Supplementary Materials of HyperCalm Sketch

1 SIMD Implementation Details

Single instruction, multiple data (SIMD) [1] is a widely-used data parallel processing technology that can perform the same operation on multiple data simultaneously. This technology well suits the data structure of the HyperCalm sketch using Bucketized Partition. Below we describe how to use SIMD instructions to accelerate HyperCalm.

HyperBF acceleration: In HyperBF, for each incoming item, we first locate three hashed blocks and clean all cells in these blocks. To ensure that outdated cells can be cleaned timely, the block size is set to 64B (cache line size). In our basic implementation, we use a loop to clean these cells, which can be accelerated using SIMD. Currently, AVX-512 instruction set provides 512-bit wide SIMD registers (ZMM). We can load a block into one 512-bit register and use 512-bit bit operations to efficiently clean this block, eliminating the complicated loops and improving efficiency.

Bucketized TimeRecorder/CalmSS acceleration: In bucketized TimeRecorder/LRU-Queue/Space-Saving, we treat the b slots as uniform data points and uses SIMD instructions to simultaneously process them in parallel. To ensure memory continuity, in each bucket, we record IDs (or fingerprints) and frequencies (or timestamps) in two arrays separately, which are called the ID array and the information array. In insertion process, we first check the ID array. If we find a matched ID/fingerprint, we update the corresponding frequency/timestamp. Otherwise, we check the information array to find the item/entry to be evicted (*e.g.*, the least recent item in TimeRecorder). We propose three SIMD acceleration techniques (matching acceleration, sorting acceleration, and find-min acceleration), to accelerate the process of ID/fingerprint matching, sorting items according to time order, and finding the minimum counter.

1) Matching acceleration: In bucketized TimeRecorder/LRU-Queue/Space-Saving, we match an ID/fingerprint with all IDs/fingerprints in the hashed bucket to check whether an item exists in this bucket, which can be accelerated with SIMD instructions. For example, when $b = 8$ and using 32-bit IDs, we can use the `_mm256_cmpeq_epi32_mask` instruction to compare the ID of the incoming item with 8 IDs in parallel. In this way, we reduce the time complexity of the matching procedure from $O(b)$ to $O(1)$.

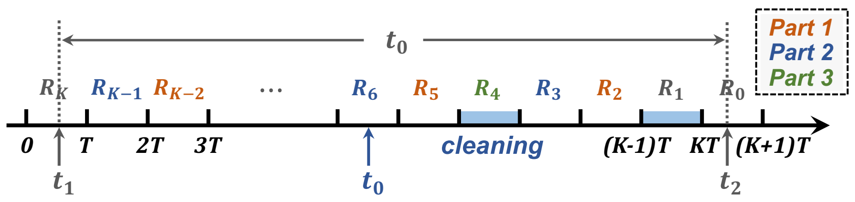
2) Sorting acceleration: In bucketized TimeRecorder/LRU-Queue, we need to sort the items/entries in each bucket according to time order, which can be accelerated using SIMD. The key idea is to use `_mm_shuffle_epi8` instruction to rearrange each byte into proper predefined order. For example, when $b = 8$ and using 16-bit fingerprints, if we want to move the 2^{th} item into the first slot, we can first define the proper order as `__m128i index = _mm_setr_epi8(2,3,0,1,4,5,6,7,8,9,10,11,12, 13,14,15);` Then we can use `_mm_shuffle_epi8(fps, index)` instruction to rearrange bytes in the 128-bit register (*i.e.*, the register storing 8 16-bit fingerprints) so as to sort the 8 fingerprints according to time order. In this way, we reduce the time complexity of the sorting procedure from $O(b)$ to $O(1)$.

3) Find-min acceleration: In bucketized Space-Saving, we need to find the least frequent entry, which can be accelerated using SIMD. The key idea is to use SIMD instructions to repeatedly compare the first half and second half of counters. For example, when $b = 4$ and using 32-bit counters, we can first use `_mm_shuffle_epi8` instruction to move the second half of counters to the first half. Then we use `_mm_min_epi32` instruction to compare the first half and second half of counters and get the packed minimum values. We repeat the above procedure until we finally get the minimum counter. In this way, we reduce the time complexity of the find-min procedure from $O(b)$ to $O(log(b))$.

2 Mathematical Proofs

2.1 Error Rate of HyperBF

We first prove the error rate of HyperBF in Theorem 4.1. A data stream can be formulated by two variables: density α and activity β, where density α is the number of distinct items observed at each moment, and activity β is the number of distinct items emerging/dying per unit time. Consider two consecutive time interval T_1 and T_2. The numbers of distinct items observed in T_1 and T_2 are $\alpha + \beta T_1$ and $\alpha + \beta T_2$, respectively. And the number of distinct items observed in the two intervals is $\alpha + \beta(T_1 + T_2)$. Most data streams can be formulated by these two variables. Take CAIDA [2] dataset as an instance, Figure 2(a) shows the average number (±5std) of distinct items observed in time intervals of different length. We can see that the linear relationship almost holds where $\alpha = 3195.2$ and $\beta = 35238.9$. Next, consider two adjacent occurrences of item e at t_1 and t_2, where $t_2 - t_1 > 2\mathcal{T}$. Let $K = \lfloor \frac{t_2}{\mathcal{T}} \rfloor - \lfloor \frac{t_1}{\mathcal{T}} \rfloor$. Let $\gamma_n = \alpha + \beta n\mathcal{T}$ denote the number of distinct items observed in a time interval of length $n\mathcal{T}$.



t_0
Part 1
Part 2
Part 3
R_K R_{K-1} R_{K-2} ⋯ R_6 R_5 R_4 R_3 R_2 R_1 R_0
0 T 2T 3T cleaning (K-1)T KT (K+1)T
t_1
t_0
t_2

Figure 1: Error rate analysis of HyperBF.

As shown in Figure 1, consider two adjacent occurrences of an item e in the data stream. Assume the timestamps of the two occurrences are t_1 and t_2, respectively. Assume $t_2 - t_1 > 2\mathcal{T}$, meaning that the second occurrence of e is the start of a batch and there is no *time division error*. Next, we derive the *error rate* of HyperBF, which is defined as the probability that HyperBF does not report a batch at t_2.

Now consider a certain hashed cell $\mathcal{B}_i[h_i(e)]$ of item e. It is obvious that $\mathcal{B}_i[h_i(e)]$ is accessed by e at both t_1 and t_2. Let t_0 be the last time that $\mathcal{B}_i[h_i(e)]$ was accessed before t_2. We have $t_1 \leqslant t_0 < t_2$. In particular, if $t_0 = t_1$, we can assert that there is no item hashed into $\mathcal{B}_i[h_i(e)]$ between t_1 and t_2. Assume that $0 \leqslant t_1 < \mathcal{T}$ and $K\mathcal{T} \leqslant t_2 < (K+1)\mathcal{T}$, *i.e.*, $K = \lfloor \frac{t_2}{\mathcal{T}} \rfloor - \lfloor \frac{t_1}{\mathcal{T}} \rfloor$. Since t_1 and t_2 are two arbitrary selected timestamps, and the timeline can be arbitrarily specified, the above assumption does not impair generality. Next, we use symbol R_i $(0 \leqslant i \leqslant K)$ to denote the time interval $[(K-i)\mathcal{T}, (K+1-i)\mathcal{T})$. We restrict the range of possible t_0 into $[\mathcal{T}, (K+1)\mathcal{T})$. And we can assume that $\mathcal{B}_i[h_i(e)]$ can be cleaned by other items only in $[0, K\mathcal{T})$.

Lemma 4.1. *Let $\mathcal{A}'_j$ be the event that $t_0 \leqslant (K+1-j)\mathcal{T}$. Let $j' = \frac{(j-K-1)\mathcal{T}+t_2}{\mathcal{T}}$. We have $\forall 1 \leqslant j < K$, $\Pr\left(\mathcal{A}'_j\right) \approx e^{-\frac{\gamma_{j'}}{m}}$.*

Proof. The probability that $\mathcal{B}_i[h_i(e)]$ is not selected by a certain hash function during the insertion of an item is $1 - \frac{1}{m}$.

Note that $t_0 \leqslant (K+1-j)\mathcal{T}$ is equivalent to the statement that $\mathcal{B}_i[h_i(e)]$ is not selected by any item that arrives between $(K+1-j)\mathcal{T}$ and t_2.

From our data stream assumption, the number of distinct items arrives between $(K+1-j)\mathcal{T}$ and t_2 is

$$\gamma_{j'} = \alpha + \beta(j-K-1)\mathcal{T} + \beta t_2$$

Since the hash values of distinct items have no significant correlation between each other, then the probability that $\mathcal{B}_i[h_i(e)]$ is not selected by any of the $\gamma_{j'}$ distinct items is

$$\Pr\left(\mathcal{A}'_j\right) = \left(1 - \frac{1}{m}\right)^{\gamma_{j'}} \approx e^{-\frac{\gamma_{j'}}{m}}$$

□

Lemma 4.2. *For $\forall 0 \leqslant j < K$, let $\mathcal{A}'_j$ denote the event that $t_0 \leqslant (K-j+1)\mathcal{T}$, and let $\mathcal{A}_j$ denote the event that $(K-j)\mathcal{T} \leqslant t_0 < (K-j+1)\mathcal{T}$, i.e., the event that $t_0 \in R_j$. Then we have that for $\forall 1 \leqslant j < K$,*

$$\Pr\left(\mathcal{A}_j\right) > e^{-\frac{\gamma_j}{m}} - e^{-\frac{\gamma_{j+1}}{m}}$$

Proof. It is obvious that for $\forall 0 \leqslant j < K$, we have

$$\Pr\left(\mathcal{A}'_j\right) = \Pr\left(\mathcal{A}'_{j+1}\right) + \Pr\left(\mathcal{A}_j\right)$$

According to Lemma 4.1, we have that for $\forall 1 \leqslant j < K$:

$$\begin{aligned}\Pr\left(\mathcal{A}_j\right) = \Pr\left(\mathcal{A}'_j\right) - \Pr\left(\mathcal{A}'_{j+1}\right) &\approx e^{-\frac{\gamma_{j'}}{m}} - e^{-\frac{\gamma_{j'+1}}{m}} \\ &= e^{-\frac{\alpha+\beta(j-K-1)\mathcal{T}+\beta t_2}{m}}\left(1 - e^{-\frac{\beta\mathcal{T}}{m}}\right) \\ &= e^{-\frac{\gamma_{j'}}{m}}\left(1 - e^{-\frac{\beta\mathcal{T}}{m}}\right)\end{aligned}$$

Since $t_2 < (K+1)\mathcal{T}$, we have that $\gamma_{j'} = \alpha + \beta(j-K-1)\mathcal{T} + \beta t_2 < \alpha + \beta j\mathcal{T} = \gamma_j$. Thus, we have

$$\begin{aligned}\Pr\left(\mathcal{A}_j\right) = e^{-\frac{\gamma_{j'}}{m}}\left(1 - e^{-\frac{\beta\mathcal{T}}{m}}\right) &> e^{-\frac{\gamma_j}{m}}\left(1 - e^{-\frac{\beta\mathcal{T}}{m}}\right) \\ = e^{-\frac{\alpha+\beta j\mathcal{T}}{m}} - e^{-\frac{\alpha+\beta(j+1)\mathcal{T}}{m}} &= e^{-\frac{\gamma_j}{m}} - e^{-\frac{\gamma_{j+1}}{m}}\end{aligned}$$

□

Lemma 4.3. *Let $m' = \frac{m}{l-1}$. Given u time intervals of length $\mathcal{T}$, $T_1, \cdots, T_u$, let $\mathcal{C}_u$ be the event that a certain cell, e.g., $\mathcal{B}_i[h_i(e)]$, is cleaned at least once in these time intervals. We have:*

$$1 - e^{-\frac{\gamma_u}{m'}} \leqslant \Pr(\mathcal{C}_u) \leqslant 1 - e^{-\frac{u\gamma_1}{m'}}$$

Proof. For each incoming item e_i, $\mathcal{B}_i[h_i(e)]$ is cleaned if and only if e_i is hashed into the same block but not the same cell with e. Therefore, the probability that $\mathcal{B}_i[h_i(e)]$ is not cleaned during the insertion of e_i is $1 - \frac{l-1}{m} = 1 - \frac{1}{m'}$.

Let x be the number of distinct items in the u intervals, $T_1, \cdots, T_u$.

According to the data stream assumption, we have $\gamma_u \leqslant x \leqslant u\gamma_1$. When the u intervals are consecutive, we have $x = \gamma_u$, and when the u intervals are disjoint and separated far away enough from each other, we have $x = u\gamma_1$.

Since the probability that $\mathcal{B}_i[h_i(e)]$ is not cleaned in $T_1, \cdots, T_u$ is $\left(1 - \frac{1}{m'}\right)^x \approx e^{-\frac{x}{m'}}$, we have

$$\Pr(\mathcal{C}_u) = 1 - e^{-\frac{x}{m'}}$$

Thus, we have

$$1 - e^{-\frac{\gamma_u}{m'}} \leqslant \Pr(\mathcal{C}_u) \leqslant 1 - e^{-\frac{u\gamma_1}{m'}}$$

□

Lemma 4.4. *Let P be the probability that a certain hashed cell of item e (e.g., $\mathcal{B}_i[h_i(e)]$) is zero at t_2. Let $m' = \frac{m}{l-1}$ and $u_j = \left\lceil \frac{j-2}{3} \right\rceil$. Let $K_1 = \left\lfloor \frac{K}{3} \right\rfloor - 1$, $K_2 = \left\lfloor \frac{K-1}{3} \right\rfloor - 1$, and $K_3 = \left\lfloor \frac{K-2}{3} \right\rfloor - 1$. Then the lower bound of P is $P' = P'_1 + P'_2 + P'_3$, where $P'_1 = \sum_{k=0}^{K_1} \left(e^{-\frac{\gamma_{3k+2}}{m}} - e^{-\frac{\gamma_{3k+3}}{m}}\right)$, $P'_2 = \sum_{k=0}^{K_2} \left(e^{-\frac{\gamma_{3k+3}}{m}} - e^{-\frac{\gamma_{3k+4}}{m}}\right)\left(1 - e^{-\frac{\gamma_{u_{3k+3}}}{m'}}\right)$, and $P'_3 = \sum_{k=0}^{K_3} \left(e^{-\frac{\gamma_{3k+4}}{m}} - e^{-\frac{\gamma_{3k+5}}{m}}\right)\left(1 - e^{-\frac{\gamma_{u_{3k+4}}}{m'}}\right)$.*

Proof. Now we discuss the possible range of t_0. Since our goal is to derive the lower bound of P, we can just ignore the case where $t_0 \in R_K$. And we note that when $t_0 \in R_0$ or $t_0 \in R_1$, $\mathcal{B}_i[h_i(e)]$ cannot be *zero* at t_2. Therefore, we only discuss the cases where $\mathcal{T} \leqslant t_0 < (K-1)\mathcal{T}$, *i.e.*, the cases where $t_0 \in R_j$ $(2 \leqslant j \leqslant K-1)$.

First, consider the cases where $t_0 \in R_{3k+2}$ $(0 \leqslant k \leqslant \left\lfloor \frac{K}{3} \right\rfloor - 1)$. In these cases, $\mathcal{B}_i[h_i(e)]$ will be cleaned to *zero* when inserting item e at t_2. Let $K_1 = \left\lfloor \frac{K}{3} \right\rfloor - 1$. We can derive the first part of P as

$$P_1 = \sum_{k=0}^{K_1} \Pr(\mathcal{A}_{3k+2}) = \sum_{k=0}^{K_1} \left(e^{-\frac{\gamma_{3k+2}}{m}} - e^{-\frac{\gamma_{3k+3}}{m}}\right) = P'_1$$

Second, consider the cases where $t_0 \in R_{3k+3}$ $(0 \leqslant k \leqslant \left\lfloor \frac{K-1}{3} \right\rfloor - 1)$. As shown in Figure 1, when $t_0 \in R_6$, in order to guarantee that $\mathcal{B}_i[h_i(e)] = 0$ at t_2, $\mathcal{B}_i[h_i(e)]$ must be cleaned at least once in time intervals R_4 and R_1. Generally, when $t_0 \in R_j$, let u_j denote the number of intervals in which $\mathcal{B}_i[h_i(e)]$ should be cleaned at least once. Then we have $u_j = \left\lceil \frac{j-2}{3} \right\rceil$. Let $K_2 = \left\lfloor \frac{K-1}{3} \right\rfloor - 1$. We can derive the second part of P as

$$P_2 = \sum_{k=0}^{K_2} \Pr(\mathcal{A}_{3k+3}) \Pr(\mathcal{C}_{u_{3k+3}}) \geqslant \sum_{k=0}^{K_2} \left(e^{-\frac{\gamma_{3k+3}}{m}} - e^{-\frac{\gamma_{3k+4}}{m}}\right)\left(1 - e^{-\frac{\gamma_{u_{3k+3}}}{m'}}\right) = P'_2$$

Third, consider the cases where $t_0 \in R_{3k+4}$ $(0 \leqslant k \leqslant \left\lfloor \frac{K-2}{3} \right\rfloor - 1)$. These cases are similar to the cases in the second part, and the proof is also similar.

Finally, we have $P = P_1 + P_2 + P_3 \geqslant P'_1 + P'_2 + P'_3$.

□

Theorem 4.1. *We define the error rate $\mathcal{E}$ of HyperBF (without Asynchronous Timeline) as the probability that HyperBF does not report a batch at t_2. Then we have:*

$$\mathcal{E} \leqslant (1 - P')^d$$

where P' is the lower bound in Lemma 4.4.

Proof. HyperBF does not report a batch at t_2 if and only if all d hashed cells $\mathcal{B}_1[h_1(e)], \cdots, \mathcal{B}_d[h_d(e)]$ are *zero* at t_2. Since the d arrays of HyperBF are independent of each other, we have that $\mathcal{E} = (1 - P)^d$, where P is the probability that one hashed cell is *zero* at t_2. Thus, we have $\mathcal{E} \leqslant (1 - P')^d$, where P' is the lower bound of P in Lemma 4.4. □

Experimental analysis (Figure 2(b)): We conduct experiments on CAIDA [2] to validate the bound in Lemma 4.4. We use the HyperBF that just has one array ($d = 1$), and allocate 4KB of memory to it ($m = 16000$). The results show that the experimental error rate is always well bounded by theoretical bound. As the volume of CAIDA data stream is very large, almost all outdated cells in HyperBF can be cleaned promptly. Therefore, the experimental error rate does not vary with K. As K grows larger, our theoretical bound becomes more accurate. Note that we only focus on a single array of HyperBF here. If we use the HyperBF consisting of $d = 8$ arrays, the error rate will be < 0.01.

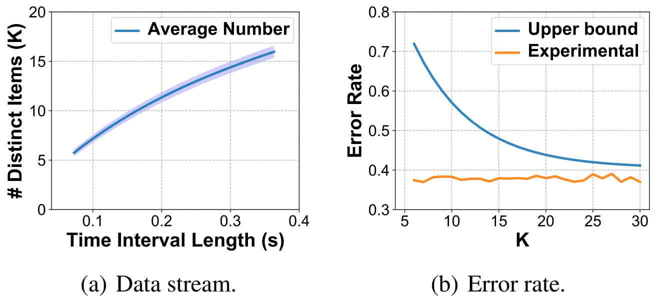


Figure 2: Error rate of HyperBF.

2.2 Error Rate of CalmSS

We define the error rate ζ of CalmSS as the probability that a cold item fails to be discarded by LRU queue, *i.e.*, the probability that a cold item enters the top-k algorithm in CalmSS. Next, we derive the upper bound of ζ.

We assume the data stream consists of two types of items: cold items and hot items, and all items of the same type have the same arrival speed. The data stream is essentially the sum of many independent Poisson processes of two kinds (hot items and cold items). Let λ_h and λ_c be the parameters of the two Poisson processes, respectively. Let n_h and n_c be the number of distinct hot items and cold items, respectively. Notice that $n_h \gg w$ and $n_c \gg w$. Therefore, we can assume that in a short time interval, all arriving items are distinct. Consider a cold item e, we assume all items that arrives between the time when e enters the LRU queue and the time when e is removed from the LRU queue are distinct *hot* items. Here, we assume all of these items are hot because we want to derive an upper bound of ζ. Cold items only promote the LRU queue to discard e, resulting in a smaller ζ.

Theorem 4.2. *For a cold item e, the probability ζ that it fails to be discarded by CalmSS, i.e., the error rate of CalmSS, is*

$$\zeta = \left(\sum_{x=0}^{w-1} \frac{1}{x!} \cdot \frac{R^x}{(R+1)^{x+1}} \cdot \Gamma(x+1) \right)^{\mathcal{P}-1}$$

where $R = \frac{n_h \lambda_h}{\lambda_c}$, and $\Gamma(z)$ represents the Gamma function.

Proof. Suppose e fails to be discarded by the LRU queue, *i.e.*, it enters the top-k algorithm. Then e must arrives $\mathcal{P}$ times in a short time, and each arrival increments the counter of e in the LRU queue by one. Let random variables $T_1, T_2, \cdots, T_{\mathcal{P}-1}$ be the time gaps between every two adjacent occurrences of e. As e arrives according to a Poisson process of intensity λ_c, we have T_i follows an exponential distribution with mean λ_c.

Let $\mathcal{D}_i$ denote the event that there arrive w hot items within T_i. It is clear that $\mathcal{D}_1, \cdots, \mathcal{D}_{\mathcal{P}-1}$ are independent of each other. Recall that the Poisson processes of different hot items are independent of each other. Let $\lambda_1 = n_h\lambda_h$ and $\lambda_2 = \lambda_c$. The probability of the event that there arrive x hot items within T is $P_{x,T} = \frac{(\lambda_1 T)^T}{x!} e^{-\lambda_1 T}$.

Then we have:

$$\Pr(\mathcal{D}_i) = \sum_{x=0}^{w-1} P_{x,T_i} = \sum_{x=0}^{w-1} \frac{(\lambda_1 T_i)^{T_i}}{x!} e^{-\lambda_1 T_i}$$

Recall that the Poisson processes of all distinct items are independent of each other. Then we have:

$$\begin{aligned} \zeta =& \mathbb{E}_{\{T_1,\cdots,T_{\mathcal{P}-1}\}} \left[\Pr\left(\prod_{i=1}^{\mathcal{P}-1} \mathcal{D}_i \right) \right] \\ =& \mathbb{E}_{\{T_1,\cdots,T_{\mathcal{P}-1}\}} \left[\prod_{i=1}^{\mathcal{P}-1} \Pr(\mathcal{D}_i) \right] = \prod_{i=1}^{\mathcal{P}-1} \mathbb{E}_{T_i} \left[\Pr(\mathcal{D}_i) \right] \end{aligned}$$

Further, we have:

$$\zeta = \prod_{i=1}^{\mathcal{P}-1} \mathbb{E}_{T_i}\left[\Pr(\mathcal{D}_i)\right] = \prod_{i=1}^{\mathcal{P}-1} \sum_{x=0}^{w-1} \int_0^{+\infty} \frac{(\lambda_1 t_i)^x}{x!} e^{-\lambda_2 t_i} \mathrm{d}t_i$$
$$= \left(\sum_{x=0}^{w-1} \frac{1}{x!} \cdot \frac{R^x}{(R+1)^{x+1}} \cdot \Gamma(x+1) \right)^{\mathcal{P}-1}$$

where $R = \frac{\lambda_1}{\lambda_2}$ and $\Gamma(z)$ represents the Gamma function. □

Experimental analysis (Figure 3): We conduct experiments to validate our theoretical bound in Theorem 4.2. We set $w = 16$, $\mathcal{P} = 4$, and generate the data stream using two kinds of Poisson processes where $n_h = 50$ and $n_c = 1$. The results show that the experimental error rate is always bounded by the theoretical upper bound. Note that when $R < 50$, the intensity of cold items λ_c is smaller than the intensity of hot items λ_h, meaning that cold items are actually not cold. Therefore, when R is small, CalmSS has large theoretical and experimental error. As R increases, our data stream assumption will be closer to truth. When $R > 50$, $\lambda_c < \lambda_h$, meaning that the cold items are really cold.

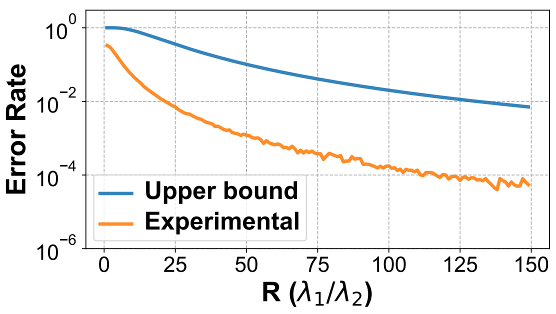


Figure 3: Error of CalmSS.

2.3 Effectiveness of Asynchronous Timeline

Theorem 4.3. *After using the Asynchronous Timeline technique, the time division error is minimized when the d timelines are evenly distributed, i.e., when the timeline offset for the i^{th} array is $o_i = \frac{(i-1)}{d}\mathcal{T}$, where the minimized error is reduced by d times compared to the synchronous version.*

Proof. The time division error occurs only when the batch interval spans two time slices in all of the d timelines. Without loss of generality, we take the first timeline as the reference timeline, *i.e.*, we set $o_1 = 0$, and we suppose $o_1 < o_2 < \cdots < o_d < \mathcal{T}$. Consider two batches arrives at timestamps $x\mathcal{T}$ and $(2-y)\mathcal{T}$ respectively, where $x > 0$, $y > 0$, and $x + y < 1$. The interval between the two batches is $\Delta t = (2 - y - x)\mathcal{T}$. It is clear that $\mathcal{T} < \Delta t < 2\mathcal{T}$. Consider the i^{th} timeline with offset $o_i \in [0, \mathcal{T})$, it can correctly perceive the second batch if and only if $x\mathcal{T} < o_i$ and $(2-y)\mathcal{T} > \mathcal{T} + o_i$. In other words, the i^{th} timeline can correctly perceive the second batch if the interval meets the above two constraints.

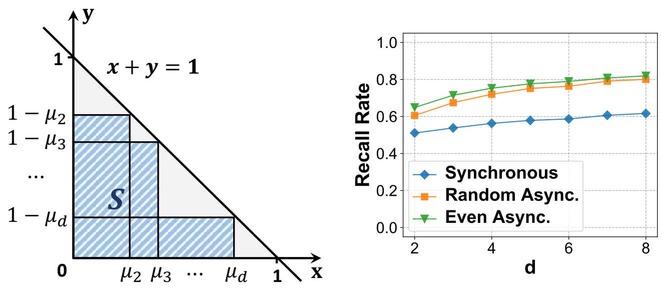
Our goal is to find the optimal $o_2, \cdots, o_d \in [0, \mathcal{T})$ to perceive as much intervals as possible. Suppose the valid values of x and y are uniformly distributed, the above problem can be transformed into a linear programming problem. As shown in Figure 4(a), the triangular area under the line $x + y = 1$ represents the feasible range of x and y. Let $o_i = \mu_i\mathcal{T}$ where $\mu_i \in [0, 1)$. Each timeline with offset o_i enables the intervals lie in the rectangular area $x > 0$, $y > 0$, $x < \mu_i$, and $y < 1 - \mu_i$ to be correctly perceives. Therefore, the goal of the linear programming is to maximize the total area S, which is the union of the $d - 1$ rectangles. And we have:

$$S = \mu_2(1-\mu_2) + (\mu_3 - \mu_2)(1-\mu_3) + (\mu_4 - \mu_3)(1-\mu_4)$$
$$+ \cdots + (\mu_d - \mu_{d-1})(1-\mu_d)$$

By applying the method of Lagrange multipliers, we can easily derive that S is maximized when $\mu_i = \frac{(i-1)}{d}$, *i.e.*, $o_i = \frac{(i-1)}{d}\mathcal{T}$, and the maximized S is $\frac{d-1}{d}$ times of the triangular area. Thus, *asynchronous timeline* reduces the error by d times.

□

Experimental analysis (Figure 4(b)): We conduct experiments on CAIDA [2] to validate Theorem 4.3. We set the batch threshold $\mathcal{T}$ to 1.454 μs, and fix the memory usage of HyperBF to 50KB. We find that *Asynchronous Timeline* technique significantly improves the accuracy of HyperBF. We also find that when using *Asynchronous Timeline*, HyperBF using d evenly distributed timelines is more accurate than HyperBF using d randomly distributed timelines. For example, when using $d = 8$ arrays, the RR of the basic HyperBF is 61%, while that of the HyperBF using *Asynchronous Timeline* is about 80%. Specifically, the RR of the HyperBF using randomly distributed timelines is 80.07%, while that of the HyeprBF using evenly distributed timelines is 81.95%.



(a) Linear programming. (b) Time division error.

Figure 4: Asynchronous Timeline analysis.

2.4 Error Rate of Bucketized Space-Saving

We theoretically analyze the error of bucketized Space-Saving in estimating the periodicities of periodic batches. We first prove bucketized Space-Saving inherits the property of basic Space-Saving in Theorem 4.4. Then we derive an error bound that is related to the parameters of bucketized Space-Saving in Theorem 4.6

Theorem 4.4. *For an arbitrary entry E that is recorded in bucketized Space-Saving, let f and $\hat{f}$ be the real and estimated frequency of E respectively (i.e., f and $\hat{f}$ are the real and estimated periodicities of periodic batch E). We have that*

$$0 \leqslant \hat{f} - f \leqslant f_{min}$$

where f_{min} is the smallest counter in the hashed bucket of E.

Proof. We prove the above theorem by induction on each entry E_k that is inserted into bucketized Space-Saving.

Base case: When bucketized Space-Saving is empty, it is obvious that the inequality holds.

Induction step: Suppose the theorem holds after the first $k - 1$ inserted entries. Now we consider the k^{th} inserted entry E_k. There are three cases as follows.

Case 1: E_k is recorded in the hashed bucket.

In this case, we just increment the counter of E_k. After inserting E_k, f_{min} monotonically increases, and for any recorded entry E, $\hat{f} - f$ remains unchanged. Thus the inequalities still hold.

Case 2: E_k is not recorded, and its hashed bucket has empty slot.

In this case, we just insert $(E_k, 1)$ into the empty slot. After inserting E_k, f_{min} monotonically increases, and for any recorded entry E, $\hat{f} - f$ remains unchanged. Thus, the inequalities still hold.

Case 3: E_k is not recorded, and its hashed bucket is full.

In this case, we find the slot recording the entry with the smallest frequency, namely (E_{min}, f_{min}), and update this slot to $(E_k, f_{min}+1)$. After inserting E_k, among all recorded entries, only E_k has changed real/estimated frequency. Let f'_k and $\hat{f}'_k$ be the real and estimated frequency of E_k after the k^{th} insertion, and let f'_{min} be the smallest counter of the hashed bucket of E_k after the k^{th} insertion. Let f_k and $\hat{f}_k$ be the real and estimated frequency of E_k before the k^{th} insertion. We have $\hat{f}'_k - f'_k = f_{min} + 1 - f_k - 1 = f_{min} - f_k \leqslant f_{min} \leqslant f'_{min}$. Thus, the inequality on the right side holds.

To prove the inequality on the left side, we consider two subcases: 1) E_k appears for the first time. In this subcase, $f'_k = 1$. We have $\hat{f}'_k - f'_k = f_{min} + 1 - 1 = f_{min} \geqslant 0$. 2) E_k appears not for the first time. Let k_0 be the last time when E_k was recorded in bucketized Space-Saving, *i.e.*, E_k was evicted from Space-Saving at the $(k_0 + 1)^{th}$ insertion. Let f_{k_0} and $\hat{f}_{k_0}$ be the real and estimated frequency of E_k at k_0, respectively. Let f_{min_0} be the smallest counter in the hashed bucket of E_k at k_0. Since E_k was evicted at the $(k_0 + 1)^{th}$ insertion, we have $\hat{f}_{k_0} = f_{min_0}$ We have $\hat{f}'_k - f'_k = f_{min} + 1 - f_{k_0} - 1 = f_{min} - f_{k_0} \geqslant f_{min_0} - f_{k_0} = \hat{f}_{k_0} - f_{k_0} \geqslant 0$. Thus, the inequality on the left side holds.

Conclusion: We have proved that the theorem holds for both the base case and the induction step. Therefore, the theorem holds for any k, and thus holds for the entire data stream. □

Lemma 4.5. *For an arbitrary entry E with real frequency $f > f_{min}$, it must exist in bucketized Space-Saving.*

Proof. Assume E is not in bucketized Space-Saving, and assume f'_{min} is the smallest counter in the hashed bucket of E at the time when E was last evicted. From Theorem 4.4, we have $f'_{min} \geqslant f$. Since $f_{min} > f'_{min}$, we have $f_{min} > f$, which contradicts the premise of $f > f_{min}$. $\square$

Theorem 4.5. *For an arbitrary entry E with real frequency $f > \gamma||f||_1$, let $\mathcal{P}$ be the probability that E is recorded in bucketized Space-Saving. We have that*

$$\mathcal{P} \geqslant 1 - \frac{1}{b\gamma z}$$

where $||f||_1$ is the frequency sum of all entries, b is the number of slots in each bucket, and z is the number of buckets.

Proof. Let f_{sum} be the frequency sum of all items in the hashed bucket of E, we have $\mathbb{E}[f_{sum}] = \frac{||f||_1}{z}$. If $f_{sum} < b\gamma||f||_1$, we have $f_{sum} < bf$, meaning that $f_{min} < f$. From Lemma 4.5, we have that E must exist in bucketized Space-Saving. According to Markov inequality, we have

$$\Pr\left(f_{sum} > b\gamma||f||_1\right) \leqslant \frac{\mathbb{E}\left(f_{sum}\right)}{b\gamma||f||_1} \leqslant \frac{1}{b\gamma z}$$

Therefore, we have that $\mathcal{P} \geqslant 1 - \frac{1}{b\gamma z}$. $\square$

Theorem 4.6. *For an arbitrary entry E that is recorded in bucketized Space-Saving, let f and $\hat{f}$ be the real and estimated frequency of E respectively. We have that*

$$\Pr\left(\hat{f} - f \leqslant \epsilon \cdot \frac{||f||_1}{S}\right) \geqslant 1 - \frac{1}{\epsilon}$$

where $||f||_1$ is the frequency sum of all entries, and $S = zb$ is the number of slots in bucketized Space-Saving.

Proof. Let f_{min} be the smallest counter in the hashed bucket of E. According to Theorem 4.4, we have

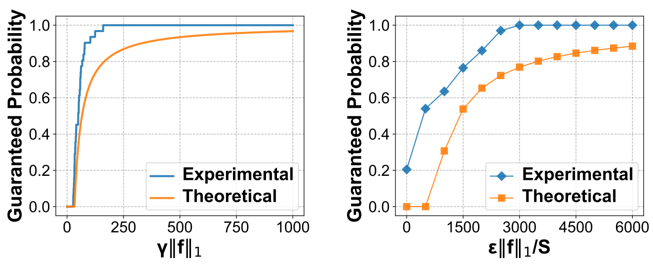
$$\mathbb{E}\left[\hat{f} - f\right] \leqslant \mathbb{E}\left[f_{min}\right] \leqslant \frac{||f||_1}{S}$$

According to Markov inequality, we have

$$\Pr\left(\hat{f} - f \geqslant \epsilon \cdot \frac{||f||_1}{S}\right) \leqslant \frac{S}{\epsilon \cdot ||f||_1} \cdot \mathbb{E}\left[\hat{f} - f\right] \leqslant \frac{1}{\epsilon}$$

$\square$

Experimental analysis (Figure 5): We conduct experiments using CAIDA [2] dataset to validate the theoretical bounds in Theorem 4.5 and Theorem 4.6, where we set $b = 16$. Figure 5(a) shows the experimental and theoretical probability in Theorem 4.5. We can see that the experimental probability is well bounded by the theoretical bound, and when $\gamma||f||_1 > 500$, both the experimental and theoretical probability are larger than 90%. Figure 5(a) shows the experimental and theoretical guaranteed probability in Theorem 4.6. We can see that the experimental guaranteed probability is well bounded by the theoretical probability.



(a) Theorem 4.5. (b) Theorem 4.6.

Figure 5: Accuracy of Bucketized Space-Saving.

3 Details about the Experimental Setup

Platform and setting: We conduct experiments on an 18-core 4.2GHz CPU server (Intel i9-10980XE) with 128GB 3200MHz DDR4 memory and 24.75MB L3 cache. We implement all codes with C++ and build them with g++ 7.5.0 (Ubuntu 7.5.0-6ubuntu2) and -O3 option. The hash functions we use are 32-bit Murmur Hash [3]. By default, the parameters of the comparing algorithms are set according to the recommendation of their authors. We first find the ground-truth batches / top-k items / periodic batches according to predefined parameters, and store them as golden labels in a large hash table.

Datasets:

1) CAIDA dataset: CAIDA [2] is a data stream of IP trace collected in 2018. Each item is identified by its source IP (4 bytes) and destination IP (4 bytes). We use two traces of different sizes: a small-scale 1-minute trace containing about 30M items (default), and a large-scale 1-hour trace containing about 1.5G items.

2) Criteo dataset: Criteo [4] is an advertising click data stream consisting of about 45M ad impressions. Each item is identified by its categorical feature and conversion feedback.

Evaluation Metrics:

1) Recall Rate (RR): The ratio of the number of correctly reported instances to the number of correct instances.

2) Precision Rate (PR): The ratio of the number of correctly reported instances to the number of reported instances.

3) F_1 Score: $\frac{2\times RR\times PR}{RR+PR}$.

4) Average Relative Error (ARE): $\frac{1}{|\Psi|}\sum_{e_i\in\Psi}\left|f_i-\hat{f}_i\right|/f_i$, where f_i is the real frequency of item e_i, $\widehat{f_i}$ is its estimated frequency, and Ψ is the query set.

5) Throughput (Mops): Million operations per second.

References

[1] Michael Flynn. Some computer organizations and their effectiveness. ieee trans comput c-21:948. *Computers, IEEE Transactions on Computers*, C-21:948 – 960, 10 1972.

[2] CAIDA dataset. Available: `http://www.caida.org/home`.

[3] Murmur hashing source codes. `https://github.com/aappleby/smhasher`.

[4] Criteo dataset. Available: `https://ailab.criteo.com/ressources/`.