp. 82–91.
11. Busch, C.; Tirthapua, S. A Deterministic Algorithm for Summarizing Asynchronous Streams over a Sliding Window. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science, STACS'07*, Aachen, Germany, 22–24 February 2007; pp. 465–475.
12. Cormode, G.; Tirthapura, S.; Xu, B. Time-decaying sketches for robust aggregation of sensor data. *SIAM J. Comput.* **2009**, *39*, 1309–1339.
13. Chan, H.L.; Lam, T.W.; Lee, L.K.; Ting, H.F. Approximating Frequent Items in Asynchronous Data Stream over a Sliding Window. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms, WAOA'09*, Copenhagen, Denmark, 10–11 September 2009; pp. 49–61.
14. Misra, J.; Gries, D. Finding repeated elements. *Sci. Comput. Program.* **1982**, *2*, 143–152.
15. Arbitman, Y.; Naor, M.; Segev, G. De-amortized Cuckoo Hashing: Provable Worst-Case Performance and Experimental Results. In *Proceedings of the 36th International Colloquium, ICALP'09*, Rhodes, Greece, 5–12 July 2009; pp. 107–118.
16. Hung, R.S.; Lee, L.K.; Ting, H.F. Finding frequent items over sliding windows with constant update time. *Inf. Process. Lett.* **2010**, *110*, 257–260.