Cluster-Reduce: Compressing Sketches for Distributed Data Streams

Yikai Zhao[†]
Peking University

Zheng Zhong[†]
Peking University

Yuanpeng Li[†]
Peking University

Yi Zhou[†]
Peking University

Yifan Zhu[§]
Peking University

Li Chen[¶]
HUAWEI

Yi Wang[‖][‡]
Southern University of Science and Technology

Tong Yang[†][‡]
Peking University

ABSTRACT

[*]Sketches, a type of probabilistic algorithms, have been widely accepted as the approximate summary of data streams. Compressing sketches is the best choice in distributed data streams to reduce communication overhead. The ideal compression algorithm should meet the following three requirements: high *efficiency* of compression procedure, support of *direct query* without decompression, and high *accuracy* of compressed sketches. However, no prior work can meet these requirements at the same time. Especially, the accuracy is poor after compression using existing methods. In this paper, we propose Cluster-Reduce, a framework for compressing sketches, which can meet all three requirements. Our key technique *nearness clustering* rearranges the adjacent counters with similar values in the sketch to significantly improve the accuracy. We use Cluster-Reduce to compress four kinds of sketches in two use-cases: distributed data streams and distributed machine learning. Extensive experimental results show that Cluster-Reduce can achieve up to 60 times smaller error than prior works. The source codes of Cluster-Reduce are available at Github anonymously [1].

CCS CONCEPTS

• **Theory of computation → Data compression**.

KEYWORDS

Sketch Compression, Distributed Data Streams, Distributed ML

[†]Department of Computer Science and Technology, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, China.
[‡]Peng Cheng Laboratory, Shenzhen, China.
[§]School of Software & Microelectrionics, Peking University, China.
[¶]Huawei Theory Lab, China.
[‖]Southern University of Science and Technology.
[*]Yikai Zhao and Zheng Zhong contribute equally to this paper. Tong Yang (yangtongemail@gmail.com) is the corresponding author.

KDD '21, August 14–18, 2021, Virtual Event, Singapore
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467217

ACM Reference Format:
Yikai Zhao, Zheng Zhong, Yuanpeng Li, Yi Zhou, Yifan Zhu, Li Chen, Yi Wang, and Tong Yang. 2021. Cluster-Reduce: Compressing Sketches for Distributed Data Streams. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467217

1 INTRODUCTION

1.1 Background and Motivation

With the increasing volume and velocity of data streams, sketches, a type of probabilistic algorithms, have been widely used in estimating item statistics [2–4], mining frequent items [5, 6], machine learning [7–9], and other data mining tasks on data stream [10, 11]. Sketches deal with large volume of data streams by compactly representing the data with sub-linear memory to record approximate information about the data stream. As for high velocity, per-item computation time of sketches is often constant in the worst case, which is much faster than that of hash tables, heaps or balanced trees [12, 13]. The above two advantages of sketches come at the cost of the approximation error. However, since most Big Data tasks can tolerate small error, and error of sketches is often small and controllable, using sketches in data mining tasks on data streams is both practical and popular.

Sketch compression is an important mechanism in many practical scenarios. We describe two use-cases that highlight the need for sketch compression: geo-distributed data analytics, and distributed machine learning. 1) Geo-distributed data analytics is to estimate the item statistics or find frequent items in geo-distributed data streams, using the sketches of CM [2], CU [3], Count [4], etc. [5, 6, 11, 14]. With the rapid growth of data stream volume, data streams processing may be scattered on multiple devices globally. In these distributed data streams, we need to transmit the sketches deployed in various devices across the network to a central analyzer, which performs the data mining task with a global review. However, this is difficult for geographically distributed devices, as Internet bandwidth can be both limited and unreliable—it is at least one order of magnitude smaller than the local memory, thus becomes a bottleneck [11]. One effective way is to build many sketches with different sizes, and send the appropriated one according to the currently available network bandwidth, but this is uneconomical in terms of computation and memory resources. A more efficient way is to compress the sketches, which can not only save bandwidth, but also make efficient use of memory and computation resources. 2) Distributed machine learning (DML) tasks use sketches to compactly represent the gradients on DML devices, such as SketchML [7], SKETCHED-SGD [8], and FetchSGD [9]. For DML using parameter servers [15], transmitting gradients often takes more time than computing gradients, and becomes the bottleneck of distributed training. To reduce bandwidth usage, researchers use sketches to compactly represent gradients [7–9]. In addition, sketch compression techniques can be employed to further save bandwidth and improve training efficiency. In summary, for the above two use-cases and more scenarios where bandwidth efficiency is crucial, sketch compression is necessary and desirable.

We summarize the design goals of sketch compression mechanism as follows:

- **Efficiency:** In data mining, the value of data decreases rapidly with time passing, the compression speed of sketches must be fast. For DML, speed is also desirable as a slow compression algorithm prolongs the model training.
- **Direct Query:** We desire that the compressed sketch can be queried directly without decompression, which cannot be supported by traditional compression algorithms. This goal is necessitated by two factors: 1) decompressing all the collected sketches in the central analyzer or parameter server is time-consuming; 2) storing all decompressed sketches is memory-consuming.
- **Accuracy:** For direct queries on the compressed sketch, the results should be accurate. Inaccurate query results may cause intolerable disruption to data mining and machine learning.

1.2 Limitations of Prior Art

Most existing works focus on compressing the CM sketch [2], which is widely used in frequency estimation. A CM sketch contains a counter array and d hash functions. For each incoming item, the CM first locates d counters by calculating d hash functions on the item ID, which are abbreviated as d-hash-counters. Then the CM sketch increases these counters by 1. To estimate the frequency of a given item, the CM reports the minimum value of the d-hash-counters. The CM can guarantee the one-side error, *i.e.*, the estimated frequency must be no less than the real frequency. Hokusai [16] is the first to compress the CM sketch. It adds up every two adjacent counters in a CM sketch to halve its memory consumption. And the compressed sketch still guarantees one-side error. An improved work, the Elastic sketch [11], first divides the counter array, and adjacent counters are divided into one group. Then Elastic uses the maximum value of the counters in each group instead of the sum to build the compressed sketch, improving the accuracy.

These works achieve the first two design goals. But they are both inaccurate, because they ignore the relative difference in counters' values. A sketch is a stochastic encoding of item statistics (*e.g.*, frequency), and adjacent counters are independent and skewed, often have great differences. Take the examples in Figure 1(a) and 1(b). In the original CM sketch, the three adjacent counters in a group are $[6, 102, 8]$. If we want a compression ratio of 3, the compression result using Hokusai is $6 + 102 + 8 = 116$, and that

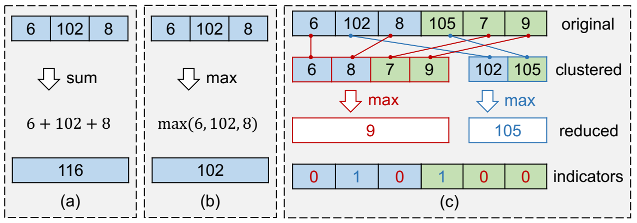


Figure 1: Examples of (a) Hokusai, (b) Elastic, and (c) Cluster-Reduce.

using Elastic is $\max(6, 102, 8) = 102$, significantly overestimating 6 and 8.

1.3 Our Proposed Algorithm

In this paper, we propose Cluster-Reduce, a generic framework to compress various sketches, which is efficient, accurate, and can be queried directly without decompression. Cluster-Reduce meets all the above three design requirements: 1) **Efficiency**: Cluster-Reduce can compress a 160MB sketch within 1 second, using a single thread running on a 4.2GHz CPU. 2) **Direct Query**: the compressed sketch can be directly queried, with a throughput of 14.7M queries per second. 3) **Accuracy**: the compressed sketch can achieve up to 60 times smaller error than the prior works.

Cluster-Reduce has two steps. The first step is named *nearness clustering*, which is the key technique of Cluster-Reduce. Same as Elastic, counters are divided into groups based on adjacency. In this step, we rearrange the chaotic counters in nearby groups into clusters. Afterwards, the values of counters in each clusters should be similar. The second step is named *unique reducing*, which reduces each cluster to a unique representative value. We use the example shown in Figure 1(c) to show how Cluster-Reduce compresses a CM sketch. In the first step, we classify the six counters into two clusters $\{6, 8, 7, 9\}$ and $\{102, 105\}$ according to a *classification strategy*, and use an indicator array $\{0, 1, 0, 1, 0, 0\}$ to indicate which cluster a counter is classified into. In the second step, we take the maximum value of each cluster as the representative, *i.e.*, 8 and 105. The error for each original counter, *i.e.*, the difference with the representative of its cluster, is $[3, 3, 1; 0, 2, 0]$. The total error of 9 is far less than 474 of using Hokusai and 381 of using Elastic.

Nearness clustering is our key technique to achieve accuracy, and we explain its rationale as follows: The counters in each group are often skewed and chaotic. If we use the sum or the maximum value to represent all counters in a group, the largest counter will significantly increase all the small counters, and cause large over-estimation errors to these counters. By classifying counters with similar values in nearby groups into a cluster, using the maximum value as the representative will only cause slight over-estimation error to each counter. Note that *nearness clustering* only allows the counters in near groups to be classified into a cluster, so as to reduce the overhead of time and space during the clustering process.

We propose a dynamic programming based method (§3.1) and an iterative clustering based method (§3.2) to obtain the classification strategy in the case of using 1-bit indicators and multi-bit indicators, respectively. The compressed sketch, namely the *clustered sketch*, can be queried directly without decompression. Take the CM sketch as an example. For a given item, Cluster-Reduce first uses the item ID to find d indicators by calculating hash functions, and locate the corresponding d clusters. Cluster-Reduce uses the minimum value of the d representatives of located clusters as the estimated frequency of the item.

1.4 Contributions

- We propose Cluster-Reduce, a generic framework to compress sketches, which is efficient, accurate, and can be queried directly without decompression.
- We propose two methods to obtain the classification strategy, and analyze the compression error of Cluster-Reduce theoretically.
- We apply Cluster-Reduce to four kinds of sketches, CM [2], CU [3], Count [4], and MinMaxSketch [7], on two use-cases: distributed data streams and DML.
- We perform a comprehensive evaluation of Cluster-Reduce, and the results show that Cluster-Reduce is superior to existing work in distributed data streams and DML applications.

2 BACKGROUND AND RELATED WORK

In this section, we first introduce some typical sketches used in distributed data streams and DML, and then introduce the related work for compressing sketches.

2.1 Sketches in Distributed Data Streams

For distributed data streams, the sketch on each distributed device records approximate information from the local data stream. A central analyzer collects all the sketches, and performs data mining and analysing for a global review. Many sketches are designed to mine item information from data streams. The most typical sketches, including the sketches of CM [2], CU [3], and Count [4], are designed for single data stream, but can be easily extended to distributed data streams. The CM sketch [2] is used to estimate item frequency, and can guarantee the one-side error. The CU sketch [3] improves CM, uses conservative update strategy to achieve higher accuracy, and also guarantees the one-side error. The Count sketch [4] achieves the unbiased estimation of item frequency, and can be combined with a heap to find top-K frequent items. Since the above three kinds of sketches are most widely used, we mainly study how to use Cluster-Reduce to compress them (see §4 for details). There are many other sketches, such as ADA-SKETCH [17], HeavyGuardian [6], WavingSketch [5], MaxLogHash [18], and others [11, 14, 19–23], that focus on estimating item frequency, finding frequent items, measuring network traffic, estimating set similarity, and other data mining tasks.

2.2 Sketches in Distributed Machine Learning

For DML, many sketch based schemes focus on optimizing gradients transmission in parameter server architecture, such as SKETCHED-SGD [8], FetchSGD [9], and SketchML [7]. In the parameter server architecture, there are multiple workers and a parameter server. Each worker trains the model with the local dataset, and transmits the gradients to the parameter server. The parameter server aggregates all the gradients, and sends the updated parameters to each worker. SKETCHED-SGD [8] and FetchSGD [9] use a Count sketch on each worker to compactly record the gradients, and transmits the sketches to the parameter server. The parameter server aggregates all the sketches, recovers the gradients from the aggregated sketch, and updates the parameters on each worker. SketchML [7] uses a Quantile sketch [24] to sort gradient values into buckets, and encodes them with bucket indexes. It also proposes a novel sketch, namely MinMaxSketch, to record the bucket indexes. SketchML transmits two sketches instead of entire gradients, significantly reducing communication costs and accelerating training.

2.3 Related Work for Compressing Sketches

To our best knowledge, there are only a few works about compressing sketches, and most of them focus on the CM sketch. Hokusai [16] is the first work to compress the CM sketch. Given a CM sketch, suppose it has a counter array $\mathcal{A}$ with width w, *i.e.*, $\mathcal{A}$ has w counters. Hokusai halves the width of its counter array by adding up every two adjacent counters. Specifically, for the compressed counter array $\mathcal{A}'$ with width $w/2$, each counter $\mathcal{A}'[i] = \mathcal{A}[2 \cdot i] + \mathcal{A}[2 \cdot i + 1]$. The Elastic sketch [11] proposes a more flexible and accurate method to compress the CM sketch. Elastic divides the counter array into groups, each of which contains the same number of adjacent counters. Elastic creates a compressed counter array with the maximum value of each group of counters, so it can achieve any integer compression ratio λ. Specifically, for the compressed counter array $\mathcal{A}'$ with width w/λ, each counter $\mathcal{A}'[i] = \max(\mathcal{A}[\lambda \cdot i], \mathcal{A}[\lambda \cdot i + 1], \cdots, \mathcal{A}[\lambda \cdot (i+1) - 1])$.

3 THE CLUSTER-REDUCE ALGORITHM

In this section, we use the CM sketch as an example to show how Cluster-Reduce works. Cluster-Reduce is divided into two steps: *nearness clustering* and *unique reducing*. We first present the basic version of nearness clustering, namely the *neighbour clustering*, and then the advanced version, namely the *region clustering*. After that, we present an optional optimization method.

3.1 Neighbour Clustering using 1-bit Indicator

Compressing the CM Sketch (Figure 2): Given an original CM sketch, it contains a counter array $\mathcal{A}$ with width w_o, and is associated with d hash functions $h_1(\cdot), \cdots, h_d(\cdot)$. To compress the CM sketch, we have two steps: neighbour clustering and unique reducing. 1) Neighbour clustering. Given a compression ratio λ, we divide the original sketch into $w_c = w_o/\lambda$ equal-sized groups, each containing λ continuous counters. For every two neighbour groups $\mathcal{G}_k$ and $\mathcal{G}_{k+1}$, we create a cluster C_k by selecting counters from $\mathcal{G}_k$ and $\mathcal{G}_{k+1}$. We propose a dynamic programming (DP) method to classify counters for minimizing the compression error[1], which is detailed later. Each counter in group $\mathcal{G}_k$ will be classified into the left cluster C_k or the right cluster C_{k+1}. We use a 1-bit indicator for each counter to record the cluster it belongs to: 0 indicates left and 1 indicates right. Then we have $w_c + 1$ clusters and an indicator array with w_o bits. 2) Unique reducing. For each cluster C_k, we traverse all counters in it and elect the maximum value as the representative of the cluster. Then we get a representative array with $w_c + 1$ counters. After the above two steps, we compress the CM sketch with w_o counters into a representative array $\mathcal{R}$ with $w_c + 1$ counters and an indicator array $\mathcal{I}$ with w_o bits.

Example 1: As shown in Figure 2, in the first step - *neighbour clustering*, the two larger counters 27 and 19 in the first group $\mathcal{G}_1$ are classified into the left cluster C_1, and their indicators are set to 0; the rest two smaller counters 11 and 9 are classified into the right

[1]The difference between the counters in the original sketch and the representatives of their clusters.

cluster C_2, and their indicators are set to 1. Other counters follow the similar procedure. In the second step - *unique reducing*, the counters in the first cluster C_1 are 27 and 19, so C_1 uses the maximum value, 27, as the representative; the counters in the second cluster C_2 are 11, 9, 8, 3, and 7, so C_2 uses the maximum value, 11, as the representative. Other counters follow the similar procedure.

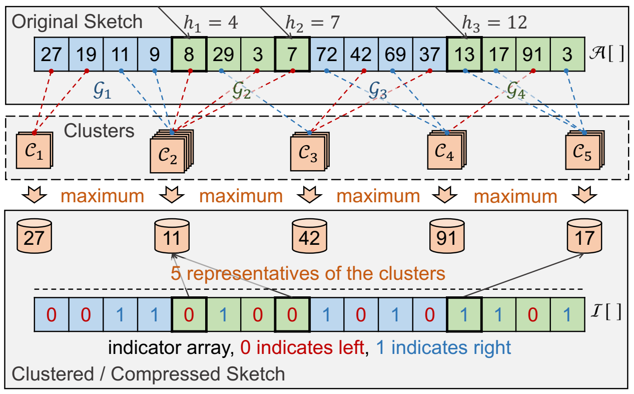


Figure 2: The clustered sketch with 1-bit indicators (parameter setting, $\langle w_o, w_c, \lambda \rangle = \langle 16, 4, 4 \rangle$).

Direct Query: For a given item e, Cluster-Reduce first gets d indicators $\mathcal{I}[h_1(e)], \mathcal{I}[h_2(e)], \cdots, \mathcal{I}[h_d(e)]$ by calculating hash functions, and locates the d corresponding clusters, *i.e.*,

$$C_{\mathcal{I}'_1}, C_{\mathcal{I}'_2}, \cdots, C_{\mathcal{I}'_d} \quad \text{where} \quad \mathcal{I}'_k = \lfloor h_k(e)/\lambda + 1 \rfloor + \mathcal{I}[h_k(e)].$$

Cluster-Reduce then reports the minimum value among d representatives of the located clusters as the estimated frequency of the item, *i.e.*, $\min\left(\mathcal{R}[\mathcal{I}'_1], \mathcal{R}[\mathcal{I}'_2], \cdots, \mathcal{R}[\mathcal{I}'_d]\right)$.

Example 2: As shown in Figure 2, for a given item e, the values calculated by three hash functions $h_1(e), h_2(e), h_3(e)$ are 4, 7, and 12. 1) For the original CM sketch, the estimated frequency of item e is the minimum value of 3-hash-counters, *i.e.*, $\min(\mathcal{A}[4], \mathcal{A}[7], \mathcal{A}[12]) = \min(8, 7, 13) = 7$. 2) For the clustered sketch, we first find that three indicators $\mathcal{I}[4], \mathcal{I}[7], \mathcal{I}[12]$ are 0, 0, and 1, and locate three corresponding clusters C_2, C_2, C_5. The estimated frequency of item e is the minimum value of the representatives of three clusters, *i.e.*, $\min(\mathcal{R}[2], \mathcal{R}[2], \mathcal{R}[5]) = \min(11, 11, 17) = 11$.

Dynamic Programming: As mentioned above, we use a dynamic programming (DP) method to find the optimal classification strategy to minimize the compression error. Now we explain the details. The DP consists of w_c steps. In step k, we obtain several optimal classification strategies, considering only the first k clusters. These can be obtained by extending the results of step $k-1$. Specifically, let $\mathcal{S}_k$ be the classification strategy for the λ counters in the group $\mathcal{G}_k$, *i.e.*, one of 2^λ strategies from $\{0, \cdots, 0\}$ to $\{1, \cdots, 1\}$. Let $O_k[\mathcal{S}_k]$ be the optimal classification strategy for the first k groups when the strategy for the group $\mathcal{G}_k$ is $\mathcal{S}_k$. By enumerating all 2^λ strategies $\mathcal{S}_{k-1}$ for the group $\mathcal{G}_{k-1}$, we can obtain $O_k[\mathcal{S}_k]$ inherited from one of $O_{k-1}[\mathcal{S}_{k-1}]$. The entire optimal classification strategy is the one with the minimal compression error among all 2^λ strategies $O_{w_c}[\mathcal{S}_{w_c}]$. Algorithm 1 (shown in Appendix A) shows the pseudo code, where $\texttt{Error}(\mathcal{A}, k, \mathcal{S})$ is the compression error in the following case: only considering the first k groups of counter array $\mathcal{A}$, and using the classification strategy $\mathcal{S}$.

Complexity Analysis: There are totally $O(w_c \cdot 2^\lambda)$ strategies $O_k[\mathcal{S}_k]$, and obtaining each strategy requires enumerating 2^λ predecessor strategies, thus the time complexity is $O(w_c \cdot 4^\lambda)$. Fortunately, we find that in the optimal strategy, for each group, the larger counters are classified to one cluster, and the remaining ones are classified to the other. Therefore, for each group, only $O(\lambda)$ instead of $O(2^\lambda)$ strategies should be considered, and the time complexity can be reduced to $O\left(w_c \cdot \lambda^2\right)$. The space complexity is $O(w_c \cdot \lambda)$.

3.2 Region Clustering using Multi-bit Indicator

Using 1-bit indicators is simple, but not flexible enough. For example, given 4 counters in a group are 1, 10, 100, and 1000, each counter has only 2 candidate clusters using 1-bit indicators. Unfortunately, classifying them into two clusters will cause very large error. To address this issue, we propose to use multi-bit indicators. Using μ-bit indicators, a counter has 2^μ candidate clusters in a region, empowering Cluster-Reduce much more flexibility to find better classification strategies.

Data Structure: The clustered sketch contains a representatives array with $w_c + 2^\mu - 1$ counters and an indicator array with w_o indicators. The key difference is that each indicator changes from 1-bit to μ-bit. Correspondingly, each counter in group $\mathcal{G}_k$ can be classified into one of the continuous 2^μ clusters in a region, *i.e.*, the clusters $C_k, C_{k+1}, \cdots, C_{k+2^\mu-1}$ nearby the group $\mathcal{G}_k$.

Compression: It uses the same two-step compression procedure as *neighbour clustering*, but uses an iterative method to obtain the classification strategy, which we propose later.

Direct Query: It uses the same query procedure as *neighbour clustering*.

Challenge: When using μ-bit indicators, the number of classification strategies for each group increases exponentially, and the time complexity of DP increases even faster. Therefore, it is necessary to design a more efficient method to obtain a classification strategy, although it may be not so accurate as DP.

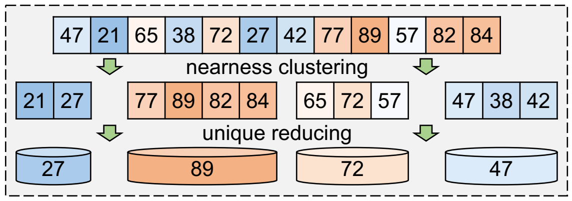


Figure 3: The optimal classification strategy to classify 12 counters into 4 clusters, which is essentially a one-dimensional clustering for integers.

Iterative Clustering: We first give some insights. Figure 3 shows the optimal classification strategy for 12 counters, and the result shows that the optimal classification strategy is actually a local, one-dimensional clustering of integers. Therefore, we extend K-means [25], one of the most widely used low-dimensional clustering methods, to an iterative clustering (IC) algorithm which can be used in the nearness clustering step of Cluster-Reduce. IC consists of multiple iterations, and each iteration has two steps. In the first step - `Nearness-Clustering`, each counter is classified into one of 2^μ nearby clusters with the closest representative (which can be classified randomly in the first iteration). In the second step - `Unique-Reducing`, each cluster uses the maximum of all the counters classified into it as its representative. Algorithm 2 (shown in Appendix A) shows the pseudo code. Our experimental results show that IC can converge within 10 iterations in most cases (see Figure 5 in Section 7.2).

3.3 Optional: Ignoring the Zero-Counters

In a CM sketch reasonably set, there are always a considerable proportion (*e.g.*, 30%) of counters that are 0, *i.e.*, they are not hashed by any items. If we only query the item inserted into the original sketch using the clustered sketch (§ 5.1), Cluster-Reduce can ignore the counters with a value of 0 during compressing, because they will never be accessed. In these cases, ignoring the zero-counters can improve the accuracy of clustered sketch. However, in some cases, we may query items that have not been inserted (§ 5.1), and hope to get an estimated frequency close to 0. In these cases, we cannot use this optional optimization of ignoring the zero-counters.

4 COMPRESSING OTHER SKETCHES

In addition to CM, our Cluster-Reduce can also be used to compress many other sketches. In this section, we take CU, Count, and MinMaxSketch as case studies. We first introduce these sketches and then show how to extend Cluster-Reduce to these sketches.

4.1 The CU Sketch

Data Structure: The CU sketch [3] can be regarded as an optimization of the CM sketch, which can achieve more accurate frequency estimation by using *conservative update* (CU) strategy. The data structure of the CU sketch is the same as that of the CM sketch. The difference is that for each incoming item e, after locating d-hash-counters, the CU sketch only increases the minimal counter(s) by 1. The frequency estimation procedure of the CU sketch is also the same as that of the CM sketch. Therefore, to compress a CU sketch, we can simply use Cluster-Reduce without any modification.

4.2 The Count Sketch

Data Structure: The Count sketch [4] is commonly used to provide unbiased estimation of item frequency. Similar to CM, the Count sketch contains a counter array $\mathcal{A}$ and d hash functions $h_1(\cdot), \cdots, h_d(\cdot)$. Besides, it also contains d additional hash functions $s_1(\cdot), \cdots, s_d(\cdot)$. For each incoming item e, the Count sketch first locates the d-hash-counters. For the i-th hashed counter, it uses hash function $s_i(e)$ to map item e to 1 or -1 uniformly, and adds or subtracts this counter by 1 accordingly. To estimate the frequency of an item e, the Count sketch reports the median value of $s_1(e) \cdot \mathcal{A}[h_1(e)], \cdots, s_d(e) \cdot \mathcal{A}[h_d(e)]$.

Modification: For the Count sketch with two-side error, we follow the key idea of K-means to minimize the compression error, *i.e.*, to take the mean rather than the maximum. When compressing the Count sketch, for each cluster, we use the mean of all counters classified to the cluster as its representative. We use iterative clustering (detailed in Algorithm 2) to compress the Count sketch, replacing the function `Unique-reducing` with Algorithm 3 (shown in Appendix A).

4.3 MinMaxSketch

Data Structure: MinMaxSketch is used to compactly store key-value $\langle K, V\rangle$ pairs in the framework of sketchML [7]. MinMaxSketch also has the same data structure as the CM sketch. To record a key-value pair $\langle K, V\rangle$, for each hashed counter $\mathcal{A}[h_i(K)]$, MinMaxSketch updates it to the smaller value between V and $\mathcal{A}[h_i(K)]$. To query the value of a given key K, MinMaxSketch reports the maximum value among d-hash-counters.

Modification: An important property of MinMaxSketch is that each counter $\mathcal{A}[i]$ does not exceed the value of any key hashed to $\mathcal{A}[i]$. To guarantee this, for each cluster, we use the minimum one (rather than the maximum one) of all counters classified to the cluster as its representative. We also use iterative clustering (IC) to compress MinMaxSketch, replacing the function `Unique-reducing` with Algorithm 4 (shown in Appendix A).

5 APPLICATIONS

In this section, we introduce two distributed applications of Cluster-Reduce: estimating item frequency in distributed data streams, and transferring gradients/parameters in DML.

5.1 Distributed Data Streams

Distributed data streams consist of multiple data streams. Each data stream contains many items, and the ID/key of each item e consists of n attributes, *i.e.*, $e = \langle a_1, \cdots, a_n\rangle$. Each data stream is assigned to a distributed device, and there is a central analyzer that collects information and performs analyses. There are two kinds of analyses on distributed data streams: exact key frequency estimation and partial key frequency estimation.

Exact Key Frequency Estimation measures the number of times an exact key $\langle a_1, \cdots, a_n\rangle$ appears in the distributed data streams. To support this kind of estimation, we set up a sketch on each distributed device according to the available local memory, which can be CM [2], CU [3], Count [4], *etc.* On each device, for each incoming item e, we use the exact key $\langle a_1, \cdots, a_n\rangle$ to insert it into the corresponding sketch. After each time period, each device uses Cluster-Reduce to *compress* the sketch according to the available network bandwidth, and then transmits the compressed sketch to the analyzer through network. For a given exact key, the analyzer first determines which devices it may appear in, and then queries the corresponding sketches. In this case, Cluster-Reduce uses the optional optimization: ignoring the zero-counters.

Partial Key Frequency Estimation measures the number of times a partial key appears in the distributed data streams. A partial key is a pattern with wildcards, *i.e.*, $\langle \cdots, *, a_{p_1}, *, \cdots, *, a_{p_m}, *, \cdots\rangle$. We follow the same procedure as exact key frequency estimation, except using the partial key $\langle a_{p_1}, \cdots, a_{p_m}\rangle$ instead of the exact key to insert each item into the sketch. For a given partial key, the analyzer cannot determine which data stream contains those items that satisfy the pattern. So the analyzer queries all sketches for the estimated frequency, and reports the sum of all results. In this case, Cluster-Reduce needs to consider the zero-counters, thus cannot use the optimization.

5.2 Distributed Machine Learning

We take SketchML [7] as an example of the DML. SketchML consists of multiple workers and a parameter server, and each worker uses two components to transmit gradients to the parameter server. SketchML uses a Quantile sketch to encode each gradient value into an integer within a range, such as $[0, 1023]$, and uses a MinMaxSketch to record the encoded gradients. By using Cluster-Reduce, we can build a relatively large MinMaxSketch on each worker according to the available local memory, and then *compress* it to a relatively small size according to the available network bandwidth when transmitting it to the parameter server.

6 MATHEMATICAL ANALYSIS

In this section, we analyze the error of compressing a CM sketch using DP (*i.e.*, 1-bit indicators). We discuss the two cases of *considering the zero-counters* and *ignoring the zero-counters*, respectively.

6.1 Considering the Zero-Counters

In this section, we calculate the compression error of some simpler classification strategies, and use the minimum of them to bound the error of DP in the case of *considering the 0-counters*.

Given an original CM sketch containing a counter array $\mathcal{A}$ with width w_o, and a compression ratio λ. We consider the following classification strategy, namely *Greedy-t*: For each $k \in [1, w_c/2]$, where $w_c = w_o/\lambda$, we classify the smallest t of the $2 \cdot \lambda$ counters in the groups $\mathcal{G}_{2\cdot k-1}$ and $\mathcal{G}_{2\cdot k}$ to the cluster $C_{2\cdot k}$; classify the remaining counters in the group $\mathcal{G}_{2\cdot k-1}$ to the cluster $C_{2\cdot k-1}$; and classify the remaining counters in the group $\mathcal{G}_{2\cdot k}$ to the cluster $C_{2\cdot k+1}$. We have the following lemma.

Lemma 6.1. *Given an original CM sketch containing a counter array $\mathcal{A}$ with width w_o and d hash functions, which is generated by n items, and a compression ratio λ. We use Greedy-t to obtain a classification strategy $\mathcal{I}_t$, and perform the unique reducing. Then we have the following bound on the compression error:*

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}_t[\mathcal{I}_t'[i]] - \mathcal{A}[i]\right)\right) \leqslant \left((2\cdot\lambda - t + 1) + \left(\frac{2\cdot\lambda+1}{2\cdot\lambda - t + 1}\right) - 3\right)\cdot(d\cdot n),$$

where $\mathcal{I}_t'[i] = \lfloor i/\lambda + 1\rfloor + \mathcal{I}_t[i]$ is the index of the cluster to which counter $\mathcal{A}[i]$ is classified.

Due to the limitation of space, the proof of the above Lemma 6.1 is shown in Appendix B. We now give the following bounds of the compression error of DP.

Theorem 6.2. *Given an original CM sketch containing a counter array $\mathcal{A}$ with width w_o and d hash functions, which is generated by n items, and a compression ratio λ. We use DP (considering the 0-counters) to obtain the optimal classification strategy $\mathcal{I}$, and perform the unique reducing. Then we have the following bound on the compression error:*

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}[\mathcal{I}'[i]] - \mathcal{A}[i]\right)\right) \leqslant \left(2\cdot\sqrt{2\cdot\lambda+1} - 3\right)\cdot(d\cdot n),$$

where $\mathcal{I}'[i] = \lfloor i/\lambda + 1\rfloor + \mathcal{I}[i]$ is the index of the cluster to which counter $\mathcal{A}[i]$ is classified.

Proof. Since we can obtain the optimal classification strategy by using DP, *i.e.*, the plan with the minimum compression error, so $\forall t$ we have

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}[\mathcal{I}'[i]] - \mathcal{A}[i]\right)\right) \leqslant E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}_t[\mathcal{I}_t'[i]] - \mathcal{A}[i]\right)\right)$$
$$\leqslant \left((2\cdot\lambda - t + 1) + \left(\frac{2\cdot\lambda+1}{2\cdot\lambda - t + 1}\right) - 3\right)\cdot(d\cdot n).$$

So we take lower bound for all t on the right side of the inequality, and then we have

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}[\mathcal{I}'[i]] - \mathcal{A}[i]\right)\right)$$
$$\leqslant \inf\left\{\left((2\cdot\lambda - t + 1) + \left(\frac{2\cdot\lambda+1}{2\cdot\lambda - t + 1}\right) - 3\right)\cdot(d\cdot n)\right\}$$
$$= \left(2\cdot\sqrt{2\cdot\lambda+1} - 3\right)\cdot(d\cdot n).$$

□

6.2 Ignoring the Zero-Counters

In this section, we use a method similar to the Section 6.1 to give the bound of compression error of DP in the case of *ignoring the 0-counters*. Due to the limitation of space, we only give the conclusion.

Theorem 6.3. *Given an original CM sketch containing a counter array $\mathcal{A}$ with width w_o and d hash functions, which is generated by n items (m distinct IDs), and a compression ratio λ. We use DP (ignoring the 0-counters) to obtain the optimal classification strategy $\mathcal{I}$, and perform the unique reducing. Then we have the following bound on the compression error:*

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}[\mathcal{I}'[i]] - \mathcal{A}[i]\right)\right) \leqslant \left(2\cdot\sqrt{2\cdot\lambda\cdot(1-p)+p} - 2\right)\cdot(d\cdot n),$$

where $\mathcal{I}'[i] = \lfloor i/\lambda + 1\rfloor + \mathcal{I}[i]$ is the index of the cluster to which counter $\mathcal{A}[i]$ is classified, and $p = \exp(-d\cdot m/w_o)$.

7 EXPERIMENTAL RESULTS

7.1 Experimental Setup

Implementation: We have implemented both dynamic programming (DP) based Cluster-Reduce, iterative clustering (IC) based Cluster-Reduce, and two existing methods from Hokusai [16] and Elastic [11] in C++. We apply these algorithms on CM [2], CU [3], and Count [2] [4]. We also simulate SketchML in C++, and implement IC on MinMaxSketch.

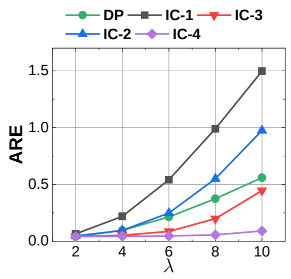
Datasets, Platform, and Metrics: See details in Appendix C.

Due to the limitation of space, we only show the experiments on CM and Count on the CAIDA dataset here. Experiments on CU and other datasets are shown in Appendix D and E.

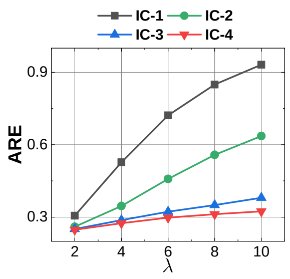
7.2 Experiments on Parameter Settings

In this section, we compare the performance of Cluster-Reduce using different parameter settings. Let $\mathcal{M}_o$ and $\mathcal{M}_b$ be the memory usage of the original sketch and the clustered sketch, respectively. Let IC-μ be the IC based Cluster-Reduce with μ-bit indicators, and let λ be the compression ratio.

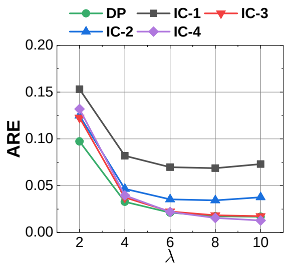
[2]To compress Count, IC and Hokusai are applicable, while DP and Elastic are not.



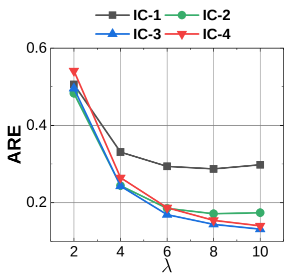
(a) ARE *v.s.* $\mathcal{M}_b$ on CM.



(b) ARE *v.s.* $\mathcal{M}_b$ on Count.

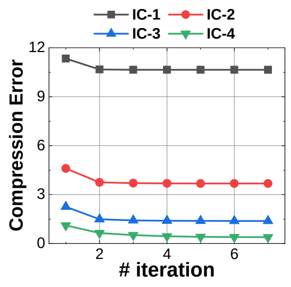


(c) ARE *v.s.* $\mathcal{M}_o$ on CM.

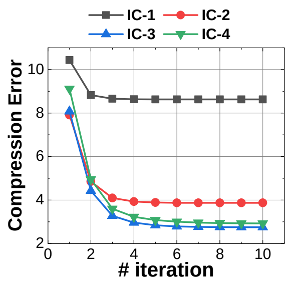


(d) ARE *v.s.* $\mathcal{M}_o$ on Count.

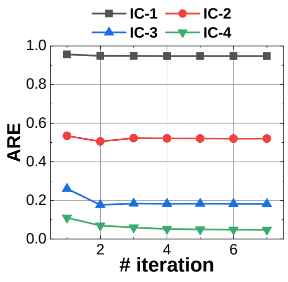
Figure 4: Comparison of loss and accuracy on different local memory and available bandwidth.



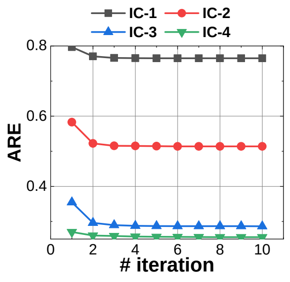
(a) Error *v.s.* # iteration on CM.



(b) Error *v.s.* # iteration on Count.

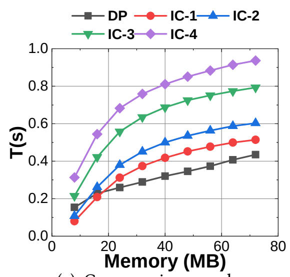


(c) ARE *v.s.* # iteration on CM.

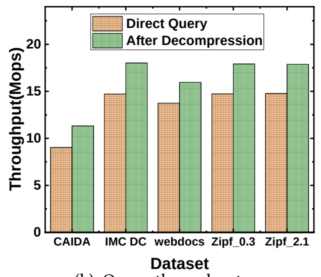


(d) ARE *v.s.* # iteration on Count.

Figure 5: Comparison of accuracy on # iteration.



(a) Compression speed.



(b) Query throughput.

Figure 6: Experiments on efficiency.

Accuracy *v.s.* $\mathcal{M}_b$ (Figure 4(a)-(b)): We find that, when limiting $\mathcal{M}_o$ and λ, larger μ usually brings higher accuracy: when $\lambda = 8$, the ARE of IC-4 on CM is 17.7 times lower than that of IC-1. Besides, with the increasing of λ, the accuracy of DP is higher than that of IC-1, and it decays slower: when $\lambda = 2$, the ARE of DP on CM is 1.57 times lower than that of IC-1, while when $\lambda = 10$, the ARE of DP is 2.67 times lower. Therefore, for 1-bit indicators, we recommend to use DP to compress CM and CU to achieve the highest accuracy.

Accuracy *v.s.* $\mathcal{M}_o$ (Figure 4(c)-(d)): We find that, when limiting $\mathcal{M}_b$, the accuracy of IC does not always increase as μ increases: when $\lambda = 2$, the ARE of IC-4 on CM is 1.07 times higher than that of IC-3. This is because the wider indicators take up more memory, resulting in smaller $\mathcal{M}_o$. Therefore, for multi-bit indicators, we recommend to use IC-3 to achieve the highest accuracy.

Accuracy *v.s.* # Iteration (Figure 5): We find that IC always converges within very few iterations: IC-3 on CM and Count converge within 4 and 8 iterations, respectively.

Efficiency (Figure 6): We find that the compression speed of DP and IC-3 reach 1.29Gbps and 0.71Gbps, respectively. The direct query throughput of Cluster-Reduce can achieve 14.7Mops (10^6 queries per second), comparable to the query throughput after decompression (*i.e.*, recovering each counter using its representative).

7.3 Experiments on Single Data Stream

In this section, we compare the accuracy of Cluster-Reduce and two existing methods, Hokusai [16] and Elastic [11], in single data stream.

Accuracy *v.s.* $\mathcal{M}_b$ (Figure 7): We find that, when $\lambda = 8$, the ARE of DP and IC-3 on CM are about 2.68/5.87 and 5.07/11.1 times lower than that of Elastic/Hokusai, respectively.

Accuracy *v.s.* $\mathcal{M}_o$ (Figure 8): We find that, when $\lambda = 8$, the ARE of DP and IC-3 on CM are about 8.23/21.3 and 7.76/20.1 times lower than that of Elastic/Hokusai, respectively.

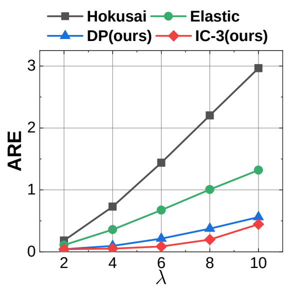
Analysis: The experimental results show that Cluster-Reduce performs best on both cases. When limiting the original memory $\mathcal{M}_o$, Cluster-Reduce brings least effect on accuracy. When limiting the available bandwidth $\mathcal{M}_b$, Cluster-Reduce can significantly improve the accuracy. The above advantages are more significant when the compression ratio λ is higher.

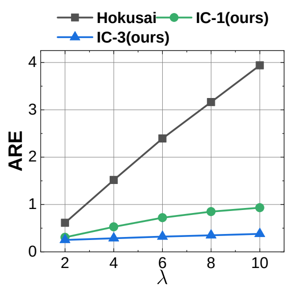
7.4 Experiments on Distributed Data Streams

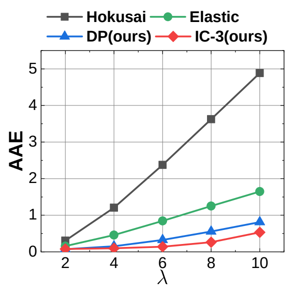
In this section, we compare the accuracy of Cluster-Reduce and two existing methods, Hokusai [16] and Elastic [11], in distributed data streams. In this case, there are 8 distributed devices and a central analyzer. Each item in the data stream is identified by 5-tuple ⟨`src IP`, `dst IP`, `src port`, `dst port`, `protocol`⟩, and we assign each item to one of 8 distributed devices according to 5-tuple.

Exact Key Frequency Estimation (Figure 9) : We take the 5-tuple as the exact key. We find that, when $\mathcal{M}_b \approx$ 1MB, the ARE of DP and IC-3 are about 4.52/7.48 and 2.48/4.10 times lower than that of Elastic/Hokusai, respectively.

Partial Key Frequency Estimation (Figure 10): We take the ⟨`src IP`, $*, *, *, *$⟩ as the partial key. We find that, when $\mathcal{M}_b \approx$ 1MB, the ARE of DP and IC-3 are about 8.43/9.65 and 11.6/13.2 times lower than that of Elastic/Hokusai, respectively.


(a) ARE *v.s.* λ on CM.


(b) ARE *v.s.* λ on Count.


(c) AAE *v.s.* λ on CM.

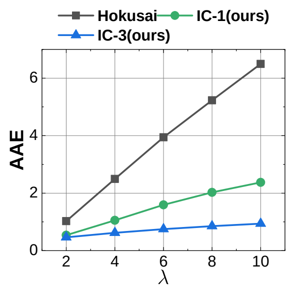
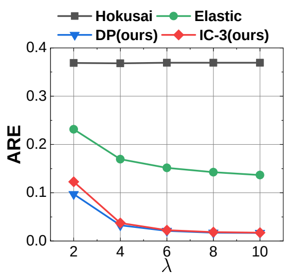
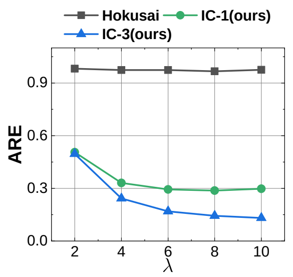
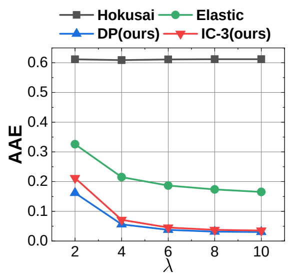

(d) AAE *v.s.* λ on Count.

Figure 7: Comparison on performance with limited original memory.


(a) ARE *v.s.* λ on CM.


(b) ARE *v.s.* λ on Count.


(c) AAE *v.s.* λ on CM.

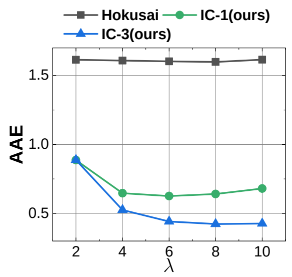
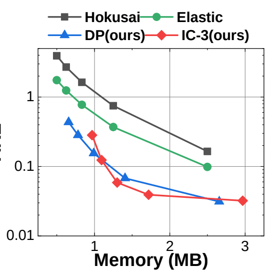
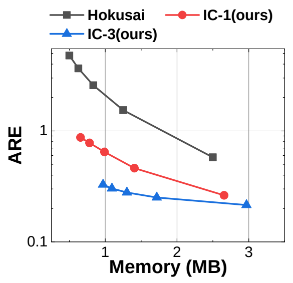
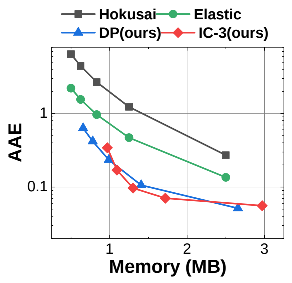

(d) AAE *v.s.* λ on Count.

Figure 8: Comparison on performance with limited available bandwidth.


(a) ARE *v.s.* $\mathcal{M}_b$ on CM.


(b) ARE *v.s.* $\mathcal{M}_b$ on Count.


(c) AAE *v.s.* $\mathcal{M}_b$ on CM.

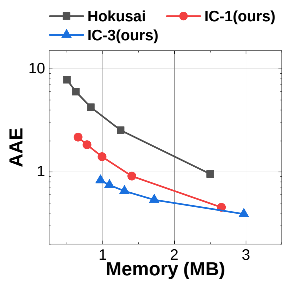
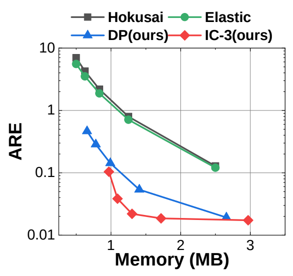

(d) AAE *v.s.* $\mathcal{M}_b$ on Count.

Figure 9: Comparison in exact key frequency estimation in distributed data streams.


(a) ARE *v.s.* $\mathcal{M}_b$ on CM.

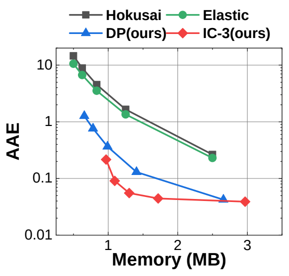

(b) AAE *v.s.* $\mathcal{M}_b$ on CM.

Figure 10: Comparison in partial key frequency estimation in distributed data streams.

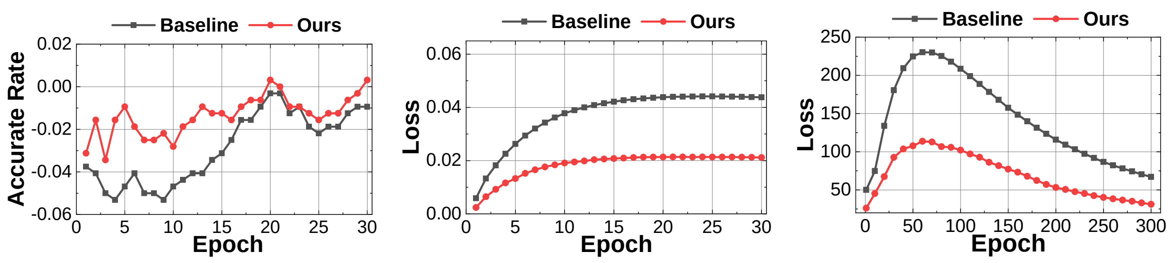
Analysis: The experimental results show that Cluster-Reduce performs best on both tasks. It worth notice that the accuracy of IC-3 is outstanding on partial key frequency estimation, which hopes that the sketch can give an estimated frequency close to 0 for items that do not appear in the data stream. IC-3 is more suitable for this task. The items that do not appear in the data stream are hashed into the 0-counters, and they will monopolize some clusters when using IC-3, so that 0 can be retained.

7.5 Experiments on Distributed ML

In this section, we compare the performance of SketchML [7] with (ours) and without using Cluster-Reduce (baseline). We conduct experiments on two distributed applications: classification (using logistic regression) and regression (using linear regression). In this case, there are 8 works and a parameter server, and we limit the available bandwidth $\mathcal{M}_b$ for SketchML, which is about $\frac{1}{6}$ of that of transmitting exact gradients.

Classification (Figure 11(a)-(b)): We find that SketchML with IC can improve the accurate rate by 1.5% on average, and reduce the loss by 4.8%: after 30 epochs, the loss of SketchML with IC reaches 0.445, and that of baseline is 0.468. Note that the loss of transmitting exact gradients is 0.424.

Regression (Figure 11(c)): We find that SketchML with IC can improve the training speed by 20.0% on average: when the loss reaches 250, SketchML with IC uses 240 epochs, and baseline uses 300 epochs. Note that the method of transmitting exact gradients uses 190 epochs. Training faster means achieving higher accuracy when using the same time.



(a) Accurate rate on classification.
(b) Loss on classification.
(c) Loss on regression.

Figure 11: Comparison of SketchML with Cluster-Reduce (ours) and SketchML without Cluster-Reduce (baseline) in DML. To highlight the advantages of Cluster-Reduce, the data shown in the figure is the difference between SketchML with/without Cluster-Reduce and an ideal method of transmitting exact gradients.

Analysis: The experimental results shows that, Cluster-Reduce can improve the training speed and accuracy in DML. Cluster-Reduce can improve the accuracy of the sketches for transmitting gradients. Therefore, the parameter server can recover gradients more accurately and update parameters more precisely, making the training faster and more accurate.

8 CONCLUSION

In this paper, we propose Cluster-Reduce, a generic framework for compressing sketches. Cluster-Reduce meets all three requirements of designing a sketch compression algorithm: it achieves a compression speed of more than 1.3Gbps, a direct query throughput of more than 14.7Mops, and an estimation error of up to 60 times smaller than the existing works. Cluster-Reduce has two steps: *nearness clustering* and *unique reducing*. In the first step, we propose two methods, a dynamic programming (DP) based method and an iterative clustering (IC) based method, to obtain optimal or fast classification strategy. We theoretically derive the error bounds for compressing the sketch using DP. We use Cluster-Reduce to compress four kinds of sketches in two use-cases: distributed data streams and DML. Experimental results show that Cluster-Reduce is significantly superior to existing tasks in both use-cases. The source codes of Cluster-Reduce are available at Github anonymously [1].

ACKNOWLEDGMENT

We thank Peng Liu for his helpful suggestions and discussions. This work is supported by National Natural Science Foundation of China (NSFC) (No. U20A20179), the project of "FANet: PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications" (No. LZC0019), and Computing and Network Innovation Lab, Huawei Cloud.

REFERENCES

[1] Source code related to Cluster-Reduce. https://github.com/Cluster-Reduce/Cluster-Reduce.
[2] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
[3] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *TOCS*, 21(3):270–313, 2003.
[4] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703. Springer, 2002.
[5] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *ACM SIGKDD*, pages 1574–1584, 2020.
[6] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *ACM SIGKDD*, pages 2584–2593, 2018.
[7] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. Sketchml: Accelerating distributed machine learning with data sketches. In *ACM SIGMOD*, pages 1269–1284, 2018.
[8] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. *NIPS*, 2019.
[9] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *ICML*, pages 8253–8265. PMLR, 2020.
[10] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *ACM SIGMOD*, pages 1449–1463, 2016.
[11] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *ACM SIGCOMM*, pages 561–575, 2018.
[12] R Bayer and E McCreight. Organization and maintenance of large ordered indices. In *ACM SIGFIDET*, pages 107–141, 1970.
[13] Leo J Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *FOCS*, pages 8–21. IEEE, 1978.
[14] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *ACM SIGMOD*, pages 1129–1140, 2018.
[15] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, pages 583–598, 2014.
[16] Sergiy Matusevych, Alexander J Smola, and Amr Ahmed. Hokusai—sketching streams in real time. In *UAI*, pages 594–603, 2012.
[17] Anshumali Shrivastava, Arnd Christian Konig, and Mikhail Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. In *ACM SIGMOD*, pages 1417–1432, 2016.
[18] Pinghui Wang, Yiyan Qi, Yuanming Zhang, Qiaozhu Zhai, Chenxu Wang, John CS Lui, and Xiaohong Guan. A memory-efficient sketch method for estimating high similarities in streaming sets. In *ACM SIGKDD*, pages 25–33, 2019.
[19] Daniel Ting. Count-min: optimal estimation and tight error bounds using empirical error distributions. In *ACM SIGKDD*, pages 2319–2328, 2018.
[20] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *ACM SIGKDDng*, pages 1015–1025, 2020.
[21] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In *ACM SIGMOD*, pages 757–772, 2018.
[22] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *ACM SIGMOD*, pages 795–810, 2015.
[23] Dingqi Yang, Paolo Rosso, Bin Li, and Philippe Cudre-Mauroux. Nodesketch: Highly-efficient graph embeddings via recursive sketching. In *ACM SIGKDD*, pages 1162–1172, 2019.
[24] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD*, pages 58–66, 2001.
[25] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.
[26] The CAIDA Anonymized Internet Traces. http://www.caida.org/data/overview/.
[27] Data Set for IMC 2010 Data Center Measurement. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.
[28] Real-life transactional dataset. http://fimi.ua.ac.be/data/.
[29] Alex Rousskov and Duane Wessels. High-performance benchmarking with web polygraph. *Software: Practice and Experience*, 34(2):187–211, 2004.
[30] UC Irvine Machine Learning Repository. http://archive.ics.uci.edu/ml/index.php.

A PSEUDO CODES

Algorithm 1: Dynamic Programming based Method.

```
for $\mathcal{S}_1 = \{0, \cdots, 0\} \to \{1, \cdots, 1\}$ do
    $\mathcal{E}_1[\mathcal{S}_1] \leftarrow$ Error($\mathcal{A}, 1, \mathcal{S}_1$) ;
    $O_1[\mathcal{S}_1] \leftarrow \mathcal{S}_1$ ;
for $k = 2 \to w_c$ do
    for $\mathcal{S}_k = \{0, \cdots, 0\} \to \{1, \cdots, 1\}$ do
        $\mathcal{E}_k[\mathcal{S}_k] \leftarrow \infty$ ;
        for $\mathcal{S}_{k-1} = \{0, \cdots, 0\} \to \{1, \cdots, 1\}$ do
            if Error($\mathcal{A}, k, O_{k-1}[\mathcal{S}_{k-1}] \bigcup \mathcal{S}_k$) $< \mathcal{E}_k[\mathcal{S}_k]$
             then
                $\mathcal{E}_k[\mathcal{S}_k] \leftarrow$ Error($\mathcal{A}, k, O_{k-1}[\mathcal{S}_{k-1}] \bigcup \mathcal{S}_k$) ;
                $O_k[\mathcal{S}_k] \leftarrow O_{k-1}[\mathcal{S}_{k-1}] \bigcup \mathcal{S}_k$ ;
$\mathcal{E}_{entire} \leftarrow \infty$;
for $\mathcal{S}_{w_c} = \{0, \cdots, 0\} \to \{1, \cdots, 1\}$ do
    if Error($\mathcal{A}, w_c, O_{w_c}[\mathcal{S}_{w_c}]$) $< \mathcal{E}_{entire}$ then
        $\mathcal{E}_{entire} \leftarrow$ Error($\mathcal{A}, w_c, O_{w_c}[\mathcal{S}_{w_c}]$) ;
        $O_{entire} \leftarrow O_{w_c}[\mathcal{S}_{w_c}]$ ;
return $O_{entire}$ ;
```

Algorithm 2: Iterative Clustering based Method.

```
for $iter = 0 \to T$ do
    Nearness-clustering($iter$);
    Unique-reducing($iter$);
Function Nearness-clustering($iter$):
    if $iter = 0$ then
        for $i = 0 \to w_o - 1$ do
            $\mathcal{I}[i] \leftarrow$ Random $(0, 2^{\mu} - 1)$;
    else
        for $i = 0 \to w_o - 1$ do
            $min\_dist \leftarrow +\infty$;
            for $j = 0 \to 2^{\mu} - 1$ do
                $C = \lfloor i/\lambda + 1 \rfloor + j$;
                if Abs $(\mathcal{A}[i] - \mathcal{R}[C]) < min\_dist$ then
                    $\mathcal{I}[i] \leftarrow j$;
                    $min\_dist \leftarrow$ Abs $(\mathcal{A}[i] - \mathcal{R}[C])$;
Function Unique-reducing($iter$):
    for $i = 1 \to w_c + 2^{\mu} - 1$ do
        $\mathcal{R}[i] = 0$;
        foreach $(\lfloor j/\lambda + 1 \rfloor + \mathcal{I}[j]) = i$ do
            $\mathcal{R}[i] = \max(\mathcal{R}[i], \mathcal{A}[j])$;
```

Algorithm 3: `Unique-reducing` for the Count sketch.

```
Function Unique-reducing($iter$):
    for $i = 0 \to w_c + 2^{\mu} - 1$ do
        $\mathcal{V} = \{\}$;
        foreach $(\lfloor j/\lambda + 1 \rfloor + \mathcal{I}[j]) = i$ do
            $\mathcal{V}$.add($\mathcal{A}[j]$);
        $\mathcal{R}_t[i]$ = mean($\mathcal{V}$);
```

B PROOF OF LEMMA 6.1

Lemma B.1. *Given an original CM sketch containing a counter array $\mathcal{A}$ with width w_o and d hash functions, which is generated by n items, and a compression ratio λ. We use Greed-t to obtain a classification strategy $\mathcal{I}_t'$, and perform the unique reducing. Then we have the following bound on the compression cost:*

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}_t[\mathcal{I}_t'[i]] - \mathcal{A}[i]\right)\right) \leqslant \left((2\cdot\lambda - t + 1) + \left(\frac{2\cdot\lambda + 1}{2\cdot\lambda - t + 1}\right) - 3\right)\cdot(d\cdot n).$$

Algorithm 4: `Unique-reducing` for MinMaxSketch.

```
Function Unique-reducing($iter$):
    for $i = 1 \to w_c + 2^{\mu} - 1$ do
        $\mathcal{R}_t[i] = \infty$;
        foreach $(\lfloor j/\lambda + 1 \rfloor + \mathcal{I}[j]) = i$ do
            $\mathcal{R}_t[i] = \min(\mathcal{R}_t[i], \mathcal{A}[j])$;
```

Proof. Recall that $\mathcal{I}_t'[i]$ indicate the cluster to which the counter $\mathcal{A}[i]$ is classified, and $\mathcal{R}_t[k]$ is the representative of the cluster $\mathcal{G}_k$, then we have:

$$E\left(\sum_{i=0}^{w_o-1}\left(\mathcal{R}_t[\mathcal{I}_t'[i]] - \mathcal{A}[i]\right)\right) = E\left(\sum_{i=0}^{w_o-1}\mathcal{R}_t[\mathcal{I}_t'[i]]\right) - (d\cdot n). \quad (1)$$

According to our classification strategy obtained by *Greedy-t* (suppose $w_o \equiv 0 \pmod{2\cdot\lambda}$), we can rearrange the first term of Formula 1 as follows:

$$E\left(\sum_{i=0}^{w_o-1}\mathcal{R}_t[\mathcal{I}_t'[i]]\right)$$
$$= E\left(\sum_{k=0}^{w_c/2}\mathcal{R}_t[2\cdot k+1]\cdot(r_{2\cdot k} + r_{2\cdot k+1}) + \sum_{k=1}^{w_c/2}\mathcal{R}_t[2\cdot k]\cdot t\right), \quad (2)$$

where $r_{2\cdot k}$ is the number of *remaining counters* in the group $\mathcal{G}_{2\cdot k}$ ($r_{2\cdot k+1}$ is similar). In particular, let $r_0 = r_{w_c+1} = 0$. Since w_o original counters are *i.i.d.*, all representative $\mathcal{R}_t[2\cdot k]$ and $\mathcal{R}_t[2\cdot k+1]$ are also *i.i.d.*, respectively. And since *i.i.d.* random variables have the same expectations, we can simplify Formula 2 as follows:

$$E\left(\sum_{i=0}^{w_c-1}\mathcal{R}_t[\mathcal{I}_t'[i]]\right) = \left(\sum_{k=0}^{w_c/2} r_{2\cdot k} + r_{2\cdot k+1}\right)\cdot E(\mathcal{R}_t[2\cdot k+1]) + \left(\sum_{k=1}^{w_c/2} t\right)\cdot E(\mathcal{R}_t[2\cdot k])$$
$$= \left(w - \frac{w_o}{2\cdot\lambda}\cdot t\right)\cdot E(\mathcal{R}_t[2\cdot k+1]) + \left(\frac{w_o}{2\cdot\lambda}\cdot t\right)\cdot E(\mathcal{R}_t[2\cdot k]). \quad (3)$$

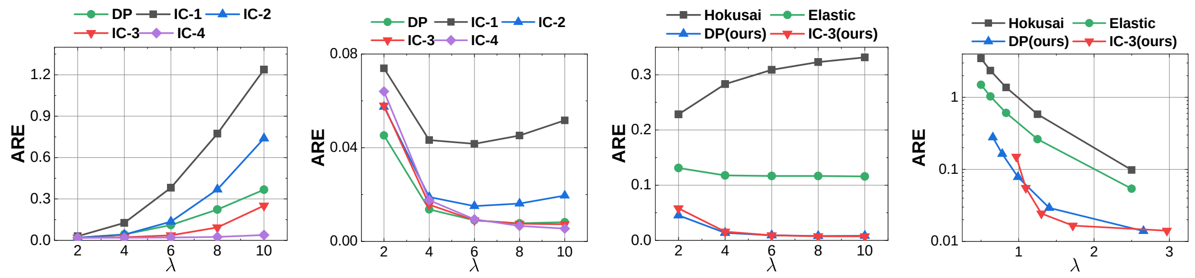
Next, we analyze $E(\mathcal{R}_t[2\cdot k+1])$ and $E(\mathcal{R}_t[2\cdot k])$. Since we have the following inequality:

$$\mathcal{R}_t[2\cdot k+1] = \max_{i=(2\cdot k-1)\cdot\lambda}^{(2\cdot k+1)\cdot\lambda-1}\mathcal{A}[i] \leqslant \sum_{i=(2\cdot k-1)\cdot\lambda}^{(2\cdot k+1)\cdot\lambda-1}\mathcal{A}[i],$$

using the linearity of expectation, we have

$$E(\mathcal{R}_t[2\cdot k+1]) \leqslant \sum_{i=(2\cdot k-1)\cdot\lambda}^{(2\cdot k+1)\cdot\lambda-1} E(\mathcal{A}[i]) = \frac{2\cdot\lambda\cdot d\cdot n}{w_o}. \quad (4)$$

And Because $\mathcal{R}_t[2\cdot k]$ is the maximum value of the smallest t counters in the $2\cdot\lambda$ counters, we can know that $(2\cdot\lambda - t + 1)$ of



(a) ARE *v.s.* $\mathcal{M}_b$ on parameter settings. (b) ARE *v.s.* $\mathcal{M}_o$ on parameter settings. (c) ARE *v.s.* $\mathcal{M}_o$ on single data stream. (d) Exact key frequency estimation.

Figure 12: Experiments on CU.

$2 \cdot \lambda$ counters are larger than $\mathcal{R}_t[2 \cdot k]$. Then using the pigeonhole principle, we can get

$$\mathcal{R}_t[2 \cdot k] \leqslant \frac{1}{2 \cdot \lambda - t + 1} \cdot \left(\sum_{i=(2 \cdot k - 2) \cdot \lambda}^{(2 \cdot k) \cdot \lambda - 1} \mathcal{A}[i] \right).$$

Again, using the linearity of expectation, we have

$$E\left(\mathcal{R}_t[2 \cdot k]\right) \leqslant \frac{\left(\sum_{i=(2 \cdot k-2) \cdot \lambda}^{(2 \cdot k) \cdot \lambda - 1} E(\mathcal{A}[i])\right)}{2 \cdot \lambda - t + 1} = \frac{2 \cdot \lambda \cdot d \cdot n}{(2 \cdot \lambda - t + 1) \cdot w_o}. \tag{5}$$

Substituting Formula 4 and 5 into Formula 3, we can get

$$E\left(\sum_{i=0}^{w_o - 1} \mathcal{R}_t[\mathcal{I}_t'[i]]\right) \leqslant \begin{array}{l} \left(w_o - \frac{w_o}{2 \cdot \lambda} \cdot t\right) \cdot \left(\frac{2 \cdot \lambda \cdot d \cdot n}{w_o}\right) \\ + \left(\frac{w_o}{2 \cdot \lambda} \cdot t\right) \cdot \left(\frac{2 \cdot \lambda \cdot d \cdot n}{(2 \cdot \lambda - t + 1) \cdot w_o}\right) \end{array}$$
$$= \left((2 \cdot \lambda - t + 1) + \left(\frac{2 \cdot \lambda + 1}{2 \cdot \lambda - t + 1}\right) - 2\right) \cdot (d \cdot n). \tag{6}$$

Then we can substitute Formula 6 into Formula 1, and we get

$$E\left(\sum_{i=0}^{w_o - 1} \left(\mathcal{R}_t[\mathcal{I}_t'[i]] - \mathcal{A}[i]\right)\right)$$
$$\leqslant \left((2 \cdot \lambda - t + 1) + \left(\frac{2 \cdot \lambda + 1}{2 \cdot \lambda - t + 1}\right) - 3\right) \cdot (d \cdot n).$$

□

C DATASETS, PLATFORM, AND METRICS

Datasets: For experiments on data streams, we use the anonymized IP trace dataset collect in Equinix-Chicago monitor from CAIDA [26], which contains about 2.7×10^7 items with 1.3×10^6 distinct IDs in total. We also use two other real datasets IMC DC [27] and webdocs [28], and two synthetic datasets following the Zipf distribution (Zipf_0.3 and Zipf_2.1, where 0.3 and 2.1 are the skewness), which are generated by Web Polygraph [29]. For experiments on machine learning, we use the Twin gas sensor arrays Data Set download from UCI Machine Learning Repository [30], which contains 640 instances and 10^5 features.

Platform: We conduct the experiments on a 18-core CPU server (Intel i9-10980XE) with 128GB memory and 24.75MB L3 cache.

Evaluation Metrics:

- **Average Relative Error (ARE):** $\frac{1}{m} \sum_{i=1}^{m} \frac{|f_i - \hat{f}_i|}{f_i}$.
  Where f_i and $\hat{f}_i$ are the real and estimated frequency of item e_i, respectively, and m is the number of distinct items.

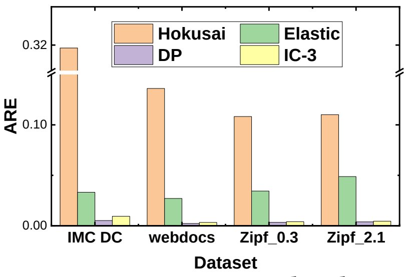


Figure 13: Experiments on other datasets.

- **Average Absolute Error (AAE):** $\frac{1}{m} \sum_{i=1}^{m} |f_i - \hat{f}_i|$.
- **Compression Error:** $\frac{1}{w_o} \sum_{i=0}^{w_o - 1} |\mathcal{R}[\mathcal{I}'[i]] - \mathcal{A}[i]|$.
  Where w_o is the width of the original counter array, $\mathcal{A}[i]$ is the i-th counter, and $\mathcal{R}[\mathcal{I}'[i]]$ is the corresponding representative.
- **Loss:** For the logistic regression, the loss function uses cross entropy. For the linear regression, the loss function uses L2-norm.

D EXPERIMENTS ON THE CU SKETCH

Accuracy *v.s.* $\mathcal{M}_b$ on Parameter Settings (Figure 12(a)): The experimental results show that, the performance of Cluster-Reduce on CU is similar to that of CM shown in Section 7.2. When $\lambda = 8$, the ARE of DP and IC-3 on CU are about 3.46 and 8.24 times lower than that of IC-1, respectively.

Accuracy *v.s.* $\mathcal{M}_o$ on Parameter Settings (Figure 12(b)): When $\lambda = 8$, the ARE of DP and IC-3 on CU are about 5.81 and 5.99 times lower than that of IC-1, respectively.

Accuracy *v.s.* $\mathcal{M}_o$ on Single Data Stream (Figure 12(c)): We find that, when $\lambda = 8$, the ARE of DP and IC-3 on CU are about 15.0/41.6 and 15.5/42.8 times lower than that of Elastic/Hokusai, respectively.

Exact Key Frequency Estimation (Figure 12(d)): We find that, when $\mathcal{M}_b \approx$ 1MB, the ARE of DP and IC-3 on CU are about 5.88/16.1 and 3.07/8.41 times lower than that of Elastic/Hokusai, respectively.

E EXPERIMENTS ON OTHER DATASETS

Accuracy *v.s.* $\mathcal{M}_o$ on Single Data Stream (Figure 13): The experimental results show that, the performance of Cluster-Reduce on other datasets is similar to the results shown in Section 7.3. When limiting $\mathcal{M}_b$, the ARE of DP on IMC DC, webdocs, Zipf_0.3, and Zipf_2.1 are about 6.33/60.5, 11.9/59.6, 10.2/32.1, and 12.4/28.1 times lower than that of Elastic/Hokusai, respectively; the ARE of IC-3 on the four datasets are about 3.51/33.5, 8.13/40.9, 8.49/26.7, and 10.6/23.9 times lower, respectively.