Learning Based Distributed Tracking

Hao WU
whw4@student.unimelb.edu.au
The University of Melbourne
Melbourne, Australia

Junhao Gan
junhao.gan@unimelb.edu.au
The University of Melbourne
Melbourne, Australia

Rui Zhang
rui.zhang@unimelb.edu.au
The University of Melbourne
Melbourne, Australia

ABSTRACT

Inspired by the great success of machine learning in the past decade, people have been thinking about the possibility of improving the theoretical results by exploring data distribution. In this paper, we revisit a fundamental problem called *Distributed Tracking* (DT) under an assumption that the data follows a certain (known or unknown) distribution, and propose a number *data-dependent* algorithms with improved theoretical bounds. Informally, in the DT problem, there is a coordinator and k players, where the coordinator holds a threshold N and each player has a counter. At each time stamp, at most one counter can be increased by one. The job of the coordinator is to capture the exact moment when the sum of all these k counters reaches N. The goal is to minimise the communication cost. While our first type of algorithms assume the concrete data distribution is *known in advance*, our second type of algorithms can learn the distribution on the fly. Both of the algorithms achieve a communication cost bounded by $O(k \log \log N)$ with high probability, improving the state-of-the-art *data-independent* bound $O(k \log \frac{N}{k})$. We further propose a number of implementation optimisation heuristics to improve both efficiency and robustness of the algorithms. Finally, we conduct extensive experiments on three real datasets and four synthetic datasets. The experimental results show that the communication cost of our algorithms is as least as 20% of that of the state-of-the-art algorithms.

CCS CONCEPTS

• **Mathematics of computing → Probabilistic algorithms**.

KEYWORDS

algorithm, sampling, distributed tracking, machine learning

ACM Reference Format:
Hao WU, Junhao Gan, and Rui Zhang. 2020. Learning Based Distributed Tracking. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403255

1 INTRODUCTION

The great success of machine learning in the past decade has proven the assumption that data in practice follows certain patterns (e.g., either known or unknown distributions). Such an assumption in turn becomes the base of those (machine learning) techniques. Inspired by this, people have been thinking about the possibility to improve theoretical results on traditional problems with machine learning techniques. Successful progresses have been made on a wide range of problems, such as frequency estimation [1, 9], approximate membership [15], combinatorial optimization [3, 12, 17], index structures [14], and etc. The rationale behind these progresses is to explore and exploit the underlying distribution of the input data and to design *data-dependent* algorithms customized for the data distribution. In this paper, we design data-dependent algorithms, improving the state-of-the-art theoretical bounds, for solving the *Distribution Tracking* (DT) problem [7].

The DT problem setting. In the DT problem, there is a coordinator and k players (a.k.a. sites); between each player and the coordinator, there is a two-way communication channel. The coordinator holds a *threshold* N, while the i-th player has a *counter* n_i initialized as 0 for all $i \in \{1, 2, \ldots, k\}$. At each time stamp, there is *at most one* (that means it can be none) player having its counter increased by one, conceptually representing an item arrives at the player. The job of the coordinator is to raise an alarm at the *exact moment* that the N-th item arrives (at some player), equivalently, the moment that the sum of the counters of all the players reaches N, i.e., $\sum_{i=1}^{k} n_i = N$. The efficiency of an algorithm for solving the DT problem is measured by the communication cost, i.e., the total number of *messages* that received and sent by the coordinator, where each message can only carry at most $O(1)$ *words*.

To solve the DT problem, a straightforward algorithm is to instruct each player to send a message to notify the coordinator for every increment on its counter. The communication cost of this algorithm is clearly N (messages). However, such a communication cost is considered expensive, as N is large in practice. Existing work [7] showed that the DT problem actually admits an algorithm (the *CMY* algorithm named after its authors) with a communication cost of $O(k \log \frac{N}{k})$. When N is far larger than k, this algorithm consumes significantly less communication cost than the straightforward algorithm.

The *CMY* algorithm. For the ease of explanation, we introduce a simplified version of the state-of-the-art *CMY* algorithm achieving the same communication bound. The simplified algorithm runs in *rounds*. In each round, if $N < 4k$, run the straightforward algorithm with $O(N) = O(k)$ messages. Otherwise (i.e., $N \geq 4k$), the coordinator sends a *slack* $s = \lfloor \frac{N}{2k} \rfloor$ to each player. Each player sends a message to notify the coordinator, whenever its counter is increased by s since its last communication with the coordinator. If the coordinator receives the k-th message, then it collects all the k counter values n_i from the players and calculate $N' = N - \sum_{i=1}^{k} n_i$. If $N' = 0$, the coordinator raises the alarm. Otherwise, start a new

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403255

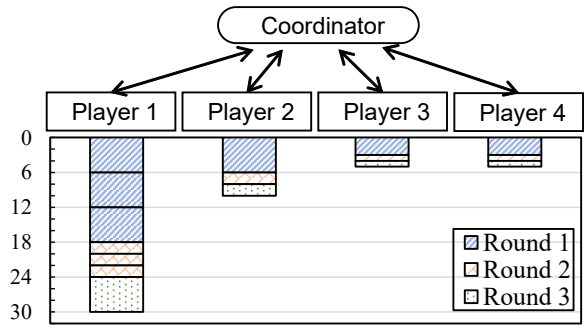


Figure 1: A running example of the simplified *CMY* algorithm with $N = 50$ and $k = 4$, where the counter increments in different rounds are highlighted with different colors and textures. The slack values in the first two rounds are $s = 6$ and $s = 2$, respectively; and in the third round, the algorithm switches to the straightforward algorithm.

round to solve a *new* DT problem instance with $N = N'$ from scratch. As it is easy to verified that in each round, the coordinator sends and receives $O(k)$ messages; and after each round, N is decreased by a constant factor. Hence, there can be at most $O(\log \frac{N}{k})$ (with respect to the original N) rounds; the total communication cost is bounded by $O(k \log \frac{N}{k})$ messages.

A running example. Figure 1 shows a running example of the DT algorithm on an instance with $N = 50$ and $k = 4$, In the first round, since $N > 4k = 16$, the coordinator sends a slack $s = \lfloor \frac{N}{2k} \rfloor = \lfloor \frac{50}{2*4} \rfloor = 6$ to each player. At the end of this round, Player 1 has received 18 counter increments and thus in total 3 messages have been sent from Player 1 to the coordinator, which were sent for every $s = 6$ increments. Likewise, Player 2 has sent a message as its counter is increased by $s = 6$, while both of the counters of Players 3 and 4 are just increased by 3: no messages were sent from them. The coordinator collects all the counters as soon as it receives the k-th (i.e., the fourth) messages and calculate $N' = 50 - (6 * 3 + 6 + 3 + 3) = 20$. Next, the coordinator starts a new round with a new instance with $N = 20$ and $k = 4$ from scratch, where $s = \lfloor \frac{20}{2*4} \rfloor = 2$. As shown in the figure, at the end of this round, the counters of the four players have been increased by 6, 2, 1 and 1, respectively; and $N' = 10$. A new round with $N = 10$ and $k = 4$ is thus started, in which the algorithm switches to the straightforward algorithm (as $N < 4k$): a message is sent for each counter increment. As for the communication cost, in each of the first two rounds, the coordinator sends 4 messages for sending the slack s to the four players, receives 4 messages, sends 4 messages for requesting the counter values, and finally receives 4 messages for the counter values. Therefore, the communication cost in each of these two rounds is 16. Plus the $4 + 10$ messages in the last round, the total communication cost is thus $16 * 2 + 14 = 46$.

Exploiting the counter increment distribution. In order to provide a worst-case communication bound, the *CMY* algorithm has to be pessimistic and conservative: it makes no assumption on the data distribution; and only identical slacks s can be sent to all the players in a round. Such pessimism and conservation, unfortunately, prevents the *CMY* algorithm from reducing the communication by exploiting the data distribution. In particular, the data distribution

Notation	Description
N	the threshold to monitor
k	the number of players
$[k]$	the set of integers from 1 to k
n_i	the counter of the i-th player
s_i	the slack of the i-th player
μ_i	the probability that a new item arrives at player i
$\bar{\mu}_i$	the estimation of μ_i
$T(N)$	the communication cost with threshold N

Table 1: Frequently used notations

we mean here is the *Multinomial Distribution of the counter increments*, more specifically, the *probability distribution* of a counter increment happening in the players. For the example shown in Figure 1, the probability distribution of the counter increments is $(0.6, 0.2, 0.1, 0.1)$: for each counter increment, it has probability of 0.6 happening in Player 1, 0.2 in Player 2, 0.1 in Player 3, and 0.1 in Player 4.

The knowledge on the probability distribution (i.e., the Multinomial Distribution) of the counter increments indeed can be leveraged to significantly reduce the communication cost. As an extreme case in our earlier example in Figure 1, suppose that one knows the *final* counter value c_i of the i-th player for all $i \in \{1, 2, \ldots, k\}$ at the moment when their sum reaches N, namely, $c_1 = 30$, $c_2 = 10$, $c_3 = c_4 = 5$. A better solution is to instruct the coordinator to send a *customized* slack $s_i = c_i$ to the i-th player, and to raise the alarm when it receives the fourth messages from the players. Clearly, the communication cost of this solution is only 8 messages, *five times* less than the cost of the *CMY* algorithm. While knowing all the final counter values c_i a prior is, of course, too good to be true, this observation sheds a light on the possibility of designing improved *data-dependent* algorithms with the knowledge of the counter increment distribution.

Motivated by the above, in this paper, we consider the DT problem under the assumption below:

Assumption 1. *The counter increments follow a certain Multinomial Distribution which can be unknown. More specifically, each counter increment occurs in the i-th player with probability μ_i for all $i \in \{1, 2, \ldots, k\}$, where $\sum_{i=1}^{k} \mu_i = 1$.*

Our contributions. We make the following contributions:

- First, for the case that the *concrete* Multinomial Distribution (i.e., the concrete values of all μ_i's) is *known* a prior, we show a data-dependent algorithm, called *StcSlk-KwnDst*, for solving the DT problem. The communication cost of the *StcSlk-KwnDst* algorithm is bounded by $O(k \log \log N)$ with high probability, improving the state-of-the-art $O(k \log \frac{N}{k})$ bound. We further propose the *DynSlk-KwnDst* algorithm which improves the practical performance of *StcSlk-KwnDst*, while retaining exactly the same communication bound.
- Second, for the case that the distribution is *unknown*, we propose two learning based algorithms, called *StcSlk-LrnDst* and *DynSlk-LrnDst*, corresponding to the two algorithms in the first case. Both of these two algorithms can learn the data distribution on the fly, meanwhile achieving exactly the same theoretical bounds as their counterpart algorithms.

- Moreover, we design an effective heuristics to optimize our algorithm implementations.
- Finally, we conduct extensive experiments on both three real datasets and four synthetic datasets of different data distributions. The experimental results show that our proposed algorithms outperform the state-of-the-art algorithms by consuming up to five times (i.e., 5x) less communication cost.

2 RELATED WORK

The *distributed tracking* (DT) problem has been well studied in terms of both upper bounds and lower bounds, since it was first proposed.

Prior to the *CMY* algorithm, an uniform slack (called *UniSlk*) algorithm was proposed by Cormode et al. [5]. The communication cost of *UniSlk* is bounded by $O(k^2 \log \frac{N}{k})$. The design of *UniSlk* is based on the observation that for N items arriving at k players, there must be at least one of player received N/k items. The *UniSlk* algorithm runs in rounds. At the at the beginning of the first round, the coordinator broadcasts a slack N/k to each players. The player notifies the coordinator when its counter exceeds N/k. Upon receiving one notification, the coordinator informs the rest of players to report the values of their counters. This ends the first round. The number of items arrived, namely, $\sum_{i=1}^{k} n_i$ is at least N/k and at most $\lceil N/k \rceil + (k-1)\lfloor N/k \rfloor \leq N$. The coordinator updates the threshold $N \leftarrow N - \sum_{i=1}^{k} n_i$. If $N = 0$, it raises an alarm, otherwise it starts a new round. Each round incurs $O(k)$ communications and decreases the threshold N by a factor of at least $(1 - 1/k) \leq \exp(-1/k)$. Therefore, the number of rounds is at most $O(k \log \frac{N}{k})$ and the communication cost is thus $O(k^2 \log \frac{N}{k})$.

Later on, the *CMY* algorithm was proposed in [7]. As introduced earlier, the *CMY* algorithm consumes $O(k \log \frac{N}{k})$ communication cost. Although we are focusing on the number of messages in this paper, in [7], Cormode et al. showed a $\Omega(k \log \frac{N}{k})$-bit communication lower bound for the DT problem. Moreover, the *CMY* algorithm indeed admits a bit-version implementation with communication cost of $O(k \log N)$ bits.

More works have been done for the variants of the DT problem. Randomized algorithm was also proposed for approximate count tracking [7]. Instead of reporting exactly the N-th item, the coordinator is allowed to raise an alarm on the arrival of any item between $[(1-\epsilon)N, N]$, where $\epsilon \in (0, 1)$ is a specified parameter. The problem is easier than the exact count tracking problem due to the relaxation and the proposed algorithm has cost $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ bits [7], where δ is the failure probability of the algorithm. [10, 11] studied the continuous count tracking problem, in which the coordinator is required to report an estimation $\hat{n}$ of $n = \sum_{i \in [k]} n_i$ at any time stamp, such that $\hat{n} \in (1 \pm \epsilon)n$. Their algorithms have communication costs of $O(\frac{k}{\epsilon} \log \frac{\epsilon N}{k})$ [11] and $O(\frac{\sqrt{k}}{\epsilon} \log N)$ messages [10], respectively. The work of [7] also considered threshold tracking of F_p moment, where DT can be considered as a special case of $p = 1$.

Data pattern has been studies and exploited for different distributed monitoring queries ([6, 8]), but none of them targets the fundamental threshold count tracking problem. [6] introduced the Update-Rate Model for distributed tracking of approximate quantiles, which assumes that items arrives at the i^{th} player at a local rate specified to i. Note that this is captured by the multinomial distribution model if we normalize the rates by the summation of the players' rates (and with proper scaling of time). Later [8] extends the idea for geometric monitoring to reduce the communication cost.

Besides, interestingly, the techniques for solving the DT problem has also been applied to solve some seemly "remote" problem in the *single machine* setting [16].

Algorithm 2.1 The ℓ-Notifications-to-End Framework

Input: a threshold N, the number of players k, and possibly a failure probability δ if applicable

1: Set $n_i \leftarrow 0$ for all $\forall i \in [k]$
2: If $N \leq 4k$, run the straightforward algorithm until it terminates.
3: If $N \leq \beta \cdot k \ln \frac{k}{\delta}$, run the *CMY* algorithm until it terminates, where β is an algorithm-specified constant.
4: The coordinator **send a slack** s_i to player i for $\forall i \in [k]$.
5: Each Player i notifies the coordinator, **when** n_i **meets certain condition with respect to** s_i.
6: The coordinator collects all the n_i's from the players, **when it receives the** ℓ**-th notification**.
7: Set $N \leftarrow N - \sum_{i=1}^{k} n_i$
8: If $N = 0$, the coordinator raises an alarm and terminate.
9: Otherwise, go to Step 1 and start a new round.

3 A UNIFIED ALGORITHM FRAMEWORK

Before we get into the details of our algorithms, in this section, we first propose a *unified algorithm framework*, called *ℓ-Notifications-to-End*, where ℓ is a *characteristic parameter* of an algorithm and not an input parameter. As we will see shortly, all our algorithms, the *CMY* algorithm as well as the *UniSlk* algorithm are all under this framework. Algorithm 2.1 shows the pseudo code of the framework.

Algorithm characteristics. Essentially, algorithms under the *ℓ-Notifications-to-End* framework only differ in the following three *characteristics*:

- **Characteristic 1:** the value of slack s_i for $\forall i \in [k]$ (Line 4);
- **Characteristic 2:** the condition for a player to notify the coordinator (Line 5);
- **Characteristic 3:** the number ℓ of notifications received to end a round (Line 6).

To see this, consider the *CMY* algorithm, where: (i) $s_i = \lfloor \frac{N}{2k} \rfloor$; (ii) a player notifies the coordinator when n_i is increased by s_i; and (iii) $\ell = k$. Thus, the *CMY* algorithm is an k-Notifications-to-End algorithm. On the other hand, the *UniSlk* algorithm is, in fact, a 1-Notification-to-End algorithm with $s_i = N/k$, where a player notifies the coordinator when $n_i \geq s_i$, and with Line 3 being never executed. As we will see in the next two sections, all our algorithms will be focusing on designing the above three characteristics.

Communication cost expression. Observe that the communication cost for executing Line 2 and Line 3 in Algorithm 2.1 are bounded by $O(k)$ and $O(k \log \log \frac{k}{\delta})$, respectively. As these two cases are easy to check and solve, it suffices to focus on the case $N = \omega(k \ln \frac{k}{\delta})$. In this case, it can be verified that an ℓ-Notification-to-End algorithm consumes $O(k + \ell)$ communication cost per round.

Therefore, the overall communication cost in this case is bounded by $O((k+\ell)\cdot R + k\log\log\frac{k}{\delta})$, where R is the total number of rounds that have been executed before entering into Line 2.

4 TRACKING WITH KNOWN DISTRIBUTION

In this section, we consider the case that the concrete Multinomial Distribution of the counter increments (a.k.a. the item arrivals) is known. That is, the concrete values of μ_i for $i \in [k]$ are given. This case allows us to just focus on algorithmic design without worrying too much about the learning of the distribution.

4.1 Tracking with Static Slacks

The challenges. We note that even the concrete values of μ_i's are known in advance, the problem is still challenging.

As the concrete values of all μ_i's are known, it is natural to think about modifying the *CMY* algorithm such that the slack s_i is set to the expected number of items that Player i will receive when the N-th item arrives, namely, $s_i = \mu_i N$ for $\forall i \in [k]$. We denote this new k-Notifications-to-End algorithm by $\mathcal{A}$.

Unfortunately, in general, it is unlikely that every player will receive *exactly* $\mu_i N$ items, i.e., $n_i = \mu_i N$, when the N-th item arrives; the actual n_i could be more or less than $\mu_i N$. As a result, $\mathcal{A}$ may fail to capture the moment of the N-th item's arrival, when the coordinator receives the k-th notification.

To remedy this, one possible way is to further modify $\mathcal{A}$ into a 1-Notifications-to-End algorithm by setting ℓ to 1, where the coordinator in $\mathcal{A}$ collects all the n_i's as soon as it receives a notification from the player. This guarantees that $\sum_{i=1}^{k} n_i < N$ and the coordinator would not miss the N-th item. However, $\mathcal{A}$ could be less efficient than *CMY*. This is because any small deviation from $\mu_i N$ would easily make $\mathcal{A}$ end the round too early, resulting in an increase on the number of rounds R. As an illustration, suppose that there are 10 players and the first three players have very low probabilities of receiving items. In particular, $\mu_1 = \mu_2 = \mu_3 = \frac{1}{N}$. In this case, if any item arrives at any of the first three players, a notification is sent to the coordinator and the round ends. With probability at least $1 - 1/e$, this round captures only $N/3$ items: for a given item, the probability that it arrives at a player other than the first three is $(1 - 3/N)$; therefore, with probability $(1 - 3/N)^{N/3} \le 1/e$, none of the first $N/3$ items arrive at any of the first three player. In comparison, *CMY* captures at least $N/2$ in one round.

To capture more items in one round, the slack should contain a leeway for the player's counter to bear deviations from its expectation, i.e., s_i should be greater than $\mu_i N$ for $i \in [k]$. The player should receive a few more items than its expectation to defer the notification. However, this conflicts with our goal of capturing the N-th item – if we set $s_i > \mu_i N$ for $i \in [k]$, then $\sum_{i\in[k]} s_i > N$. The coordinator could miss the arrival of the N-th item. The following constraint for ensuring the correctness

$$\sum_{i=1}^{k} s_i \le N \tag{1}$$

implies that tracking N items in just one round seems too ambitious. We relax our goal and, instead, aim to track t items for some $t < N$ and set $s_i > \mu_i t$, while ensuring the correctness, i.e., $\sum_{i=1}^{k} s_i \le N$. We prove that there exists s_i's, such that when the first t items arrives, with high probability that none of the player has received more than s_i items. It implies that no notification would have been sent to the coordinator. In other words, when the first notification is sent, more than t items have arrived in this round. Clearly, the larger t is, the less number of rounds we need.

The *StcSlk-KwnDst* Algorithm. Motivated by the above observation, we propose our first data-dependent algorithm, *StcSlk-KwnDst*, which is an 1-Notifications-to-End algorithm. characterized by the followings:

- **Characteristic 1**:

$$s_i = \mu_i t + \sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta}, \forall i \in [k]; \tag{2}$$

- **Characteristic 2**: Player i notifies the coordinator when $n_i = s_i$;
- **Characteristic 3**: $\ell = 1$, that is, the coordinator ends a round when it receives the first notification from the players.

Substituting the above implementations of the three characteristics to Lines 4, 5 and 6, respectively, in Algorithm 2.1 gives the pseudo code of the *StcSlk-KwnDst* algorithm. Furthermore, we have the following key theorem:

Theorem 4.1. *With probability $1-\delta'$, the StcSlk-KwnDst algorithm:*

- *runs at most $O(\log\log\frac{N}{k\log\frac{k}{\delta}} + \log\log\frac{k}{\delta})$ rounds;*
- *has total communication cost $O(k\log\log\frac{N}{k\log\frac{k}{\delta}} + k\log\log\frac{k}{\delta})$,*

where $\delta' = \delta\cdot O(\log\log N)$.

Answering the following three questions is the key to proving Theorem 4.1:

- Why does Expression (2) ensure that *StcSlk-KwnDst* can capture at least t items at one round with probability $1-\delta$?
- How large can t be?
- How many rounds does *StcSlk-KwnDst* need?

In what's follows, we address these questions one by one.

Setting the Slack. Expression (2) is actually derived from the following concentration inequality:

Fact 1. *(Bernstein inequality [4]) Let $Y_1, \ldots, Y_t$ be independent, random variables. Let $Y = \sum_{j=1}^{t} Y_j$, and $M > 0$ be such that $Y_j \le E[Y_j] + M$ for all $j \in [t]$. For any $\lambda \ge 0$,*

$$\mathbf{Pr}[Y \ge E[Y] + \lambda] \le \exp\left(-\frac{\lambda^2}{2(Var[Y] + M\lambda/3)}\right). \tag{3}$$

Consider a fixed Player i and the first t items arrived in the current round. Denote by X_j the Bernoulli random variable that $X_j = 1$ if the j-th item arrives at player i, and $X_j = 0$ otherwise. Then $E[X_j] = \mu_i$, $Var[X_j] = \mu_i(1-\mu_i)$ and $X_j \le E[X_j] + 1$ for $j \in [t]$. By dentition, the counter $n_i = \sum_{j\in[t]} X_j$. Fact 1 gives the following results:

Lemma 4.2. *Given $t > 0$ and a failure probability $\delta > 0$, define*

$$UB_i = \mu_i t + \sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta}. \tag{4}$$

Then the probability $Pr[n_i \ge UB_i] \le \frac{\delta}{k}$. Likewise, for

$$LB_i = \mu_i t - \sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} - \frac{2}{3}\ln\frac{k}{\delta}, \tag{5}$$

we have $Pr[n_i \le LB_i] \le \frac{\delta}{k}$.

The proof of Lemma 4.2 can be found in the Appendix A. By Lemma 4.2, for a fixed i, by setting the slack $s_i = UB_i$, when the first t items arrive, the event $n_i \ge s_i$ happens with probability at most δ/k. By union bound, the probability that $n_i \ge s_i$ for any $i \in [k]$ is at most δ. Therefore, the following corollary holds.

Corollary 4.3. *With probability* $\ge 1-\delta$*, the coordinator receives no notification from the players for the first* t *items in a round.*

Setting t**.** Define function $f(t) \doteq \sum_{i\in[k]} s_i$, with $s_i = UB_i$ (Expression (2)). It is easy to verify that $f(t)$ is monotonically increasing with t. Clearly, the larger t the more items can be tracked in a round, while t should also satisfy: $f(t) \le N$ to ensure the correctness of the algorithm. As a result, it is desired to maximise t subject to $f(t) \le N$. This will possibly reduce the total number of rounds in the *StcSlk-KwnDst* algorithm and hence, reduce the total communication cost. However, computing the optimal t precisely may not be an easy task. Nonetheless, as we show shortly, $t = N - (\sqrt{2}+2/3)\sqrt{kN\ln\frac{k}{\delta}}$ is already good enough for our purpose. In particular, we have the following lemma whose proof is in given in Appendix A.

Lemma 4.4. $f(t) \le N$ *for* $t = N - (\sqrt{2}+2/3)\sqrt{kN\ln\frac{k}{\delta}}$.

Bounding the communication cost. Consider an implementation of the *StcSlk-KwnDst* algorithm with its three characteristics plug in to the unified framework (Algorithm 2.1), where we further explicitly set $\beta = 2\cdot(\sqrt{2}+2/3)^2$. According to this implementation, we know that when $N \ge \beta \cdot k\ln\frac{k}{\delta}$, at the end of each round, the value of N can be decreased to at most $(\sqrt{2}+2/3)\sqrt{kN\ln\frac{k}{\delta}}$ with probability $1-\delta$ (by Corollary 4.3 and Lemma 4.4). Furthermore, when N is found smaller than $\beta\cdot k\ln\frac{k}{\delta}$ at the start of a round, the algorithm switches to the *CMY* algorithm (according to Line 3 in Algorithm 2.1). Therefore, this gives the following recursion, where $T(N)$ denotes the communication cost of the algorithm with respect to N.

$$T(N) = \begin{cases} T\left((\sqrt{2}+2/3)\sqrt{kN\ln\frac{k}{\delta}}\right) + O(k), & N \ge 2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta} \\ T(N/2) + O(k), & 4k \le N < 2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta} \\ O(k), & N < 4k \end{cases}$$

Solving the recursion gives the last lemma we need for Theorem 4.1.

Lemma 4.5. $T(N) = O(k\log\log\frac{N}{k\ln\frac{k}{\delta}} + k\log\log\frac{k}{\delta})$.

Proving Theorem 4.1. Putting Corollary 4.3, Lemma 4.4 and 4.5 together, it thus completes the proof for Theorem 4.1.

4.2 Tracking with Dynamic Slacks

In the previous subsection, we know that the *StcSlk-KwnDst* algorithm assigns to Player i a slack $s_i = UB_i > \mu_i t$. In expectation, $\mu_i t$ items arrives at the i-th player. Intuitively, $UB_i - \mu_i t$ is the *tolerance* that how much the counter n_i is allowed to deviate from its expectation $\mu_i t$, when t items arrives. As soon as n_i reaches UB_i, Player i is not allowed to further receive any new items (to ensure the correctness). Hence, at this moment, the coordinator collects the precise counters and ends the current round.

While the above strategy has been shown to be effective in the previous subsection, setting $s_i = UB_i$ is a *static* slack assignment strategy. In the sense that, the tolerance for deviations, i.e., $UB_i - \mu_i t$, is *pre-determined* and fixed for each Player i. When the coordinator ends a round, except for the player sending the notification, all other players actually have not fully used up their deviation tolerance. An immediate question comes up: Can we further improve the utilization of those *non-fully-used* deviation tolerances, before ending a round?

Motivated by the question, we design a new slack assignment strategy to *dynamically* adjust the deviation tolerance for the players. The basic idea is as follows. First, observe that the sum of all the deviation tolerance of the players is computed as

$$\sum_{i=1}^{k}(UB_i - \mu_i t) \le N - t.$$

Instead of pre-assigning a static tolerance to each player, we adopt the strategy of the *CMY* algorithm. More specifically, the coordinator sends a base value $b_i = \mu_i t$, and a deviation tolerance $s_i = \lfloor\frac{N-t}{2k}\rfloor$ to Player i, for $\forall i \in [k]$. A player sends a notification to the coordinator for every counter increment s_i *only when* $n_i \ge b_i$. The coordinator collects the counters and ends the round when it receives the k-th notification. The resulted algorithm is called *DynSlk-KwnDst*; since a round is ended when the coordinator receives k notifications, the *DynSlk-KwnDst* algorithm is a k-Notifications-to-End algorithm, according to our unified framework.

In particular, *DynSlk-KwnDst* implements the three characteristics as follows:

- **Characteristic 1:** the slack is a pair (b_i, s_i) for $\forall i \in [k]$, where $b_i = \mu_i t$ and $s_i = \lfloor\frac{N-t}{2k}\rfloor$;
- **Characteristic 2:** Player i sends a notification to the coordinator for every counter increment s_i *only when* $n_i \ge b_i$;
- **Characteristic 3:** $\ell = k$.

Substituting the above implementations to the algorithm framework (Algorithm 2.1), gives the pseudo code of the *DynSlk-KwnDst* algorithm.

As strategy of *CMY* can guarantee that the coordinator will not miss the arrival of the $(N-t)$-th item, it thus guarantees that no more than N items can be received in a round. Therefore, the correctness of the *DynSlk-KwnDst* algorithm follows. Furthermore, the theorem below shows that with probability at least $1-\delta$, at the end of a round, at least t items arrive to the players.

Theorem 4.6. *With probability at least* $1-\delta$*, less than* k *notifications will be sent from the players for the first* t *items in a round.*

By Theorem 4.6, Lemmas 4.4 and 4.5, we have:

Theorem 4.7. *The DynSlk-KwnDst algorithm achieves exactly the same bounds of StcSlk-KwnDst as stated in Theorem 4.1.*

5 LEARNING BASED TRACKING

In this section, we consider the case that the underlying counter increment distribution is *unknown*. The basic idea is to learn the

distribution on the fly. To learn the unknown distribution, we run the *CMY* algorithm for the *first round*. This allows us to receive at least $N/2$ items with only $O(k)$ communication. At the end of this first round, we estimate μ_i by $\bar{\mu}_i = \frac{n_i}{\sum_{i\in[k]} n_i}$ for $\forall i \in [k]$. These $\bar{\mu}_i$'s are used in the subsequent rounds to determine the slacks. As $\bar{\mu}_i$'s are just estimations, they may introduce additional errors. Furthermore, since $\bar{\mu}_i$ could be an underestimation of μ_i, the upper bound UB_i computed by simply replacing μ_i with $\bar{\mu}_i$ in Expression (4) may be no longer a proper upper bound for n_i when the first t items arrive. Therefore, modifications to the previous algorithms are required.

The modifications consist of three steps. First we construct some $\hat{\mu}_i$ based on $\bar{\mu}_i$ such that it is guaranteed that $\hat{\mu}_i \geq \mu_i$. Next, we show how to construct UB_i with $\hat{\mu}_i$. Last, t needs to be change to ensure $\sum_{i\in[k]} UB_i \leq N$.

Upper bound on μ_i. The concentration inequality below is needed.

Fact 2. *(Empirical Bernstein Bound) [2] Let $Y_1, ..., Y_w$ be independent, random variables with mean μ. Let $\bar{Y} = \frac{1}{w}\sum_{j\in[w]} Y_j$, and $M > 0$ be such that $|Y_j| \leq M$ for all $j \in [w]$. With probability at most δ, it holds that*

$$|\bar{Y} - \mu| \geq \sqrt{\frac{2\bar{\sigma}^2 \ln\frac{3}{\delta}}{w}} + \frac{3M\ln\frac{3}{\delta}}{w}$$

where $\bar{\sigma}^2$ is the empirical variance of Y_j's: $\bar{\sigma}^2 = 1/w\sum_{j\in[w]}(Y_j - \bar{Y})^2$.

Consider a fixed $i \in [k]$ and the number of items n_i that arrive at Player i, for the w items tracked in the first round. Denote Y_j the Bernoulli random variable such that $Y_j = 1$ if the j-th item arrives at Player i, and $Y_j = 0$ otherwise. Then $Y_j \leq 1$ for $j \in [w]$. Denote $\bar{\mu}_i = \bar{Y} = 1/w\sum_{j\in[w]} Y_j$ as the empirical mean. As Y_j's are Bernoulli random variables, the empirical variance is $\bar{\sigma}^2 = \bar{\mu}_i(1-\bar{\mu}_i)$. According to Fact 2, a upper bound $\hat{\mu}_i$ on μ_i can be obtained:

$$\hat{\mu}_i \doteq \bar{\mu}_i + \sqrt{\frac{2(\bar{\mu}_i - (\bar{\mu}_i)^2)\ln\frac{3}{\delta}}{w}} + \frac{3\ln\frac{3}{\delta}}{w} \tag{6}$$

Modification on UB_i. The slack $s_i = UB_i$ is modified as below,

$$UB_i = \hat{\mu}_i t + \sqrt{2t\hat{\mu}_i \ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta} \tag{7}$$

Modification on t. We need to change the value of t. Define $\hat{\Sigma} = \sum_{i\in[k]}\hat{\mu}_i$. Then we set

$$t = N/\hat{\Sigma} - (\sqrt{2} + \frac{2}{3})\sqrt{k(N/\hat{\Sigma})\ln\frac{k}{\delta}} \tag{8}$$

Substituting the modified $\hat{u}_i, UB_i$ and t to the known-distribution counterparts, we can have the learning based versions for tracking with static slack (called *StcSlk-LrnDst*) and with dynamic slacks (called *DynSlk-LrnDst*) respectively. Some extra care is required to set the constant $\beta = 2\cdot(2\sqrt{2} + 2/3 + 3)^2$ in the condition of when to switch to the *CMY* algorithm (at Line 3 in Algorithm 2.1) in the framework. Moreover, we show our final theorem whose proof can be found in Appendix A.

Theorem 5.1. *With probability at least $1-\delta$, the communication cost of the StcSlk-LrnDst algorithm (respectively, the DynSlk-LrnDst algorithm) is bounded by $O(k\log\log\frac{N}{k\ln\frac{k}{\delta}} + k\log\log\frac{k}{\delta})$).*

Name	***Threshold*** (N)	***#Players*** (k)
WorldCup Day 30/60/90	3.4 M/48 M/1.8 M	8/29/2
Dartmouth 1st Oct/Nov/Dec	184 K/254 K/297 K	336/348/319
Uber Feb/Apr/June	2.2 M/2.2 M/2.8 M	262/262/262
Uniform	$2^{10} - 2^{24}$	$2 - 256$
Gaussian	$2^{10} - 2^{24}$	$2 - 256$
Zipfian	$2^{10} - 2^{24}$	$2 - 256$
Exponential	$2^{10} - 2^{24}$	$2 - 256$

Table 2: Dataset Characteristics ($M = 10^6$ and $K = 10^3$)

6 EXPERIMENTAL EVALUATION

This section evaluates the proposed algorithms against the state-of-art competitors on a machine running on Ubuntu 18.04 with Intel(R) Core(TM) i7-8665U CPU @1.90 GHz and 16GB memory. We compare our four algorithms: *StcSlk-KwnDst*, *StcSlk-LrnDst*, *DynSlk-KwnDst*, *DynSlk-LrnDst* with *CMY* and *UniSlk*. All the algorithms are implemented by C++ and compiled with gcc 7.4.0. A backup heuristic is implemented such that, when the empirical distribution is not stable, our algorithms can detect this case and switch to *CMY*. The details can be found in Appendix B. We conduct experiments on three real datasets and four synthetic datasets; the meta data are summarised in Table 2.

Real Datasets. Below are the three real datasets we used. In each of the real datasets, we assign a unique id to each player (randomly and uniquely) in $[0, k)$, where k is the number of players in the corresponding dataset.

World Cup HTTP request data[1]. The dataset consists of 92 days' requests to the 1998 World Cup website servers between April 30, 1998 and July 26, 1998. We use the requests of three representative days, namely the 30-th, the 60-th and the 90-th day, as the datasets in our experiment. On each selected day, the players are the servers that have received at least one request and the threshold to track is the number of requests on that day. Each item is a request that arrives at some server, in ascending order according to its time stamp.

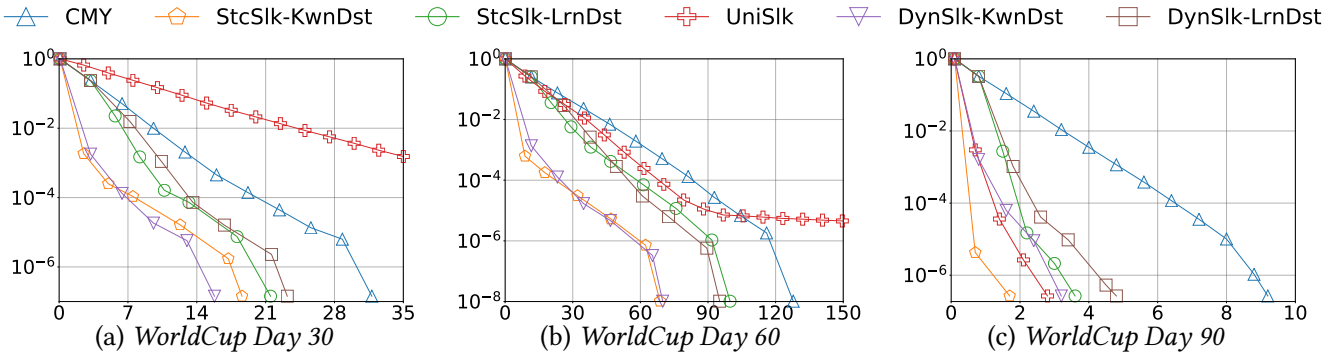
Dartmouth Campus Snmp Traceset [13]. The dataset contains polling records of access points (AP) at Dartmouth College by Simple Network Management Protocol (SNMP) in Fall 2001. We use the records in three days, i.e., 1st Oct, 1st Nov, and 1st Dec. Each AP is reviewed as a player and the number of polling records on the selected day as the threshold. Each polling record is an item arriving in ascending order by its time stamp.

Uber Pickups[2]. This dataset contains data on over the Uber pickups in New York City from January to June, 2015. Each record has a pickup time, a pickup location id and some other information. We take the pickups in three months, i.e., February, April and June as datasets. We consider the locations as players and the threshold to report is the number of pickups within the corresponding month. Each pickup record is treated as an item arriving in ascending order by the pickup time.

Synthetic Datasets. The data in the synthetic datasets are generated with various distributions. Specifically, the distributions are:

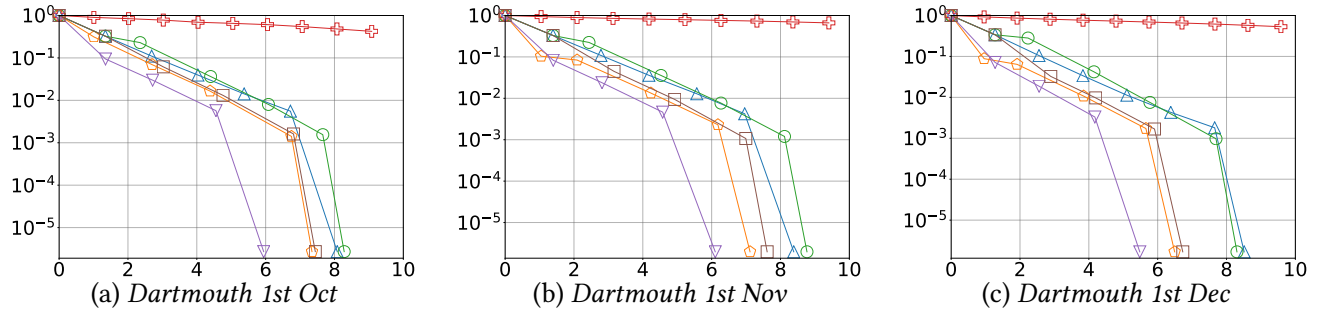
[1] ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html

[2] https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city#uber-raw-data-janjune-15.csv



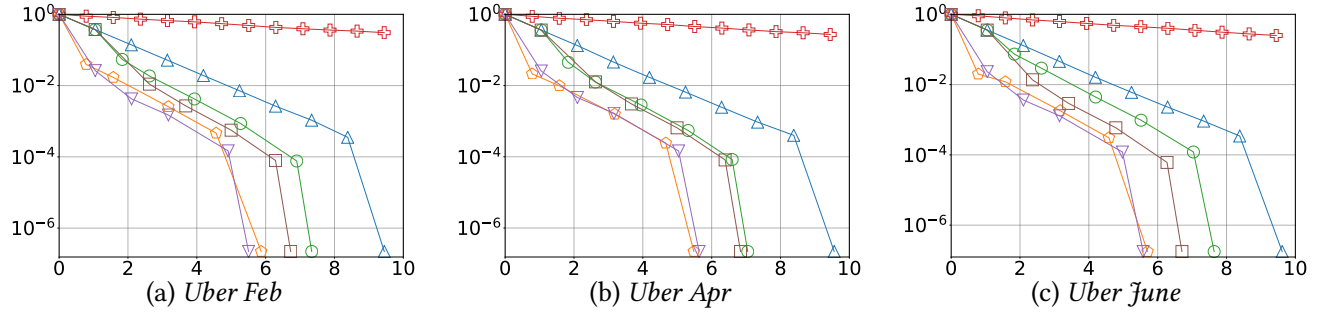
(a) *WorldCup Day 30* (b) *WorldCup Day 60* (c) *WorldCup Day 90*

Figure 2: The Untracked Item Percentage v.s. Communication Cost (10x) on WorldCup datasets



(a) *Dartmouth 1st Oct* (b) *Dartmouth 1st Nov* (c) *Dartmouth 1st Dec*

Figure 3: The Untracked Item Percentage v.s. Communication Cost (10^3x) on Dartmouth datasets



(a) *Uber Feb* (b) *Uber Apr* (c) *Uber June*

Figure 4: The Untracked Item Percentage v.s. Communication Cost (10^3x) on Uber datasets

Uniform, Gaussian, Zipfian and Exponential. The number k of the players varies from 2 to 256 (with multiplicative factor 2) and the threshold N ranges from 2^{10} to 2^{24}. Moreover, for each generated value x, if x is not $[0, k)$, then we just simply discard it; on the other hand, if x is not an integer, we take its floor, i.e., $\lfloor x \rfloor$ to round it into a player id. In particular, the parameters of each of distribution are as follows:

- **Uniform**: the id of the player for each item is generated uniformly at random in $[0, k)$.
- **Gaussian**: we set the mean to $k/2$ and standard variance to $k/6$;
- **Zipfian**: each item arrives at player i with probability *proportional*[3] to $\frac{1}{\sqrt{i+1}}$;
- **Exponential**: each item has probability proportional to $\exp(-i)$ arriving at the i-th player.

[3]By proportional we mean here: the probability is normalized subject to the condition that the sum of the probabilities corresponding to the players is 1.

Parameter Settings. As the combination of all the parameters and the distributions is considerably large, we set the default values of N to 2^{20} and of k to 16. When varying a parameter, the other is set to its default value. Furthermore, the *StcSlk-KwnDst* and *DynSlk-KwnDst* require to know the concrete distribution of the datasets as an input, which may be unavailable for real datasets. Thus, we use *frequencies* of each players as its multinormial distribution. Finally, we set the failure probability to 1%, which suffices for most of the applications in practice.

6.1 Results on the Real Datasets

Figure 2-4 illustrate the results of the algorithms on three real datasets. All plots in the figures refer to the percentage of untracked items as a function of the number of used communications. Each plot represents a round. Different algorithms require different number of communications for a round. Some plots are truncated for *UniSlk* because it takes much more communications that others.

The figures show several results. First all algorithms, *StcSlk-KwnDst* and *DynSlk-KwnDst* perform the best, with *DynSlk-KwnDst* slightly better in most cases. This is as expected because they know the frequency information and have more knowledge than the other algorithms. It confirms the effectiveness of our strategy. Compared to *CMY*, the number of communications reduces by 25% (Figure 3 (b)) to 75% (Figure 2 (c)).

Second, the performances of *StcSlk-LrnDst* and *DynSlk-LrnDst* are inferior to *StcSlk-KwnDst*/*DynSlk-KwnDst* but better than *CMY* in general. Compared to the *StcSlk-KwnDst*/*StcSlk-KwnDst*, they don't know the item arrival frequency at each player in the datasets and therefore have less information. They run *CMY* for the first round to learn an approximate distribution of the dataset. Therefore, their performance in the first round is exactly the same as *CMY* in the first round. The learned distribution helps in tracking the incoming items in most cases. Compared to *CMY*, we observe much sharper decreases in the curves from the second round.

Third, both *DynSlk-KwnDst* and *DynSlk-LrnDst* exhibit better performance than *StcSlk-KwnDst* and *StcSlk-LrnDst*. As the real datasets do not necessarily have perfect distribution, incorporating *CMY* to handle counters' deviation from their expectation values in an aggregate and dynamic manner is more stable than using merely predetermined and static slacks. The only exception is the dataset *WorldCup Day 90* (Figure 2 (c)), in which there are only two players and the distribution is rather skew and stable. Therefore, *StcSlk-KwnDst* and *StcSlk-LrnDst* win with static slacks.

The figures also show the effectiveness of our backup mechanism (as described in Appendix B) when the distribution of real dataset is not stable. In *Dartmouth* datasets, the distribution is rather unstable – *StcSlk-KwnDst* and *StcSlk-LrnDst* fail in the second round and track much fewer percentage of items in the second round than the first one. Detecting the degeneracy in efficiency, they switch to *CMY* in the third round. *StcSlk-LrnDst* lose only by a marginal amount to *CMY* even in this case.

Finally, the *UniSlk* algorithm gives the worst performance as its time complexity grows quadratically with respect to k, the number of players. Further, it is sensitive to skew distortions. In *WorldCup Day 60* (Figure 2 (b)), *UniSlk* exhibits frequent termination of rounds as the tailing items comes in a very unbalanced manner. All other algorithms have switched to *CMY* and handle the tailing items smoothly.

6.2 Results on the Synthetic Datasets

Sensitivity to Distribution. Figure 5 shows the efficiency of the algorithms under various distributions. It plots the percentage of untracked items as a function of the number of communications. Each plot represents one round. When the datasets are generated from some distribution, our algorithms perform consistently better than the *CMY* algorithms, regardless of the distribution. The communication is reduced by 33% to 80%. Furthermore, our algorithms with static slacks perform better than their *CMY* counterparts. When the distribution is stable, the static slacks capture more accurately the number of items a player will receive.

Sensitivity to Threshold. Figure 6 plots the number of communications as a function of the N (the threshold) under the various distributions, with the number of players fixed to default value 16. We truncate the plots with cost more than 10^3. *UniSlk* performs really good on Uniform distribution as each player receives roughly the same number of items. But its communication cost blows up on other datasets. It does not show up in the plot for *Exponential* distribution because it uses more than 10^3 communication even for $N = 1k$. Moreover, the figures shows an increasing efficiency gain of our algorithms compared to *CMY*, as N increases. This is consistent with our analysis of their communication complexity. While *StcSlk-KwnDst* offers the best performance over all the datasets, *DynSlk-KwnDst*, *StcSlk-LrnDst* and *DynSlk-LrnDst* yield comparable performance.

Sensitivity to Number of Players. Figure 7 plots the number of communications as a function of the number of players under the various distributions, with the threshold fixed to default value $1m$. We truncate the result for *UniSlk* when its communication exceeds 10^4. The figures illustrate a shrinking gap in the number of communications between *CMY* and the our algorithms as the number of players increases. This complies with our theoretical analysis as the ratio of the communication complexity between the two is given by $O((k \log \frac{N}{k})/(k \log\log \frac{N}{k \ln \frac{k}{\delta}} + k \log\log \frac{k}{\delta})) = O((\log \frac{N}{k})/(\log\log \frac{N}{k \ln \frac{k}{\delta}} + \log\log \frac{k}{\delta}))$. When N and δ are fixed, the ratio decreases as k increases. The only exception is the dataset with Exponential distribution and with 2 players, in which the *CMY* perform very well. In such case the number of items the first player receives is roughly exp(1) times that of the second player. A round terminates after player one sending two notifications to the coordinator (the slack size is $s = n_1/2$). On the other hand, player two is assigned the same slack $s = n_1/2$ but receives $n_1/\exp(1)$ items. Only a small fraction of the slack is wasted in this case.

7 CONCLUSION

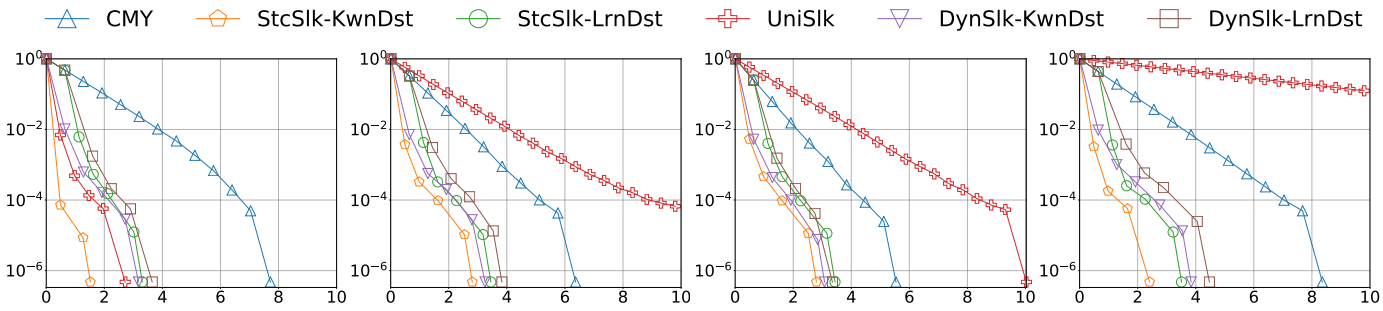
The paper exploits the counter increment distribution and presents four data-dependent algorithms that utilize knowledge on the data distribution. All our algorithms have communication cost $O(k \log\log \frac{N}{k \ln \frac{k}{\delta}} + k \log\log \frac{k}{\delta})$, where δ is a parameter controls the failure probability, improving the state-of-the-art $O(k \log \frac{N}{k})$ bound. In addition, our algorithms are equipped with backup mechanism that guarantees comparable performance as the data-independent *CMY* algorithm when the distribution fluctuates. We experimentally evaluate our algorithms against the state-of-the-art competitors, using both real and synthetic datasets. Our experimental results show the efficiency and robustness of our algorithms.

ACKNOWLEDGMENTS

Junhao Gan is supported by Australian Research Council (ARC) DECRA DE190101118.

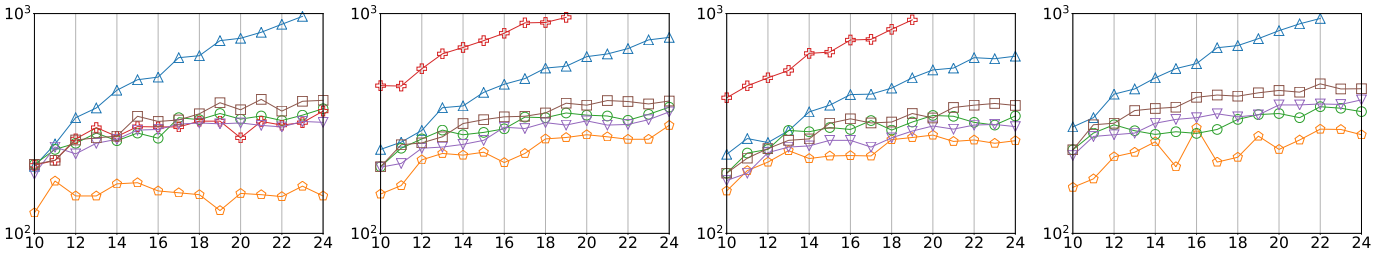
REFERENCES

[1] Anders Aamand, Piotr Indyk, and Ali Vakilian. 2019. (Learned) Frequency Estimation Algorithms under Zipfian Distribution. *CoRR* abs/1908.05198 (2019).
[2] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. 2009. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.* 410, 19 (2009), 1876–1902.



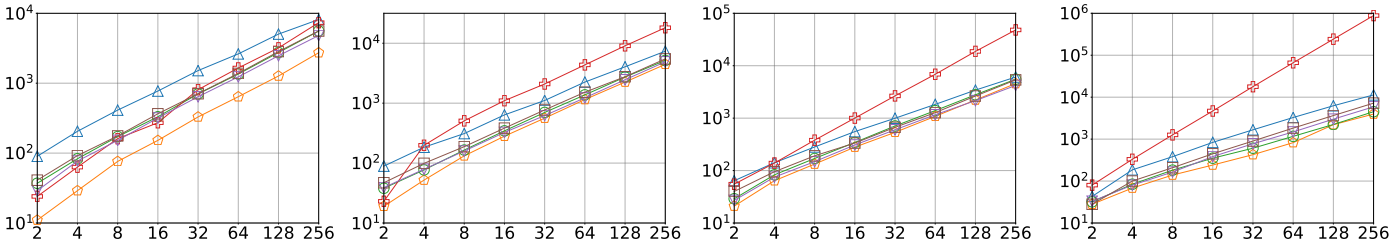
(a) *Uniform* (b) *Gaussian* (c) *Zipfian* (d) *Exponential*

Figure 5: The Untracked Item Percentage v.s. Communication Cost (100x) on synthetic datasets



(a) *Uniform* (b) *Gaussian* (c) *Zipfian* (d) *Exponential*

Figure 6: Communication Cost v.s. N (2^x) on synthetic datasets



(a) *Uniform* (b) *Gaussian* (c) *Zipfian* (d) *Exponential*

Figure 7: Communication Cost v.s. k on synthetic datasets

[3] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings.*

[4] Fan R. K. Chung and Lincoln Lu. 2006. Survey: Concentration Inequalities and Martingale Inequalities: A Survey. *Internet Mathematics* 3, 1 (2006), 79–127.

[5] Graham Cormode. 2013. The continuous distributed monitoring model. *SIGMOD Record* 42, 1 (2013), 5–14.

[6] Graham Cormode, Minos N. Garofalakis, S. Muthukrishnan, and Rajeev Rastogi. 2005. Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005.* 25–36.

[7] Graham Cormode, S. Muthukrishnan, and Ke Yi. 2011. Algorithms for distributed functional monitoring. *ACM Trans. Algorithms* 7, 2 (2011), 21:1–21:20.

[8] Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Izchak Sharfman, and Assaf Schuster. 2012. Prediction-based geometric monitoring over distributed data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012.* 265–276.

[9] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-Based Frequency Estimation Algorithms. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.*

[10] Zengfeng Huang, Ke Yi, and Qin Zhang. 2019. Randomized Algorithms for Tracking Distributed Count, Frequencies, and Ranks. *Algorithmica* 81, 6 (2019), 2222–2243.

[11] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. 2006. Communication-efficient distributed monitoring of thresholded counts. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006.* 289–300.

[12] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA.* 6348–6358.

[13] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. 2009. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). Downloaded from https://crawdad.org/dartmouth/campus/20090909/snmp. https://doi.org/10.15783/C7F59T traceset: snmp.

[14] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* 489–504.

[15] Michael Mitzenmacher. 2018. A Model for Learned Bloom Filters and Optimizing by Sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* 462–471.

[16] Miao Qiao, Junhao Gan, and Yufei Tao. 2016. Range Thresholding on Streams. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016.* 571–582.

[17] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Annual Conference on Neural Information Processing Systems (NeuIPS), December 7-12, 2015, Montreal, Quebec, Canada.* 2692–2700.

A PROOFS OF LEMMAS AND THEOREMS

Proof of Lemma 4.2. Denote $X = n_i = \sum_{j\in[t]} X_j$. As $E[X_j] = \mu_i$, by linearity of expectation and independence, we have $E[X] = t\mu_i$. Further, as the X_j's are independent and $Var[X_j] = \sigma^2 = \mu_i(1-\mu_i)$, $Var[X] = t\sigma^2$. Applying Fact 1 with $Y_j = X_j$, $M = 1$ and setting the failure probability to $\frac{\delta}{k}$, we have

$$\mathbf{Pr}[X - t\mu_i \geq \lambda] \leq \exp\left(-\frac{\lambda^2}{2(t\sigma^2 + \lambda/3)}\right) = \frac{\delta}{k}$$

It follows that

$$\lambda^2 - \left(\frac{2}{3}\ln\frac{k}{\delta}\right)\lambda - 2t\sigma^2\ln\frac{k}{\delta} = 0$$

Solving the quadratic equation gives

$$\begin{aligned}\lambda &= \frac{1}{2}\left(\frac{2}{3}\ln\frac{k}{\delta} + \sqrt{\frac{4}{9}\ln^2\frac{k}{\delta} + 8t\sigma^2\ln\frac{k}{\delta}}\right)\\ &\leq \frac{2}{3}\ln\frac{k}{\delta} + \sqrt{2t\sigma^2\ln\frac{k}{\delta}}\end{aligned}$$

Therefore $UB_i \doteq t\mu_i + \frac{2}{3}\ln\frac{k}{\delta} + \sqrt{2t\sigma^2\ln\frac{k}{\delta}} \geq t\mu_i + \lambda$, and we have

$$\mathbf{Pr}[X \geq UB_i] \leq \mathbf{Pr}[X \geq t\mu_i + \lambda] \leq \frac{\delta}{k}$$

Similarly, if we take $Z_j = -X_j$, then $E[Z_j] = -\mu_i$ and $Var[Z_j] = Var[-X_j] = \sigma^2$. Let $Z = \sum_{j\in[t]} Z_j$, then we have

$$\mathbf{Pr}\left[Z \geq -t\mu_i + \sqrt{2t\sigma^2\ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta}\right] \leq \frac{\delta}{k}$$

which is equivalent to

$$\mathbf{Pr}\left[X \leq t\mu_i - \sqrt{2t\sigma^2\ln\frac{k}{\delta}} - \frac{2}{3}\ln\frac{k}{\delta}\right] \leq \frac{\delta}{k}$$

This finishes the proof. □

Proof of Lemma 4.4. Recall that $s_i = \mu_i t + \sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta}$. Summing over all players $i \in [k]$, we get

$$f(t) = \sum_{i=1}^{k}\mu_i t + \sum_{i=1}^{k}\sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} + \sum_{i=1}^{k}\frac{2}{3}\ln\frac{k}{\delta}$$

The first and third terms sum up to t and $\frac{2k}{3}\ln\frac{k}{\delta}$ respectively. It is left to bound the second term. We utilize the concavity of the square root function $\sqrt{\cdot}$ and obtain

$$\sum_{i=1}^{k}\frac{1}{k}\sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} \leq \sqrt{\sum_{i=1}^{k}\frac{1}{k}\left(2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}\right)} \quad (9)$$

$$\leq \sqrt{\frac{2t}{k}\ln\frac{k}{\delta}} \quad (10)$$

The second inequality follows from $(1-\mu_i) \leq 1$ for $\forall i \in [k]$ and $\sum_{i=1}^{k}\mu_i = 1$. Therefore,

$$f(t) \leq t + \sqrt{2kt\ln\frac{k}{\delta}} + \frac{2k}{3}\ln\frac{k}{\delta} \quad (11)$$

Further, when the *StcSlk-KwnDst* does not run the *CMY* algorithm, it holds that $k\ln\frac{k}{\delta} \leq N$. Hence $\frac{2k}{3}\ln\frac{k}{\delta} \leq \frac{2}{3}\sqrt{kN\ln\frac{k}{\delta}}$. Combining that $t \leq N$, we have

$$f(t) \leq t + \sqrt{2kN\ln\frac{k}{\delta}} + \frac{2}{3}\sqrt{kN\ln\frac{k}{\delta}} \quad (12)$$

It suffices to take $t = N - (\sqrt{2} + 2/3)\sqrt{kN\ln\frac{k}{\delta}}$ to ensure that $f(t) \leq N$ □.

Proof of Lemma 4.5. The claim is trivial true when $N \in (0, 4k)$. If $4k \leq N \leq 2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}$, then

$$T(N) = T(N/2) + O(k) = T(4k) + O(k\log\frac{N}{4k}) = O(k\log\frac{N}{k})$$

Since $N \leq 2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}$, we have $T(N) = O(k\log\log\frac{k}{\delta})$. Finally, if $N > 2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}$, rewrite the following numbers as a power of two:

$$(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta} = 2^{d_1}, \quad N = 2^{d_2}$$

for positive numbers $d_1 = \log\left((\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}\right)$ and $d_2 = \log N$. Define $S(d_2) = T(2^{d_2}) = T(N)$. Then

$$\begin{aligned}S(d_2) &= T\left((\sqrt{2}+2/3)\sqrt{kN\ln\frac{k}{\delta}}\right) + O(k)\\ &= S((d_1+d_2)/2) + O(k)\\ &= S(d_1 + (d_2-d_1)/2) + O(k)\\ &= S(d_1 + (d_2-d_1)/4) + 2\cdot O(k)\\ &= \ldots\\ &= S(d_1 + (d_2-d_1)/2^{\log(d_2-d_1)}) + \log(d_2-d_1)\cdot O(k)\end{aligned}$$

By the definition of $S(\cdot)$, we have $S(d_1 + (d_2-d_1)/2^{\log(d_2-d_1)}) = S(d_1+1) = T(2^{d_1}\cdot 2) = T\left(2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}\right)$. Moreover, $\log(d_2 - d_1) = \log\log 2^{d_2-d_1} = \log\log\frac{N}{(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}}$. Therefore,

$$T(N) = T\left(2(\sqrt{2}+2/3)^2 k\ln\frac{k}{\delta}\right) + O(k\log\log\frac{N}{k\ln\frac{k}{\delta}})$$

The former term equals to $O(k\log\log\frac{k}{\delta})$.

□

Proof of Theorem 4.6. Denote by n_i the number of items received by player i and by I the set of players with $n_i > \mu_i t$ (i.e., the set of players that may send notifications to the coordinator). Our goal is to prove that players in I send less than k notifications[4]:

$$\sum_{i\in I}\frac{n_i - \mu_i t}{(N-t)/2k} < k$$

First notice that by Lemma 4.2, with probability at least $1-\delta$, we have for all $i \in [k]$

$$|n_i - \mu_i t| \leq \sqrt{2t\mu_i(1-\mu_i)\ln\frac{k}{\delta}} + \frac{2}{3}\ln\frac{k}{\delta}$$

[4]Without loss of generality, we assume that $\frac{N-t}{2k}$ is always an integer and thus, we can get rid of the floor operation. This is because otherwise, one can always use at most $2k$ straightforward communications to reduce $N-t$ to a multiple of $2k$. The communication bound will not be affected.

by similar argument as Inequality (9-10), we have

$$\sum_{i\in[k]} |n_i - \mu_i t| \le \sqrt{2kt \ln \frac{k}{\delta}} + \frac{2k}{3} \ln \frac{k}{\delta} \tag{13}$$

On the other hand, we have $\sum_{i=1}^{k} n_i = t$, hence

$$\sum_{i\in I} (n_i - \mu_i t) = \sum_{i\in[k]\setminus I} (\mu_i t - n_i)$$

It follows that

$$\sum_{i\in[k]} |n_i - \mu_i t| = 2 \sum_{i\in I} (n_i - \mu_i t) \tag{14}$$

Combining Inequality (13) and Equality (14), we have

$$\sum_{i\in I} \frac{n_i - \mu_i t}{(N-t)/2k} \le \frac{1}{2} \frac{\sqrt{2kt \ln \frac{k}{\delta}} + \frac{2k}{3} \ln \frac{k}{\delta}}{(N-t)/2k} < k$$

The last inequality can be simplified to $t + \sqrt{2kt \ln \frac{k}{\delta}} + \frac{2k}{3} \ln \frac{k}{\delta} < N$, which holds for $t = N - (\sqrt{2} + 2/3)\sqrt{kN \ln \frac{k}{\delta}}$, as proven in Lemma 4.4.

□

Proof of Theorem 5.1. Denote N_0 the threshold to track when the algorithm begins and let w be the items tracked by *CMY* in the first round. As *CMY* tracks at least half the threshold in one round, it holds that $w \ge N_0/2$. After the first round, the threshold to track is $N \leftarrow N_0 - w$. Therefore, $w \ge N$ holds in all the subsequent rounds.

For any subsequent round that runs our customized tracking algorithm, we are going to prove that: (i) it captures at least t items, where t is defined by Expression (8); (ii) $\sum_{i\in[k]} UB_i \le N$; (iii) $t = N - O(\sqrt{kN \log \frac{k}{\delta}})$, in order to construct a similar recursion as the one that Lemma 4.5 solves. The theorem is proven by the same techniques used by Lemma 4.5.

First, when the first t items arrives, by Equation (4), with probability $1 - \delta$, we have

$$n_i \le \mu_i t + \sqrt{2t\mu_i(1-\mu_i) \ln \frac{k}{\delta}} + \frac{2}{3} \ln \frac{k}{\delta}$$

for all $i \in [k]$. Observing that $(1 - \mu_i) \le 1$ and $\mu_i < \hat{\mu}_i$, we get

$$n_i \le \hat{\mu}_i t + \sqrt{2t\hat{\mu}_i \ln \frac{k}{\delta}} + \frac{2}{3} \ln \frac{k}{\delta}$$

As a result, $UB_i = \hat{\mu}_i t + \sqrt{2t\hat{\mu}_i \ln \frac{k}{\delta}} + \frac{2}{3} \ln \frac{k}{\delta}$ defined by Expression (7) is indeed an upper bound of n_i.

Second, it remains to verify that the these UB_i satisfy that *correctness constraint* (Inequality (1)). Recall that $\sum_{i\in[k]} \hat{\mu}_i = \hat{\Sigma}$. Summing over $i \in [k]$, we have

$$\sum_{i=1}^{k} UB_i = t\hat{\Sigma} + \sum_{i=1}^{k} \sqrt{2t\hat{\mu}_i \ln \frac{k}{\delta}} + \sum_{i=1}^{k} \frac{2}{3} \ln \frac{k}{\delta}$$

The third term sums up to $\frac{2}{3}k \ln \frac{k}{\delta}$. By concavity of the square root function, the second term is upper bounded by $\sqrt{2kt\hat{\Sigma} \ln \frac{k}{\delta}}$. Now, by the definition of $t = N/\hat{\Sigma} - (\sqrt{2} + \frac{2}{3})\sqrt{k(N/\hat{\Sigma}) \ln \frac{k}{\delta}}$ in Expression (8), we have

$$\sum_{i=1}^{k} UB_i \le N - (\sqrt{2} + \frac{2}{3})\sqrt{kN\hat{\Sigma} \ln \frac{k}{\delta}} + \sqrt{2kN \ln \frac{k}{\delta}} + \frac{2}{3} k \ln \frac{k}{\delta}$$

which concludes that $\sum_{i=1}^{k} UB_i \le N$ as $N > k \ln \frac{k}{\delta}$ when this round runs our customized algorithm.

Finally, we need to show that $t = N - O(\sqrt{kN \log \frac{k}{\delta}})$. It suffices to show that $N/\hat{\Sigma} = N - O(\sqrt{kN \log \frac{k}{\delta}})$. By substituting $\hat{\mu}_i$ with Expression (6), and by a similar argument as in Inequality (9-10), we get

$$\hat{\Sigma} \le (1 + \sqrt{\frac{2k \ln \frac{3}{\delta}}{w}} + \frac{3k \ln \frac{3}{\delta}}{w})$$

Therefore,

$$N/\hat{\Sigma} \ge N/(1 + \sqrt{\frac{2k \ln \frac{3}{\delta}}{w}} + \frac{3k \ln \frac{3}{\delta}}{w})$$

Define $g(x) = \sqrt{\frac{2k \ln \frac{3}{\delta}}{x}} + \frac{3k \ln \frac{3}{\delta}}{x}$. By $w \ge N$, it holds $g(w) \le g(N)$. Hence $N(1 + g(w))(1 - g(N)) \le N(1 - g(N)^2) \le N$ and $N/\hat{\Sigma} \ge N/(1 + g(w)) \ge N(1 - g(N))$. Thus, the proof is completed.

□

B IMPLEMENTATION OPTIMISATIONS

Detecting non-stable distribution. In general, the performance of a data-dependent algorithm may degenerate, when the empirical data does not follow the underlying distribution well. This is also the case for our algorithms. To remedy this issue, we propose a simple heuristic to detect whether the current empirical data still follows a distribution well. If it does not, we switch to the *CMY* algorithm right away to minimize the impact of performance degeneration. The heuristic works as follows.

Denote by N the threshold to track at the start of the current round and N' the one at the start of the next round. In other words, $N - N'$ items have been tracked in the current round. Intuitively, the ratio of $\frac{N-N'}{N}$ serves as an indicator of the effectiveness of the algorithm. If the ratio drops below a pre-specified threshold (say 0.75), we switch the algorithm to the *CMY* algorithm. Such a mechanism guarantees that as soon as the empirical distribution is detected to be unstable, our algorithm will lose its effectiveness in at most one round, compared to the *CMY* algorithm. And therefore, at most $O(k)$ communication can be wasted and performance is still upper bounded by the communication bound of the *CMY* algorithm.