A Cloud-Edge Collaborative Framework for Distributed Triangle Counting on Graph Stream

Ruilin Hu[†], Chao Song[†]*, Jie Wu[‡] and Li Lu[†]
[†]School of Computer Science and Engineering, University of Electronic Science and Technology of China, China
[‡]Department of Computer and Information Sciences, Temple University, US
Email: huruilin@std.uestc.edu.cn, {chaosong, luli2009}@uestc.edu.cn, jiewu@temple.edu

***Abstract*—Graph computing in cloud-edge collaborative environments faces critical challenges in distributed task processing, particularly in fundamental operations such as subgraph isomorphism that underpins triangle counting applications. In typical architectures where data streams are transmitted from edge collectors to cloud masters, conventional approaches employ reservoir sampling to distribute edge streams among workers for triangle estimation. However, the computational accuracy degradation is caused by cross-domain edge distribution strategies. In this paper, we propose a cloud-edge collaborative framework for distributed triangle counting. We employ spectral clustering analysis to reveal latent domain relationships that guide edges distribution. Our experimental evaluation uses streaming data with global relative error measurement across multiple datasets, demonstrating superior performance over existing algorithms.**

***Index Terms*—cloud-edge collaborative, graph computation, triangle counting, graph stream**

I. Introduction

Triangle counting is a fundamental algorithmic task in graph computing frameworks. It aims to identify and quantify triangular subgraphs, which is critical for analyzing network cohesion and clustering properties. The research of triangle counting within networks is deeply rooted in the broader landscape of network science, encompassing diverse domains such as social networks, communication, the Internet of Things, and information sciences. Despite the structural diversity among networks in these domains, triangles appear as a universal topological structure, offering rich senses into network dynamics. The presence of triangles in real-life networks has spurred the development of metrics like the clustering coefficient [1] and the transitivity ratio [2] to characterize and analyze networks effectively. These metrics provide measures of the level of clustering or cohesion within networks, stressing the importance of triangles in understanding network structure and function. In social networks specifically, the existence of triangles has been studied and explained through various social science theories such as homophily and transitivity. These theories explain the propensity for individuals to form connections with others who share similar attributes or relationships, thus giving rise to triangular relationships within social networks. The computational task of counting triangles in social networks is essential for numerous applications, including the computation of transitivity ratios and clustering coefficients. These metrics serve as key indicators for network

* Chao Song is the corresponding author.

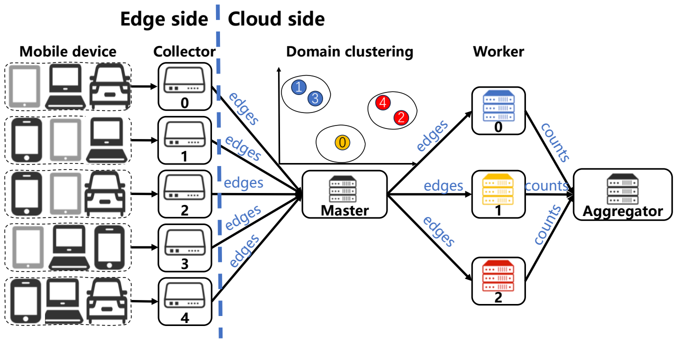


Fig. 1. Overview of cloud-edge collaborative framework for distributed triangle counting.

analysis and evolution model [3], providing valuable insights into network cohesion and dynamics.

With the popularization of the Internet of Things and smart terminals, a large amount of graph data has emerged in real-world scenarios, and there is an urgent need to build an efficient processing architecture. Based on the collaborative characteristics of edge computing and cloud computing, researchers uses edge collectors to achieve distributed data collection and preprocessing, relies on the powerful computing power of cloud servers to complete complex computing tasks. In the cloud-edge collaborative scenario, multiple mobile devices are deployed across distinct geographical domains. Each collector gathers information on the graph edges[1] connecting devices within each domain. Notably, the source nodes of these edges are identical, effectively resulting in numerous wedges.

In this paper, we propose a Cloud-edge collaborative Framework for Distributed triangle Counting (CFDC) as shown in Fig. 1. In the example of a mobile social network, the multiple mobile devices are distributed in various geographic domains, and each collector collects the edge streams from the devices in each domain. In the mobile social network, each node represents a mobile device, and each edge represents a communication link from a source device to a destination device. The edges from the source devices at a domain are transmitted to the master by its connected collector. The master built a domain-to-domain adjacency matrix for clustering. According to the results, the master assigns the edges from the different domains to different workers, thereby enabling the

[1]The term “edge” collectively refers to graph edges.

distributed counting of triangles in this mobile social network.

Therefore, the following three challenges are faced: (1) Previous research adopted a Master-Worker-Aggregator architecture, in which the master collected a large number of edges and sent them to workers. (2) How to distribute edges to maximize triangles formed within each worker. (3) Given the dynamic edge streams, how can their correlations be detected and the distribution strategy be updated in real-time?

To address these challenges, our contributions are as follows: (1) We transform the edge distribution task into assigning domain IDs, which reduces the hash space and reduces the master's distribution workload. (2) We perform domain clustering to ensure that more edges forming triangles are distributed to the same worker. (3) At each graph snapshot, we build an adjacency matrix to adjust our distribution strategy to the dynamic of the streaming edge.

II. Related Work

Streaming triangle counting. Much effort has been given to developing centralized streaming algorithms for triangle counting in a large graph stream. Bar-Yossef et al. [4] first designed streaming algorithms to count triangles in a stream using the reduction paradigm alone. Subsequently, the algorithm's space bounds were improved by Tsourakakis et al. [5], who presented a graph sparsification method by sampling each edge with equal probability for estimating the count of global triangles in the sampled graph. Additionally, Ahmed et al. [6, 7], proposed a method of sampling edges with different probabilities, which is a more general edge sampling framework. Recently, Shin et al. [8] proposed ThinkD, an algorithm that accurately estimates the counts of global triangles in a fully dynamic graph stream with additions and deletions of edges. Gou et al. [9] proposed SWTC, which can solve unbiased sampling and cardinality estimation issues with bounded memory usage. Liu et al. [10] use an asynchronous callback function for in-storage graphs with transparent acceleration to improve cost and power efficiency in triangle counting. Kang et al. [11] based on the newly proposed Edge-RLDP privacy notion, is a privacy-preserved federated estimator for triangle count with three perturbation algorithms that enhances estimation accuracy and achieves (ϵ, δ)-Edge-RLDP.

Distributed algorithms of triangle counting. Many distributed algorithms were developed to relieve the computing pressure of a single machine. These include distributed-memory [12] and MapReduce [13]. However, these distributed algorithms do not apply to the real-time counting of triangles in large-scale dynamic graphs since they require all the edges to be input at once. Recently, numerous studies have been conducted on distributed streaming algorithms for global triangle counting. Shin et al. proposed Tri-Fly [14], which parallels TRIÈST-IMPR [15] directly by distributing edges to all workers. They later developed CoCoS [16] based on Tri-Fly, improved the accuracy significantly. Finally, Yang et al. [17] proposed a distributed streaming framework based on the Master-Worker-Aggregator architecture.

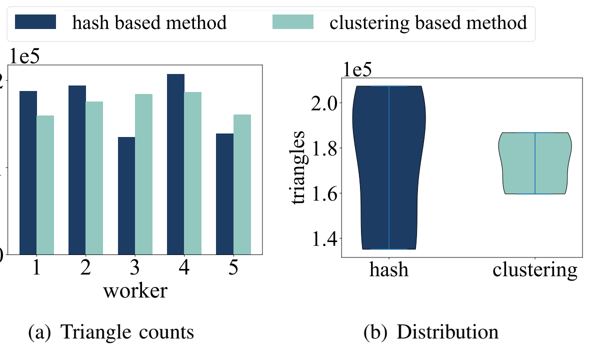


Fig. 2. Distribution of the triangle counts in different workers under the methods of hash and clustering

TABLE I
FREQUENTLY-USED SYMBOLS

Symbol	Definition
$\Pi = (e^{(1)}, e^{(2)}, \cdots)$	graph stream of edges
$e^{(t)}$	edge arrives at time t
$G^{(t)}$	the graph at time t
n	the numbers of input edges
k	the size of the reservoir
$\Delta^{(t)}$	the triangle counts in the graph
$\lvert W \rvert$	the number of workers
$\hat{\tau}$	the estimate of the global triangles count
$cluster\,(e) = j$	cluster function for mapping edge e to W_i
c	the number of the clusters

III. Preliminary and Problem

A. Motivation

Streaming triangle counting is an approximate rather than an exact result. Reservoir sampling includes a sampling probability. This indicates that the more triangles sampled, the more accurate the estimated value will be when recovered using the sampling probability. Traditional methods use hashes to distribute triangles, resulting in an uneven distribution. Fig. 2 shows the distribution of triangles through hashing and the clustering of triangles among each of the five workers in the Email-Enron dataset. Fig. 2(a) illustrates the distribution of triangles to various workers using different strategies. Fig. 2(b) uses a violin plot to demonstrate this phenomenon, indicating that the clustering method leads to a more concentrated distribution of triangles. Our objective was to achieve an even distribution of triangles among workers to improve the accuracy of reservoir sampling.

B. Problem Definition

We address the challenge of estimating the counts of global triangles in a graph stream, denoting a sequence of edges, and employing multiple machines with constrained storage capacities. We consider the following conditions: C1 **Lack of prior knowledge**: There is a lack of information regarding the input graph stream, including details such as the number of edges, degree distribution, etc. These factors cannot be known in advance. C2 **Limited storage capacity**: Each of the $|W|$ workers can store a maximum of $k(\geq 2)$ edges. However, the number of edges in the input graph stream may exceed

k and even $|W| \times k$. C3 **One-pass processing**: Edges are processed sequentially in the order of arrival. Past edges cannot be accessed unless stored in the limited storage described.

Under these conditions, we define the problem of distributed estimation of global triangle counts in a graph stream.

Problem 1: Distributed estimation of triangle counts in a graph stream. **Given**: A graph stream $(e^{(1)}, e^{(2)}, \dots)$, and n distributed stores, which can store up to $k(\geq 2)$ edges. **Maintain:** The estimation errors of global triangle count $|\Delta^{(t)}|$ for each time $t \in 1, 2, \cdots$. **To Minimize**: The biases and variances of the estimates.

C. Reservoir Sampling Algorithm

When addressing the problem of triangle counting in large-scale graph data, traditional methods often encounter memory bottlenecks. In contrast, the reservoir sampling algorithm, with its sublinear space complexity, stores only a subset of edges in the streaming graph, thereby reducing memory consumption while ensuring sampling uniformity. This characteristic allows it to efficiently maintain a dynamically updated edge set, providing a reliable data foundation for subsequent triangle counting. The reservoir sampling algorithm is sublinear in terms of space complexity, reducing computational memory usage while ensuring the uniformity and accuracy of the sampled edges. In reservoir sampling, for each input edge n, the probability of including it in the edge sample is k/n. When the $(n+1)^{th}$ edge arrives, the probability of replacing it in the edge sample is $P_i = k/(n+1)$. At the same time, the probability of the previous edge n will be replaced is $P_c = 1/k$. Therefore, the probability of the $(n+1)^{th}$ edge replacing the i^{th} edge is $P_i P_c$, and the probability of the i^{th} edge not being replaced by the $(n+1)^{th}$ edge is $P = 1 - P_i P_c = 1 - (k/(n+1)(1/k)) = 1 - 1/(n+1)$.

The execution strategy can be divided into three steps. Determine whether the reservoir is full and whether to sample. The worker deletes the sampling set when the reservoir is full. Update the counter after deleting edges from the sampling set, as the counter represents the exact counts of triangles that can be formed by the edges in the sampling set. Given the constraints of large-scale graph data processing, where maintaining a full adjacency structure is often impractical, reservoir sampling provides a memory-efficient alternative. By ensuring uniform sampling, it enables an unbiased representation of the evolving edge set, thereby preserving essential structural properties for downstream triangle counting tasks. Moreover, the stepwise execution strategy allows for dynamic adaptation to streaming graph updates, which is critical for accurately estimating triangle counts over time. This makes reservoir sampling a suitable foundation for the proposed solution, as it balances computational efficiency with statistical reliability in large-scale streaming graph environments.

IV. Methodology

For streaming triangle counting in cloud-edge scenario, we propose a Cloud-edge collaborative Framework for Distributed triangle Counting (CFDC) shown in Fig. 1. The collector collects edges from various domains, and at the same time tags the edges with a domain, the master clusters through the received edges, and according to the clustering results, assigns them to different workers, thereby enabling distributed counting of triangles. The aggregator collects and consolidates the estimates received from the workers. An appropriate aggregation technique can determine the estimated values of the global triangle counts within the graph stream.

A. Domain Clustering

An edge represents a connection between two nodes, namely the source and target. While the source pertains to a specific domain, the target may belong to another domain, influenced by factors like geographic correlation. For instance, in a mobile social network, the multiple mobile devices are distributed in various geographic domains, and each collector collects the devices in each domain. In the mobile social network, each node represents a mobile device, and each edge represents a communication link from a source device to a destination device. The edges from the source devices at a domain are transmitted to the master by its connected collector. The master built a domain-to-domain adjacency matrix for clustering. The collector between domains then creates a relationship, which is strong or weak. We use the notation $w_{i,j}$ to denote the flow between domains A and B. An adjacency matrix W is constructed where greater flow corresponds to higher values. Next, we describe how to cluster domains using an adjacency matrix.

Clustering problem: From an intuitive perspective, clustering is grouping similar data points, ensuring high intra-group similarity while minimizing inter-group similarity. The most straightforward approach is to find a partitioning method that minimizes the total weight of edges between different groups. We use the methodological framework of the minimum cut problem to elucidate the relationship between spectral clustering algorithms and subgraph partitioning algorithms. Given a weight graph with weight adjacency W, the most intuitive strategy for constructing graph cuts is to solve a minimum cut problem. Denote the sum of the connection weights between two sets as $W(A,B) = \sum_{i \in A, j \in B} w_{i,j}$. We marked $\bar{A}$ as the complementary set of A. Given a partition of set A into c subsets, the minimum cut problem can be described as: Pick a subgraph division $cut(A_1, \cdots, A_c)$ and ask for the function that minimizes this division: $cut(A_1, \cdots, A_c) = \frac{1}{2}\sum_{i=1}^{c} W(A_i, \bar{A}_i)$. We use a factor of $\frac{1}{2}$ to ensure that each edge is computed only once. In practice, the loss function generally does not give a satisfactory partition, as the solution to this minimum cut tends to divide the entire dataset into a clump and a very isolated point, which is not the clustering result we want. An improvement is to constrain the partition so that $A_1, \cdots, A_c$ are all relatively "large". Commonly used loss functions for this purpose are RatioCut, which is defined as follows. $RatioCut(A_1, \cdots, A_c) = \sum_{i=1}^{c} \frac{cut(A_i, \bar{A}_i)}{|A_i|}$. RatioCut for any number of clusters c. We can write it as c clustering numbers of linear programming relaxation. Given a partition $A_1, \cdots, A_c$ of the set A, we define c indicator vectors $h_j = h_{1,j}, \cdots, h_{n,j})^T$.

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, concerning $h'_c L h_c$, we have:

$$\begin{aligned} h'_c L h_c &= \sum_{i,j} w_{i,j}(h_{i,c} - h_{j,c})^2 \\ &= \sum_{i \in A, j \in \bar{A}} w_{i,j}/|A_c| + \sum_{j \in A, j \in \bar{A}} w_{i,j}/|A_c| \\ &= 2 * cut(A_c, \bar{A}_c)/|A_c| \end{aligned} \tag{1}$$

According to $h'_i L h_i = (H'LH)_{ii}$, the original equation $\min_{A_1, \cdots, A_c (c \in A)} \text{RatioCut}(A_1, \cdots, A_c)$. Substituting the above equation, we obtain:

$$\min_{A_1, \cdots, A_c} \textbf{Trace}(H'LH), s.t. H'H = I \tag{2}$$

Here, $\textbf{Trace}(\cdot)$ denotes the trace of a matrix. We can conclude that the solution to linear programming relaxation at this point is $H = (u_1, \cdots, u_c)$, which represents the first c smallest characteristic root of the Laplacian matrix L.

From the perspective of graph partitioning methods, the objective converges on the eigen decomposition of the symmetric matrix L_{sym}. We begin by constructing the adjacency matrix W, which encodes the pairwise similarity among the domains. The degree matrix D, in turn, quantifies the structural relationships between a given domain and the others, capturing the number of domains to which it is connected. The domain graph Laplacian is then formulated as $L = D - W$ Notably, the diagonal entries of W encapsulate self-similarity information, where similarity is defined in terms of edges from one domain to another. In contrast, the degree matrix encapsulates global relational information. Since this connectivity information is intrinsic to the domain itself, and given that the degree matrix is diagonal, incorporating it into the diagonal of W to construct the graph Laplacian.

B. Algorithm: CFDC

The algorithm adopts a clustering function $cluster(e^{(t)}_{u,v}) : E \rightarrow M$ to map edges to workers directly with a fixed probability $P = 1/m (m \geq \sqrt{|W|})$. Each worker executes the algorithm TRIÈST-IMPR and sends the total counts of global triangles observed to the aggregator, which counts the final estimations. We introduce the CFDC algorithm in detail.

Collector (lines 4-5): The collector collects the edges of each domain and tags each edge with the domain ID.

Master (lines 6-9): As shown in Algorithm 1, the master unicasts edges and clustering label directly to all workers without any judgment. It ensures that every edge participates in the counter-update algorithm of each worker.

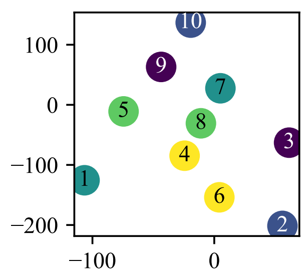
Worker (lines 10-15): Each worker maintains a global triangle counter $\tau^{(t)}_i$ indicating the contributions of every triangle Δ. Thus, there are two edges $e^{(t)}_{u,v}$ and $e^{(t)}_{u,w}$ stored in $S^{(t)}_i$ when $e^{(t)}_{v,w}$ is observed in Π. Here, we abbreviate the triangle Δ. The contribution of a triangle Δ is $\eta^{(t)}_{f(\Delta)} = \min(1, \frac{(k-1)(k-2)}{(l^{(t)}_{i-1})(l^{(t)}_{i-2})})$. Once the worker W_i receives the edge $e^{(t)}_{u,v}$ from the master, it unconditionally calls on UpdateCounter ($e^{(t)}_{u,v}$) directly to increase $\tau^{(t)}_i, \tau^{(t)}_i(w)(w \in \{u, v\} \cup N^{S^{(t)}_i}_{u,v})$ by $|N^{S^{(t)}_i}_u \cap N^{S^{(t)}_i}_v| \cdot \eta^{(t)}_i$ (line 8). Then, $cluster(e^{(t)}_{u,v})$ determines whether to sample based on the clustering label, and subsequently sends the counting result to the aggregator.

Algorithm 1: CFDC

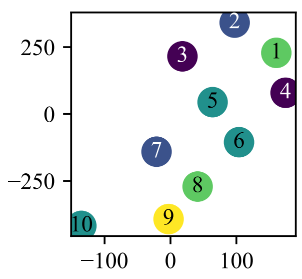
```
1  Input: $\Pi = (e^{(1)}, e^{(2)}, \cdots)$
2  Output: estimate of the global triangles count $\hat{\tau}^{(t)}$
3  for each day do
4      Collecter:
5      label the edge with domain ID, and send edges to
         the master
6      Master:
7      while $e^{(t)} \in \Pi$ do
8          distribute $e^{(t)}$ to workers by cluster lable
9          Get the next day's clustering labels by eq. 2
10     Worker:
11     for $e^{(t)}$ from master to $range(e^{(t)})$ do
12         $\tau^j_i \leftarrow$ UpdateCounter($e^{(t)}$)
13         if ReservoirSample($e^{(t)}$) then
14             UpdateReservoir($e^{(t)}, p$)
15         send $\tau^j_i$ and the edge stream to the Aggregator
16     Aggregator:
17     $\hat{\tau} \leftarrow 0$;
18     calculate $\hat{\tau}$ by $\hat{\tau} = \sum_i^{|W|} \hat{\tau}_i$
```

Aggregator (lines 16-18): The aggregator aggregates the estimates of workers and calculates the probability that a triangle in $G^{(t)}$ can be counted to find the final estimations $\hat{\tau}^{(t)}$ and $\hat{\tau}^{(t)}(u)$. We use a cluster lable $cluster(e^{(t)}_{u,v}) : E \rightarrow M$ to map edges to workers, where $M\_cluster = \{1, 2, \cdots, k\}$, Therefore, the probability of mapping $e^{(t)}_{u,v}$ to W_i is $P_{e^{(t)}_{u,v}, i} = P[cluster(e^{(t)}_{u,v}) = i] = \frac{1}{k}$. We use e_1 to denote $e^{(t)}_{u,v}$ and e_2 to denote $e^{(t)}_{u,w}$, thus, the probability of mapping the two edges e_1 and e_2 to the same worker (i.e., W_i) is $P_{e_1,i} \cdot P_{e_2,i} = \frac{1}{k}$. Therefore, the probability that a triangle in $G^{(t)}_i$ can be counted in a single worker, and the aggregator calculates the global triangle counts $\hat{\tau}^{(t)}$ as $\hat{\tau}^{(t)} = \sum_i^{|W|} \hat{\tau}^{(t)}_i$.

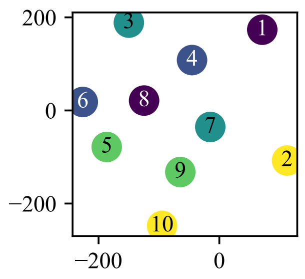
For example, a triangle $\Delta(a, b, c)$ consists of three edges: (a, b), (a, c), and (b, c). When using a hash-based edge distribution approach, these three edges can be sent to different workers with some probability. In contrast, under the clustering method, the edges forming a triangle are sent to the same domain, such that (a, b) and (a, c) belong to one domain, while (a, c) and (b, c) belong to another. The entire triangle $\Delta(a, b, c)$ can be counted within the same worker by merging these two domains into a single cluster.



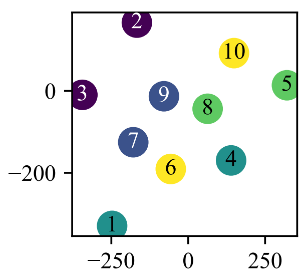
(a) Email-Enron



(b) DBLP

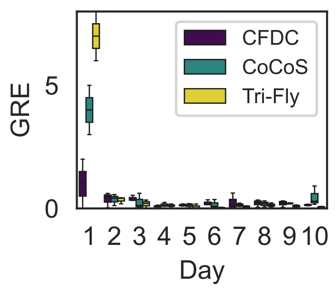


(c) Gowalla

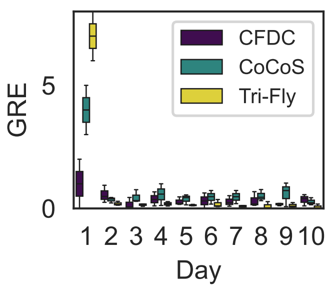


(d) YouTube

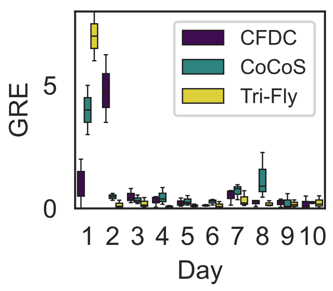
Fig. 3. Results of domain clustering in feature space with different datasets.



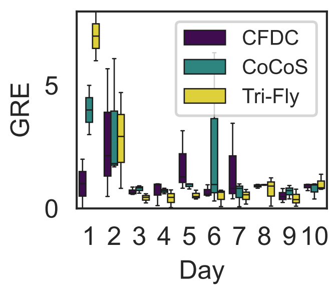
(a) Email-Enron



(b) DBLP



(c) Gowalla



(d) YouTube

Fig. 4. Global relative error of triangle counts per day with different datasets.

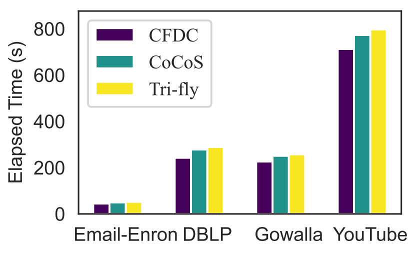


Fig. 5. Elapsed time of triangle counts with different datasets.

V. Experiment

This section presents experiments to analyze accuracy and clustering effects. All experiments were conducted on a machine with 24 CPUs x Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz and 128GB RAM.

A. Datasets

In our evaluation, we used real-world graph datasets to assess the effectiveness of our algorithms. The datasets are publicly available and sourced from the SNAP4[2] (Stanford Large Dataset Network Collection). These datasets represent a diverse range of network structures and characteristics. To ensure consistency in our analysis, we exclude self-loops and consider the edges as undirected. This simplification allows us to focus on the fundamental properties of graph structures and evaluate the performance of our algorithms in a standardized manner. By using these public datasets, we ensure the reliability and reproducibility of our findings. Since there is no time information in these datasets, we generate timestamps using a Poisson distribution for the probability of different edges arriving at the collector.

[2]https://snap.stanford.edu/data/index.html

B. Baseline

We consider two streaming algorithms for triangle counting in streaming graphs as competitors. **Tri-Fly [14]**: Tri-Fly is a simple distributed TRIÈST-IMPR replication using broadcast edge stream. Each worker runs the TRIÈST-IMPR algorithm, and the aggregator averages the counts. **CoCoS [16]**: The master distribution strategy optimizes the "Lucky" approach to reduce communication overhead, while workers reduce their sampling range and facilitate redundancy between workers.

C. Evaluation Metrics

We consider the following metrics to evaluate the accuracy and distributed performance. **Global Relative Error (GRE) [14]**: The GRE metric quantifies the accuracy of the estimated global triangle counts compared to the ground truth values. It provides a measure of how closely the estimated value, denoted $\tau^{(t)}$, aligns with the true value, denoted $\tau^{(t)}$. The Global Relative Error is defined as follows: GRE $= \frac{|\hat{\tau}^{(t)} - \tau^{(t)}|}{\tau}$. **Elapsed Time**: When multiple CPUs process tasks simultaneously, the CPU time will be greater than the elapsed time. In this paper, elapsed time refers to the total running time of the entire distributed architecture.

D. Performance of CFDC

1) Clustering Effect: As shown in Fig. 3 axes are the two-dimensional representations of the original high-dimensional data, created to preserve the structure and relationships between data points in a simplified form. Clustering performed well across multiple domains on four different datasets. This suggests that the clustering algorithm is effective in distinguishing and categorizing data across various domains. The phenomenon implies that the clustering algorithm possesses a certain level of generalization capability, allowing it to adapt to different types and structures of data.

The results of this experiment indicate that the clustering algorithm performs well across diverse domains of data. The cross-domain clustering efficacy reflect some universal similarities or structures among the data, enabling the algorithm to capture the underlying patterns and relationships.

2) Accuracy: Fig. 4 presents a comparative analysis of the accuracy of triangle counting across various datasets and over time. The figure is composed of multiple subfigures, each corresponding to a different dataset, namely Email-Enron, DBLP, Gowalla, and YouTube. The subfigures show the Global Relative Error (GRE) of triangle counts daily, indicating how the accuracy of the triangle counting evolves with each passing day. The CFDC algorithm provides a lower global relative error across various datasets, demonstrating its superiority in the task of distributed triangle counting. These results support the effectiveness of the algorithm proposed in the paper and provide a basis for its application in similar scenarios in the future.

3) Elapsed Time: The results in Fig. 5 show that the CFDC method outperforms both CoCoS and Tri-Fly in terms of elapsed time. This indicates that CFDC computes triangle counts in graph streams more efficiently. Moreover, the lower elapsed times exhibited by CFDC suggest that it is better suited for processing large-scale graph data, which is critical for applications requiring real-time analytics.

VI. Conlusion

We propose a cloud-edge collaborative framework in distributed triangle counting in graph streams. We employ spectral clustering analysis to reveal latent domain relationships that guide edges distribution. The approach's superiority over existing algorithms in terms of counting accuracy and elapsed time has been substantiated through experiments on various datasets.

VII. Acknowledgments

The work of UESTC team was supported by the National Natural Science Foundation of China under Grant No. 62472070, 82241060, 62020106013; the Sichuan Science and Technology Program No.2024NSFSC0492; the National Key R&D Program of China under Grant 2021YFB3101303 and 2021YFB3101302; the Full-Time Postdoctoral Research and Development Fund of West China Hospital of Sichuan University No. 2023HXBH129.

References

[1] C. Grabow, S. Grosskinsky, J. Kurths, and M. Timme, "Collective relaxation dynamics of small-world networks," *CoRR*, vol. abs/1507.04624, 2015.

[2] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949.

[3] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–36, 2014.

[4] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, "Reductions in streaming algorithms, with an application to counting triangles in graphs," in *Proc. of ACM SIAM*, 2002.

[5] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "DOULION: counting triangles in massive graphs with a coin," in *Proc. of ACM SIGKDD*, 2009.

[6] N. K. Ahmed, N. G. Duffield, J. Neville, and R. R. Kompella, "Graph sample and hold: a framework for big-graph analytics," in *Proc. of 20th ACM SIGKDD*, 2014.

[7] N. K. Ahmed, N. G. Duffield, T. L. Willke, and R. A. Rossi, "On sampling from massive graph streams," *Proc. of VLDB Endow.*, 2017.

[8] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos, "Fast, accurate and provable triangle counting in fully dynamic graph streams," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 2, pp. 12:1–12:39, 2020.

[9] X. Gou and L. Zou, "Sliding window-based approximate triangle counting over streaming graphs with duplicate edges," in *Proc. of ACM SIGMOD*, 2021.

[10] Y. Liu, T. Wang, Y. Liu, H. Chen, and C. Li, "Edge-protected triangle count estimation under relationship local differential privacy," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 10, pp. 5138–5152, 2024.

[11] S. Kang and S. Jun, "Sting: Near-storage accelerator framework for scalable triangle counting and beyond," in *Proc. of DAC*, 2024.

[12] S. Arifuzzaman, M. Khan, and M. V. Marathe, "PATRIC: a parallel algorithm for counting triangles in massive networks," in *Proc. of ACM CIKM*, 2013.

[13] H. Park and C. Chung, "An efficient mapreduce algorithm for counting triangles in a very large graph," in *Proc. of 22nd ACM CIKM*, 2013.

[14] K. Shin, M. Hammoud, E. Lee, and C. Faloutsos, "Tri-fly: Distributed estimation of global and local triangle counts in graph streams," in *Proc. of PAKDD*, 2018.

[15] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, "Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size," *ACM Trans. Knowl. Discov. Data*, pp. 43:1–43:50, 2017.

[16] K. Shin, E. Lee, J. Oh, M. Hammoud, and C. Faloutsos, "Cocos: Fast and accurate distributed triangle counting in graph streams," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 3, pp. 38:1–38:30, 2021.

[17] X. Yang, C. Song, M. Yu, J. Gu, and M. Liu, "Distributed triangle approximately counting algorithms in simple graph stream," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 4, pp. 79:1–79:43, 2022.