Towards Efficient Graph Processing in Geo-Distributed Data Centers

Feng Yao, Qian Tao, Shengyuan Lin, Yanfeng Zhang, Wenyuan Yu, Shufeng Gong, *Associate Member, IEEE*, Qiange Wang, Ge Yu, *Senior Member, IEEE*, and Jingren Zhou, *Fellow, IEEE*

***Abstract*—Iterative graph processing is widely used as a significant paradigm for large-scale data analysis. In many global businesses of multinational enterprises, graph-structure data is usually geographically distributed in different regions to support low-latency services. Geo-distributed graph processing suffers from the Wide Area Networks (WANs) with scarce and heterogeneous bandwidth, thus essentially differs from traditional distributed graph processing. In this paper, we propose RAGraph, a *Region-Aware framework for geo-distributed graph processing*. At the core of RAGraph, we design a region-aware graph processing framework that allows advancing inefficient global updates locally and enables sensible coordination-free message interactions and flexible replaceable communication module. In terms of graph data preprocessing, RAGraph introduces a contribution-driven edge migration algorithm to effectively utilize network resources. RAGraph also contains an adaptive hierarchical message interaction engine to switch interaction modes adaptively based on network heterogeneity and fluctuation, and a discrepancy-aware message filtering strategy to filter important messages. Experimental results show that RAGraph can achieve an average speedup of 9.7× (up to 98×) and an average WAN cost reduction of 78.5% (up to 97.3%) compared with state-of-the-art systems.**

***Index Terms*—Graph processing, geo-distributed data centers, heterogeneous network.**

Received 20 March 2024; revised 31 July 2024; accepted 21 August 2024. Date of publication 3 September 2024; date of current version 24 September 2024. This work was supported in part by the National Natural Science Foundation of China under Grant U2241212, Grant 62072082, Grant 62202088, and Grant 62137001, in part by the 111 Project under Grant B16009, in part by the Joint Funds of Natural Science Foundation of Liaoning Province under Grant 2023-MSBA-078, in part by the Fundamental Research Funds for the Central Universities under Grant N2216015 and Grant N2416011, and a research grant from Alibaba Innovative Research (AIR) Program. Recommended for acceptance by P. D'Ambra. *(Corresponding authors: Yanfeng Zhang; Shufeng Gong.)*

Feng Yao, Shengyuan Lin, Shufeng Gong, Qiange Wang, and Ge Yu are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China (e-mail: yaofeng@stumail.neu.edu.cn; linshengyuan@stumail.neu.edu.cn; wangqiange@stumail.neu.edu.cn; gongsf@mail.neu.edu.cn; yuge@mail.neu.edu.cn).

Qian Tao, Wenyuan Yu, and Jingren Zhou are with Alibaba Group, Hangzhou 311121, China (e-mail: qian.tao@alibaba-inc.com; wenyuan.ywy@alibaba-inc.com; jingren.zhou@alibaba-inc.com).

Yanfeng Zhang is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China, also with the National Frontiers Science Center for Industrial Intelligence and Systems Optimization, Northeastern University, Shenyang 110819, China, and also with the Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang 110819, China (e-mail: zhangyf@mail.neu.edu.cn).

Digital Object Identifier 10.1109/TPDS.2024.3453872

I. Introduction

Iterative graph processing has emerged as a significant paradigm in many fields. With the rapid growth in the size of graph-structured data, there has been a surge in research efforts aimed at extending graph processing to distributed environments to facilitate computation over large-scale graphs. These studies cover multiple directions, including graph partitioning [1], [2], the design of processing models [3], [4], [5], and the development of parallel algorithms [6], [7].

Unfortunately, most existing frameworks assume that the graph-structured data is distributed to multiple machines within a single-site data center, equipped with high network bandwidth and homogeneous communication links. While in real-world application scenarios, geographically distributing the graph-structured data across multiple data centers is often necessary due to various constraints. A typical example is managing global-scale social networks across countries. Facebook has established over 20 data centers located in Europe, Asia, and America, and almost 90% of its daily active users are outside North America [8]. Another common application with geographically distributed graphs is federated graph computation [9], in which multiple data owners share partial access permission to their local graphs stored in private data centers and collaboratively execute graph analytics on the data union. In such *geo-distributed* applications, multiple data centers are connected by Wide Area Networks (WANs), which results in scarce and heterogeneous network bandwidth [10], [11]. Regulatory and privacy concerns [12] may also prohibit aggregating graph data to a central site. Therefore, traditional distributed graph processing designed with uniform scheduling strategies is no longer effective.

We summarize two essential challenges that lead to the inefficiencies in geo-distributed iterative graph processing through an illustrative example.

Example 1: Fig. 1(a) illustrates the network topology of a AliCloud ECS geo-distributed cluster. The cluster consists of three geo-distributed data centers D_1, D_2, and D_3, each of which is an 8-node cluster connected by 10 Gbps Ethernet. In contrast, the network bandwidth between data centers can only reach up to 100 Mbps and be heterogeneous due to diverse WAN connections. Moreover, the WAN links are unstable due to *network fluctuation*, which may occur even in a short period. Traditional distributed graph processing systems treat the worker as peer

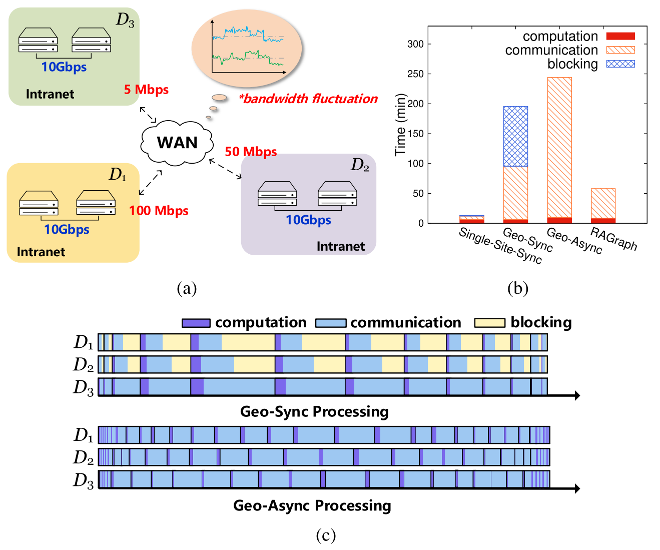


Fig. 1. An example of geo-distributed graph processing. (a) The bandwidth of geo-distributed networks. (b) Performance of geo-distributed and single-site iterative graph processing. (c) Performance breakdown of sync/async parallel processing modes on geo-distributed networks.

workers and assume each pair of them has the same network bandwidth. Based on this, their optimization and scheduling strategies are of a global uniform mindset, which leads to hardly adapting to heterogeneous geo-distributed systems with hierarchy network connections.

To expose the challenges raised by the above issues in iterative graph processing, we run the PageRank algorithm on Twitter graph [13] on two clusters with different configurations. The first cluster consists of 24 AliCloud ECS instances, all located within a single-site data center. These instances are interconnected via a 10Gbps Ethernet network. The second cluster is also equipped with 24 identical ECS instances, deployed in a geo-distributed cluster shown in Fig. 1(a). We test a state-of-the-art synchronous parallel system, GRAPE [6], on both clusters (i.e., Single-Site-Sync and Geo-Sync) and, additionally, an advanced asynchronous parallel processing system, Maiter [14], on the geo-distributed cluster (i.e., Geo-Async). The overall computation, communication, and blocking time are reported in Fig. 1(b). Fig. 1(c) shows the performance breakdown of geo-distributed iterative graph processing under two parallel processing models (sync and async). Compared with processing in a single-site data center, most of the increased running time in geo-distributed data centers is communication and blocking time, with computation time being comparatively insignificant. □

Example 1 reveals two essential challenges of the geo-distributed iterative graph processing due to the hierarchical and heterogeneous networks.

Imbalance of Message Transmission: Message transmission time between data centers is much longer than that within a data center, which results in communication time among data centers occupying most of the execution time, as shown in Fig. 1(b). Therefore, reducing cross-datacenter communication is the key to geo-distributed graph processing. In addition, the imbalance and fluctuation of network transmission between data centers make inefficient utilization of WAN resources on partial transmission links. Worse, successive iterations exacerbate the imbalanced message transmission in iterative graph processing, which is more time-consuming.

Inefficiency of Graph Processing Model: Synchronous graph processing models, e.g., the Bulk Synchronous Parallel (BSP) model [15], require coordinated computation and communication among vertices in each iteration. When it comes to geo-distributed data centers, the bandwidth among which is highly heterogeneous, the barriers will block the messages (i.e., coordinated waiting) in each superstep and dramatically increase the time cost. Back to Fig. 1(c), D_1 and D_2 get stuck in blocking until D_3 finishes communication, thus resulting in a long blocking time of Geo-Sync. Conversely, Geo-Async under Asynchronous Parallel (AP) model [5] allows workers to execute independently to avoid coordinated waiting but incurs frequent communication and high transmission cost. Therefore, both synchronous and asynchronous parallel models are not well-qualified for geo-distributed graph processing.

Among various graph processing systems, Monarch [16] and GeoGraph [17] are designed for geo-distributed graph processing. Monarch reduces WAN usage for synchronous parallel processing by optimizing local computation under the GAS model. GeoGraph reduces communication over the WANs by constructing hierarchical clustering among data centers. Both of them enhance the performance of geo-distributed graph processing tasks, but inevitably coordinate with other workers on the WANs and fail to consider the impact of network fluctuation.

RAGraph: To address these problems, we design and implement a Region-Aware framework for iterative graph algorithms in geo-distributed environments. The framework (1) allows advancing inefficient global updates to local computation to optimize execution time, (2) designs a two-layer coordination-free message interaction view to eliminate coordinated waiting, and (3) mitigates the impact of network congestion by replacing communication roles. The framework implements unified message management through the proxy in combination with the above designs. Regarding data preprocessing for the Region-Aware framework, we establish a vertex contribution metric to characterize the ability of a vertex to generate and propagate influential messages. Subsequently, considering both the contribution metric and network heterogeneity, we introduce an edge migration algorithm to effectively enhance the utilization of scarce network resources. Furthermore, based on the Region-Aware framework, we propose two runtime optimizations, including an adaptive hierarchical message interaction engine and a discrepancy-aware message filtering strategy. The adaptive hierarchical message interaction engine proposes two message interaction ideas of eager/lazy for network heterogeneity, and switches between both modes adaptively by analyzing the communication link status to address the impact of network fluctuation. On the other hand, we develop an adaptive bucket structure in discrepancy-aware message filtering. The range of buckets adaptively adjusts with iterations to filter the important messages of the current phase from the messages generated by different iterations. The above effective components comprise RAGraph, a geo-distributed graph processing system.

A preliminary version of this paper appeared in [18]. This manuscript enhances the framework's completeness and robustness from the perspective of graph data preprocessing. Specifically, we make the following new contributions:

- We introduce a vertex contribution metric to characterize the ability of a vertex to generate and propagate influential messages, which in turn reflects its contribution to convergence.
- We propose a contribution-driven edge migration algorithm that simultaneously considers contribution metric and network heterogeneity to improve the utilization of scarce network resources.
- We integrate the edge migration algorithm into RAGraph and conduct experimental evaluations. Experiments show that the preprocessing step further leads to a 1.23–2.7× speedup and a 14.7%–49.4% reduction in WAN cost over the previous framework performance.

II. Preliminaries

This section reviews the preliminaries for the vertex-centric model and monotonic property of iterative graph algorithms.

Graphs: A *graph* $G = (V, E, C)$ consists of a finite set V of vertices, a set E of directed edges with each $(u, v) \in E$ representing a directed edge from vertex u to vertex v, and a series of functions C which represents the characterizations $C_V^{(i)}(v)$ or $C_E^{(j)}(e)$ owned by vertex v in V or edge e in E.

Vertex-Centric Model: In vertex-centric graph processing model [19], a program $\mathcal{P}$ is executed iteratively on the input graph G for each vertex v, say $\mathcal{P}_v$, and interacts with the programs on v's neighbors in each iteration until the states of the vertices converge. Formally, $\mathcal{P}$ can be represented by a triple $(\mathcal{A}, \mathcal{U}, \mathcal{I})$ for each vertex, where the aggregation function $\mathcal{A}$ aggregates the messages received from neighbors, update function $\mathcal{U}$ updates the vertex state, and interaction function $\mathcal{I}$ defines how vertices interact. Specifically, For a vertex v at the ith iteration, the program $\mathcal{P}_v$ performs as follows:

$$\begin{aligned} x_v^i &= \mathcal{A}(M_v^{i-1}) \\ s_v^i &= \mathcal{U}(s_v^{i-1}, x_v^i) \\ m_{v,w}^i &= \mathcal{I}(s_v^i, x_v^i, C_E(v,w)) \quad (\forall w \in \mathsf{N}_{out}(v)) \end{aligned} \tag{1}$$

where M_v^{i-1} is defined as the set of messages sent from the vertices pointing to v, i.e., $M_v^{i-1} = \{m_{u,v}^{i-1} \mid e(u,v) \in E\}$. x_v^i is the aggregation result of v obtained by $\mathcal{A}$ and s_v^i denotes the current state of vertex v at round i. Based on the state s_v^{i-1} of the previous round and the aggregation result x_v^i, v updates its state s_v^i by $\mathcal{U}$. Finally, vertex v generates the messages for each out edge (v, w) based on s_v^i, x_v^i, and $C_E(v, w)$, and sends to v's neighbors by $\mathcal{I}$. Here $\mathsf{N}_{out}(v) = \{w \mid e(v, w) \in E\}$.

Monotonic Property: Many iterative graph algorithms in vertex-centric programs exhibit monotonicity. That is, the vertex state varies monotonically until convergence. Compared with the vanilla program in (1), these algorithms natively have identical $\mathcal{A}$ and $\mathcal{U}$, and $\mathcal{I}$ does not take the state as the input, i.e., $\mathcal{I}(x_v^i, C_E(v,w))$. Besides, $\mathcal{A}$ and $\mathcal{I}$ satisfy the *monotonic conditions*.

TABLE I
A List of Graph Algorithms With Monotonic Property

Algo.	$\mathcal{A}$	$\mathcal{I}$	**Algo.**	$\mathcal{A}$	$\mathcal{I}$
PageRank	sum	$d \times x_v / N_v$	**SSSP**	min	$x_v + C_E(v,w)$
Katz metric	sum	$\beta \times x_v$	**CC**	max	x_v
Adsorption	sum	$p_w^{cont} \times x_v \times C_E(v,w)$	**PHP**	sum	$d \times x_v \times C_E(v,w)$ or 0 $(w = source)$
SimRank	sum	$d/(N_v \times N_w)$	**HITS**	sum	$d \times x_v$
Computing Paths in DAG	count	max	**BFS**	min	$x_v + 1$

Monotonic Conditions: $\mathcal{A}, \mathcal{I}$ satisfy monotonic conditions if:

(C1) $\mathcal{A}(X \cup Y) = \mathcal{A}(Y \cup X)$ and $\mathcal{A}(\mathcal{A}(X) \cup Y) = \mathcal{A}(X \cup Y)$
(C2) $\mathcal{I}(\mathcal{A}(X \cup Y)) = \mathcal{A}(\mathcal{I}(X) \cup \mathcal{I}(Y))$

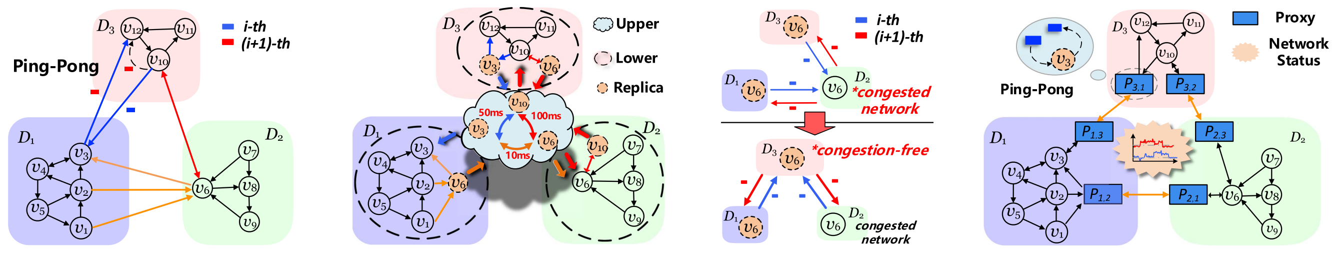
Condition **(C1)** indicates that $\mathcal{A}$ is commutative and associative [14], [20] so that messages can be partially aggregated and updated by $\mathcal{A}$. Condition **(C2)** relaxes the restriction on the composition order of $\mathcal{A}$ and $\mathcal{I}$. In other words, $\mathcal{A}$ can be eliminated from a series of sequential $\mathcal{A}$, $\mathcal{I}$ operations. It also states that the input of $\mathcal{I}$ contains the intermediate result and omits the vertex state [21]. Henceforth, we denote the set of *partial messages* from the element group $*$ as M_*^i and use $\mathcal{A}$ and $\mathcal{U}$ interchangeably.

Table I summarizes some typical iterative graph algorithms that satisfy the monotonic conditions. Conversely, some graph algorithms do not adhere to the monotonic conditions. For example, GCN-Forward [22], whose aggregation function $\mathcal{A}$ is sum, satisfies condition **(C1)**, but the activation functions used as interaction functions, such as ReLU and Sigmoid, do not satisfy condition **(C2)**. In addition, algorithms like Graph Coloring [23] and Triangle Counting [24] require simultaneous access to complete neighbor messages, thereby not meeting the monotonic conditions.

Example 2: We take an iterative graph algorithm, delta-based PageRank [14], as an example of execution in a monotonic fashion, which can be represented as follows:

- $\mathcal{A}(M_{*,v}^{i-1}) = \mathsf{sum}(M_{*,v}^{i-1})$; $\mathcal{A}(s_v^{i-1}, x_v^i) = \mathsf{sum}(s_v^{i-1}, x_v^i)$;
- $\mathcal{I}(x_v^i, C_E(v,w)) = d \times x_v^i / N_v$ $(\forall w \in \mathsf{N}_{out}(v))$.

Here d is a constant damping factor, and N_v denotes the out-degree of a vertex v in graph G. Initially, $s_v^0 = 0$ and $M_v^0 = \{1 - d\}$ for all $v \in V$. When executing, since aggregation function $\mathcal{A}$ (i.e., sum) has commutative and associative properties satisfying condition **(C1)**, the vertex v can gather partial messages from incoming neighbors and use them to update its state monotonically. Then v computes the message for interaction, i.e., $d \times x_v^i / N_v$, by the interaction function $\mathcal{I}$ from the gathered partial messages and propagates the message to outgoing neighbors. It is evident that $\mathcal{A}$ and $\mathcal{I}$ satisfy condition **(C2)**. $\square$



(a) Ping-Pong Effect. (b) Two-layer View. (c) Comm. Strategy. (d) Proxy Design.

Fig. 2. An example of Region-Aware framework. (a) Cross-datacenter ping-pong effect. (b) Two-layer view of local-global interaction. (c) A replacement communication strategy. (d) Region-Aware proxy design.

III. Region-Aware Graph Processing Framework

This section begins with three observations under the *monotonic property* to draw inspiration for RAGraph's framework design, then proposes the design details of the framework.

A. Observations

We start with a toy example and three observations that could help optimize geo-distributed graph processing. In a geo-distributed cluster, as reported in Fig. 1(a), a sample graph is given with 12 vertices distributed, as shown in Fig. 2(a).

Observation 1: Ping-Pong Effect: Consider the execution of PageRank at the blue arrows in Fig. 2(a). For the message initiated from v_{10} transferred via v_3 to v_{12}, v_3 first receives the message $m_{(10,3)}$ from v_{10}, and then generates a message $m_{(3,12)}$ to send to v_{12} with value $d \times m_{(10,3)}/N_{v_3}$, which suffers from inefficient WAN transmission. We call this the *ping-pong effect*. An alternative way is to use message $m_{(10,3)}$ to directly generate $m_{(3,12)}$ in data center D_3 by $d \times m_{(10,3)}/N_{v_3}$, if D_3 has knowledge of the vertex v_3's out-degree N_{v_3}. Consequently, the locally generated messages (e.g., $m_{(3,12)}$) can be used directly for subsequent computations in D_3 without waiting for the cross-datacenter propagation. Afterward, $m_{(10,3)}$ is sent to v_3, as message $m_{(3,12)}$ has been applied from v_{10} to v_{12} without v_3, $m_{(10,3)}$ only interacts via v_3 with other neighbors of v_3 except v_{12}. Since communications between data centers are inefficient, by avoiding the round-trip transmission waiting, this can boost the message passing and result in a shorter running time.

In addition, the ping-pong effect also occurs in other communication-intensive boundary structures. As the red bidirectional arrow in Fig. 2(b) shows, both v_6 and v_{10} can compute locally via the ping-pong effect by using the messages sent to each other. Besides, as the yellow arrows show, D_1 can use messages $m_{(1,6)}$ and $m_{(2,6)}$ from v_1 and v_2 to locally generate the message $m_{(6,3)}$ in advance.

Observation 2: Inefficient Uniform Interaction: Consider the imbalance of network bandwidth between the inside (i.e., local) and outside (i.e., global) of the data center and among data centers. The general approach of applying a uniform interaction pattern suffers from efficiency gaps due to imbalanced networks, e.g., BSP model [15]. Besides, the local-global alternate execution suffers from LAN-WAN bandwidth imbalance, resulting in local message interaction being subject to inefficient global message interaction. This motivated us to layer the message interaction based on the network and connect the layers via vertex *replicas*.

Based on the above considerations, we propose a *two-layer coordination-free* message interaction view, as shown in Fig. 2(b), to eliminate forced global message interaction per iteration and coordination on global. Take PageRank as an example. At the lower layer of the view, a data center, e.g., D_2, computes local PageRank scores s_v via locally generated partial messages, i.e., $d \times \mathsf{sum}(M_{*,v})/N_v$, where $M_{*,v} = \{\cup m_{(u,v)} \mid u \in V_{D_2}\}$, on the subgraph as an independent execution unit without forced global message interaction. The replicas (e.g., v_6 in D_1 and D_3) initiate the lower-upper layer interaction in different data centers only when necessary. At the upper layer of the view, global message interactions are performed via replicas without full attendance to eliminate coordinated waiting and provide fresh global messages, i.e., $M_{*,v} = \{\cup m_{(u,v)} \mid u \in V \setminus V_{D_2}\}$, for lower-layer computations in D_2. Afterward,V_{D_2} can immediately use the received $M_{*,v}$ for local computation without coordination.

Observation 3: Replica Replaceable Communication: In practice, the bandwidth between data centers is allocated based on typical or average usage rather than peak usage [25], resulting in *intermittent network congestion* (*a.k.a.* network fluctuation). When network congestion occurs, the network throughput on the data center link drops, resulting in round-trip of message delays.

Based on the message passing of v_6 in the upper layer in Fig. 2(b) with the original vertex v_6 in D_2 and replicas of v_6 in D_1 and D_3. Consider the communication pattern shown in the upper part of Fig. 2(c), and still take PageRank as an example. The replicas of v_6 send messages to the original v_6 in D_2 at any time. Correspondingly, the original v_6 aggregates part or all of the messages from the replicas and local neighbors, i.e., $x_{v_6} = \mathsf{sum}(M_{*,v_6})$, where $M_{*,v} \subseteq \{m_{(D_1,v_6)} \cup m_{(D_2,v_6)} \cup m_{(D_3,v_6)}\}$, and generates new messages, i.e., $d \times x_{v_6}/N_{v_6}$, to be scattered back to D_1 and D_3. Such *one-to-many* communications result in a large inflow and outflow of messages on the "one" side (i.e., D_2). Transmission delays are intolerable when congestions occur on D_2.

Intuitively, if any replica of v_6 knows the out-degree N_{v_6}, the aggregation and generation of messages can still proceed. So we can find a substitute to share the congested communication. As shown in the lower part of Fig. 2(c), a replica of v_6 in the current congestion-free data center, D_3, is selected as a replacement for the original v_6 in D_2 to handle communications from v_6 in D_1 and D_2.

B. Region-Aware Message Management

The observations in Section III-A inspire us to design a *Region-Aware* message management framework, the structure of which is depicted in Fig. 2(d). Specifically, each data center D_k constructs a proxy $P_{k,l}$ for each remote data center D_l $(k \neq l)$. Each proxy uniformly maintains the corresponding *datacenter-wide replicas*. That is, the proxy is responsible for the global message interaction and replaceable communication at the upper layer and assisting acceleration of the ping-pong effect at the lower layer in the two-layer interaction view, e.g., $P_{3,1}$ generates the message sent back from v_3 to v_{12} directly with cached message m_{v_3}.

Formally, we define the workflow of the Region-Aware message management as follows:

For any vertex $v \in V_k$:

$$lx_v^i = \mathcal{A}\left(\{m_{u,v}^{i-1} \mid u \in V_k\}\right), \tag{2}$$

$$gx_v^i = \mathcal{A}\left(\{m_{u,v}^{i-1} \mid u \in V \setminus V_k\}\right), \tag{3}$$

$$s_v^i = \mathcal{A}\left(s_v^{i-1}, \mathcal{A}\left(lx_v^i, gx_v^i\right)\right), \tag{4}$$

$$m_{v,w}^i = \mathcal{I}\left(\mathcal{A}\left(lx_v^i, gx_v^i\right), C_E(v,w)\right) \text{ for } w \in V_k, \tag{5}$$

$$m_{v,w}^i = \mathcal{I}\left(\mathcal{A}\left(lx_v^i, \{m_{u,v}^{i-1} \mid u \in V \setminus V_l\}\right), C_E(v,w)\right) \tag{6}$$

for $w \in V_l (l \neq k)$.

For any proxy $P_{k,l}$:

$$m_{P_{k},w}^i = \overline{\mathcal{I}}\left(\mathcal{A}\left(\{m_{v,w}^i \mid v \in V_k\}\right)\right) \text{ for } w \in V_l, \tag{7}$$

$$m_{P_l,u}^{i+1} = \mathcal{I}\left(m_{P_k,w}^i, C_E(w,u)\right) \text{ for } u \in V_k \text{ and } w \in V_l. \tag{8}$$

For any vertex v, in each iteration, v in data center D_k first aggregates the messages received from its local neighbors (2) and other data centers (3), then updates its state in the ith round based on the aggregation results (4). It finally generates messages to its local neighbors (5) and remote neighbors (6). Equation (6) indicates that the generated messages sent to the remote proxy in D_l are only based on the messages from the data centers *except* for D_l, since our optimization corresponding to Observation 1 (i.e., the ping-pong effect) has already applied the effect of the messages from D_l in last round. All vertex operations (2)–(6) occur on the lower layer of the two-layer interaction view.

For any proxy $P_{k,l}$, for two-layer interaction view, $P_{k,l}$ takes different interaction functions for upper-layer message interaction and lower-layer message management. Specifically, at the upper layer, $P_{k,l}$ sends the cached messages to the remote data center D_l through a direct interaction function $\overline{\mathcal{I}}$ (7) without coordinating with other proxies. At the lower layer, considering the ping-pong effect, $P_{k,l}$ directly computes the message that will be sent back to local neighbor u from remote neighbor w through the cached message to apply it to u one step ahead through the interaction function $\mathcal{I}$ (8).

Example 3: We employ the process of PageRank as an example to illustrate the above workflow:

- $\mathcal{A}(M_{*,v}^{i-1}) = \mathsf{sum}(M_{*,v}^{i-1})$; $\mathcal{A}(s_v^{i-1}, x_v^i) = \mathsf{sum}(s_v^{i-1}, x_v^i)$;
- $\mathcal{I}(x_v^i, C_E(v,w)) = d \times x_v^i / N_v$ $(\forall w \in \mathsf{N}_{out}(v))$;
- $\overline{\mathcal{I}}(x_{P_k,w}^i) = x_w^i$ $(\forall w \in V_l)$.

PageRank uses sum to aggregate messages from local (i.e., lx_v^i) and remote (i.e., gx_v^i) neighbors and update state s_v^i. Since condition **(C2)** holds, the interaction function $\mathcal{I}$, i.e., $d \times x_v^i / N_v$, can be applied to multiparty operations. In the ping-pong effect, the proxy uses the cached global messages x_w^i to generate new local messages. In the two-layer view, vertex v in the lower layer can continuously generate local and global messages via $\mathcal{I}$ from lx_v^i or gx_v^i, and use $\overline{\mathcal{I}}$ in the upper layer to send global messages x_w^i without coordination. In addition, the proxy can perform $\mathcal{I}$ replacing the congested side when it knows the v's out-degree N_v. □

C. Theoretical Analysis

This subsection provides a theoretical analysis for the proper execution of the Region-Aware framework. RAGraph performs layered interaction in the hierarchical network and coordination-free message interaction between replicas with ping-pong computation may confuse the process. We introduce *Delta State Conflict-free Replicated Data Type* (δ-CRDT) [26], [27] to guarantee the *Strong Eventual Consistency* between replicas (i.e., all correct replicas reach the same state without conflicts).

δ-CRDT provides a *mutation function* m^δ for an update operation, and the state transition of each replica by joining (i.e., $\sqcup$) the current state s and $m^\delta(s)$, i.e., $s' = s \sqcup m^\delta(s)$. Interactions between replicas are occurred by joining each other's mutation updates $m^\delta(s_*)$.

Assume (1) $\sqcup$ is *associative*, *commutative*, and *idempotent* (i.e., ACI property), (2) the object has causal consistency assurance, and (3) $m^\delta(s)$ on each replica is joined to each other at least once, then δ-CRDT guarantees that all replicas eventually reach a consistent convergence state without conflicts.

Applying CRDT to Graph Processing: Back to the iterative graph processing with monotonic property, $\mathcal{A}$ and $\mathcal{I}$ can be analogous to the join (i.e., $\sqcup$) and mutation (i.e., m^δ) functions of δ-CRDT respectively. Specifically, the state of vertex v after i iterations is:

$$s_v^i = \mathcal{A}(s_v^{i-1}, \mathcal{A}(M_v^{i-1})) \tag{9}$$

$$= \mathcal{A}\left(s_v^{i-1}, \cup_{k=1}^{j} m_{k,v}^{i-1}\right) \tag{10}$$

$$= \mathcal{A}(s_v^{i-1} \cup \mathcal{I}(x_{1,v}^{i-1}) \cup \ldots \cup \mathcal{I}(x_{j,v}^{i-1})) \tag{11}$$

where $m_{k,v}^{i-1} = \mathcal{I}(x_{k,v}^{i-1})$. Following the conditions (**C1**) and (**C2**), (9) can be reorganized to obtain (11). Eq. (11) can be considered that part of the messages (e.g., $m_{v,v}^{i-1}$) come from v, while others (e.g., $\cup_{k=1}^{j}\{m_{k,v}^{i-1} \mid k \neq v\}$) come from the replicas. As a result, all replicas use $\mathcal{A}$ to *join* the new messages through the *mutation* of $\mathcal{I}$. We next explore in detail the feasibility of $\mathcal{A}$ and $\mathcal{I}$ to ensure the correct execution following δ-CRDT.

Causal Consistency: Causal consistency means that all "causally" related (or potentially related) events must appear in the same order. δ-CRDT guarantees correct causality by specifying the *causal merging* of a group of mutation updates. For iterative graph processing, following monotonic property, we have:

Theorem 1: Consider iterative graph processing $\mathcal{P}$ with $\mathcal{A}$ and $\mathcal{I}$ for multi-replica participation. If $\mathcal{A}$ and $\mathcal{I}$ satisfy the monotonic conditions, then RAGraph with $\mathcal{A}, \mathcal{I}$ and $\mathcal{P}$ guarantees that successive joins on individual proxies have no causality. □

Proof sketch: Based on monotonic property, the state s^n of a vertex after n iterations is:

$$\begin{aligned} s^n &= \mathcal{A}\left(s^{n-1}, \mathcal{A}(M^{n-1})\right) \\ &= \mathcal{A}\left(\mathcal{A}(s^{n-2}, \mathcal{A}(M^{n-2})) \cup \mathcal{A} \circ \mathcal{I}(\mathcal{A}(M^{n-2}))\right) \\ &= \mathcal{A}\left(s^0 \cup (\mathcal{A} \circ \mathcal{I})(M^0) \cup \ldots \cup (\mathcal{A} \circ \mathcal{I})^n(M^0)\right) \end{aligned} \tag{12}$$

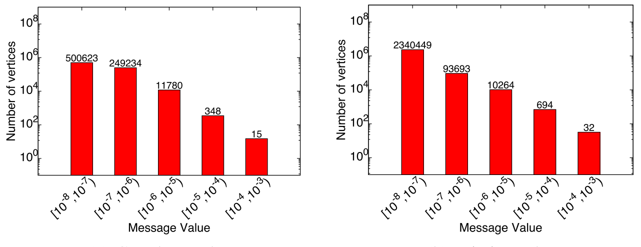
Here $\circ$ is the function composition operator, which represents a set of operations to be applied consecutively, e.g., $\mathcal{A} \circ \mathcal{I}(M) = \mathcal{A}(\mathcal{I}(M))$. s^0 and M^0 denote the initial vertex state and the intermediate message set, respectively. For arbitrary round i, $i > 0$, we have $\mathcal{A}(M^i) = \mathcal{A} \circ \mathcal{I}(\mathcal{A}(M^{i-1})) = \mathcal{A} \circ \mathcal{I}(M^{i-1})$ and $M^i = \cup_j m^i$ under monotonic conditions. We decompose the set M^0 (i.e., $s^n = \mathcal{A}((s^0 \cup (\mathcal{A} \circ \mathcal{I})(\cup_j m^0) \cup ... \cup (\mathcal{A} \circ \mathcal{I})^n(\cup_j m^0)))$. Following condition (**C2**), any unordered m (e.g., $\{m^i, m^{i-3}, m^{i+3}\}$) can be joined to act on $\mathcal{I}$. Therefore, when the set M^0 is dispersed among the replicas, there is no causality in the message delivery between the proxies under the established correct rules. □

ACI Property: δ-CRDT guarantees eventual consistency and convergence of replicas via the ACI property. However, as mentioned previously, condition (**C1**) defines the commutative and associative properties of $\mathcal{A}$ but has no constraint on idempotence (i.e., $\mathcal{A}(X, X) = X$). For example, PageRank's aggregation function sum does not satisfy the idempotent property. The idempotent property avoids duplicate delivery anomaly. Theorem 2 gives system constraints to guarantee the equivalence with ACI property.

Theorem 2: Assume an underlying reliable communication protocol. If each message is aggregated by $\mathcal{A}$ to each replica exactly once, and the replica performs exactly-once interaction with its neighbors by $\mathcal{I}$, then all replicas reach the same state without time constraint. □

Proof sketch: Define an *elementary message* as one that does not go through other vertices except its destination. Our observation is that for a pair of adjacent vertices u, v in different data centers D_k and D_l, respectively, there will be only one elementary message from u to v, the path of which is $u \longrightarrow P_{k,l} \longrightarrow v$. The value of the message will be affected by the interaction function $\mathcal{I}$ and $\overline{\mathcal{I}}$ in path $u \longrightarrow P_{k,l}$ and $P_{k,l} \longrightarrow v$, respectively. Since $\overline{\mathcal{I}}$ does not change the input, the value of the message from u to v is exactly $\mathcal{I}(x_u^i, C_E(u, v))$, same as the value directly received from u.

From the vertex-centric model, the result of the complete message obtained by vertex v in round ith is $\mathcal{A}(\{m_{u,v}^{i-1} \mid e(u, v) \in E\})$. Based on (2) and (3), the intermediate result of v in the ith round would be $\mathcal{A}(lx_v^i, gx_v^i) = \mathcal{A}(\{m_{u,v}^{i-1} \mid e(u, v) \in E\})$. Therefore, when aggregating gx_v^i from other replicas exactly once, a complete and correct message result from the current round is obtained. □



(a) Google graph (b) WiKi graph

Fig. 3. The distribution of message values on the boundary vertices.

IV. Contribution-Driven Edge Migration

This section further explores the data preprocessing optimization for cross-datacenter message transmission efficiency from a data placement perspective.

Data in geo-distributed environments is generated and maintained at the nearest data center. The attempt to replace all edges among data centers for achieving optimal communication cost results in extensive data transmission, which is unaffordable in real production environments [28]. Moreover, as the graph data evolves, the existing placement of edges becomes no longer optimal. For the above reasons, RAGraph transforms the edge repartitioning problem to edge migration, focusing on only incrementally modifying the placement of a minor number of edges on the boundary.

In iterative graph processing, the message value of vertex interaction reflects the contribution of the vertex to the convergence of the algorithm [29], [30]. In a geo-distributed cluster configuration depicted in Fig. 1(a), we count the distribution of message values for boundary vertices on an iteration step when performing PageRank on the Wiki graph [31] and Google graph [32] using the graph processing system GRAPE [6]. As shown in Fig. 3, less than 5% of the vertices have large-valued messages, while a substantial number exhibit messages with low contributions. Transmitting messages from a large number of low-contribution vertices across data centers is inefficient, as it depletes valuable network resources while offering minimal contributions to convergence.

Basic Idea: Edge migration aims at more efficient message interaction on the boundary by migrating edges with low-contribution message propagation in scarce and heterogeneous networks. Specifically, RAGraph evaluates the boundary vertices in considering contribution and network bandwidth, and based on the evaluation scores, low-contribution edges are relocated within the dependent data centers until high-contribution vertices appear on the boundary.

We first define the *vertex contribution* ($\mathcal{C}$) to measure the degree of influence of vertex generation and propagation of messages. Following the vertex-centric model, the contribution of a vertex v depends on its capability to collect numerous or significant message values from incoming neighbors, or to initiate widespread interactions through outgoing neighbors.

The contribution metric $\mathcal{C}_v$ of vertex v is formulated as:

$$\mathcal{C}_v = \sum_{u\in \mathsf{N}_{in}(v)} \frac{\alpha \cdot w_{u,v} \cdot \mathcal{D}(u)}{\sum_{w\in \mathsf{N}_{out}(u)} w_{u,w}} + \mathcal{D}(v), \quad (13)$$

where α is a decay factor, 0.8 by default. $\mathcal{D}(v)$ denotes the degree of vertex v, and $w_{u,v}$ is the weight of edge (u,v), which is set to 1 on the unweighted graph. Here $\mathsf{N}_{in}(v) = \{u \mid e(u,v) \in E\}$ and $\mathsf{N}_{out}(u) = \{w \mid e(u,w) \in E\}$.

Objective Function: Given a set of data centers $D = \{D_1, ..., D_h\}$ and a set of boundary vertices $V_b = \{v_1, ...v_n\}$. The average network transmission rate R from D_i to D_j is denoted as R_{D_i,D_j}. Intuitively, a higher $\mathcal{C}$ and R correspond to more efficient message interaction, resulting in an overall increase in communication gain. Therefore, our objective is to find an optimal migration scheme that maximizes the overall communication gain:

$$O_g = \max \sum_{v\in V_b} \sum_{D_j \in D(v)} \mathcal{C}_v \cdot R_{D_i,D_j}. \quad (14)$$

Here the original vertex of v is in D_i, and $D(v)$ denotes the data center set where replicas of vertex v are located.

Geo-distributed graph processing primarily focuses on message interaction. We use vertex contribution to characterize the data center load, i.e., $\mathcal{L}_{D_i} = \sum_{v\in V_i} \mathcal{C}_v$. The load constraint is formulated to balance the message interaction cost as follows:

$$\frac{\max\limits_{D_i} \sum_{v\in V_i} \mathcal{C}_v}{\frac{1}{h}\sum_{v\in V} \mathcal{C}_v} \le 1 + \xi \quad (15)$$

where imbalance factor ξ is a constant satisfying $0 \le \xi \le 1$.

Edge Migration: For graph data naturally distributed over data centers, we first extract the vertex contribution $\mathcal{C}$ in parallel and obtain the average network transmission rate R on each network link. Subsequently, according to metric $\mathcal{C}$ and R, we propose a heuristic edge migration algorithm. The core idea is to continuously search for high contribution vertices as boundary vertices near the boundary based on the contribution metric. Meanwhile, traversed low-contribution edges are sent to the corresponding remote data centers to balance the load.

Algorithm 1 describes the whole process of edge migration on contribution basis. Each data center D_i and its corresponding graph G_i arrange the remaining data centers in descending order based on R (Line 2). For each pair of data centers, we migrate edges from the high load side to the low load side and calculate the load difference between the two (Line 5–6). In the performing migration data center, we establish a migration buffer *mb* to cache edges designated for transmission (Line 7). Subsequently, we sort the boundary vertices of the data center pair in increasing order of contribution (Line 8). For each boundary vertex, a breadth-first search (BFS) is performed to collect the set of edges E' from candidate neighbors whose contributions are less than that boundary vertex. We then determine whether the inclusion of E' exceeds the tolerable imbalance load difference. If not, we add E' to mb; otherwise, we incrementally add edges in E' to mb until the capacity is saturated (Line 9–19). Finally, we transmit the edges in mb to the corresponding data center and update the load on both sides (Line 20–21).

Algorithm 1: Edge Migration Algorithm.

```
1  procedure Migration ($G_i$, $D_i$)
2  $D' \leftarrow$ Sort($D \setminus D_i$);
3  for each $D_j$ in $D'$ do
4      Migration executed in high-load data center $D_{exec}$
5      $D_{exec}$ = Max($\mathcal{L}_{D_i}, \mathcal{L}_{D_j}$);
6      $\Delta c = |\mathcal{L}_{D_i} - \mathcal{L}_{D_j}|$ / 2;
7      $mb \leftarrow \emptyset$;  //migration buffer
8      $V_b'$ = Sort($V_i \cap V_j \cap V_b$);  //boundary vertex set of $D_{exec}$
9      while $\mathcal{L}_{mb} \le \Delta c$ do
10         $v_b \leftarrow V_b'$;
11         $E' \leftarrow$ searchEdgeByBfs($v_b$);  //search for candidates
12         $mb' \leftarrow mb + E'$;
13         if $\mathcal{L}_{mb'} \le (1+\xi)\cdot\Delta c$ then
14             $mb \leftarrow mb + E'$;
15         else
16             while $\mathcal{L}_{mb} \le (1+\xi)\cdot\Delta c$ do
17                 $e' \leftarrow E'$;
18                 $mb \leftarrow mb + e'$;
19         Update the boundary vertex set $V_b'$ if $V_b'$ is empty;
20     migrateMb($mb$);  //migrate edges in the $mb$
21     recount($\mathcal{L}_{D_i}, \mathcal{L}_{D_j}$);  //update $\mathcal{L}_{D_i}$ and $\mathcal{L}_{D_j}$
22 end procedure
```

When the graph changes, performing contribution measurement and incrementally updating the data center load on the topology surrounding the change is only necessary. Afterward, edge migration is executed according to Algorithm 1.

Task Outsourcing: Upon migrating specific edges to a remote data center, the corresponding graph processing tasks are also outsourced to that data center. After the migration, the data center globally updates the location information for these migrated vertices, while the proxy concurrently updates the maintained information. The above operations ensure that these migrations can be processed like local vertices by the remote data center, while freeing the original data center from executing processing tasks on these vertices.

V. Heterogeneous-Aware Message Passing Management

This section presents two important runtime optimizations based on the Region-Aware framework.

A. Adaptive Hierarchical Message Interaction

To adapt to the heterogeneous and fluctuating networks, we design an adaptive hierarchical message interaction engine. Our approach derives from two insights into the message passing in geo-distributed environments. First, as shown in Example 1, the bandwidth of WANs is highly heterogeneous. As a result, the commonly adopted real-time message passing in the asynchronous model would generate frequent cross-datacenter communication and cause an intolerant overhead. In contrast, an alternative way is to prioritize computation of subgraph within the data center to achieve significant message interaction with less frequent communication but ignores precious WAN resource utilization. A better way is to consider combining the two in the network status.

Second, the data transmission rate between data centers *fluctuates significantly* [33]. Meanwhile, during the iterative computation, the number of vertices activated for computing in data centers changes dynamically, which leads to a variable amount of transmitted messages. From this perspective, RAGraph needs to adaptively switch message interaction strategies based on the current network transmission status.

Basic Idea: Based on the above considerations, we design and implement an adaptive hierarchical message interaction engine on the proxy. The key idea of the engine is to allow *hierarchical* message interactions and to *adaptively* choose the message interaction strategies based on the status of the network. The proxies in the Region-Aware framework are equipped with two types of message interactions: *eager* message interaction for timely vertex updates and *lazy* message interaction for significant vertex updates.

For the part of the network with low latency (including intra-region and part of inter-region networks), the proxy adopts an eager mode, sending messages eagerly to the corresponding data center as soon as they are generated. In eager message interaction, the sender proxy can proactively determine when to send the messages to the other proxy. In contrast, for the part of the network with high latency, the sender proxy adopts a lazy mode in which the receiver proxy decides when to fetch messages from the lazy sender proxy. Specifically, in the receiver proxy, when the cached messages tend to achieve local convergence, and no external messages are received, a "fetch" request is sent to the corresponding data center. Correspondingly, the sender proxy keeps accumulating messages and sends the accumulated messages when receiving the "fetch" request, which we name *lazy* message interaction.

How to choose the message interaction mode? Here we propose an adaptive strategy to switch the mode based on the network fluctuation and message traffics. We define τ as the average bandwidth of the global network and μ as the maximum message size for the remote vertices of each proxy's. During the execution, each proxy counts the average transmission data size $S_{\delta t}$ and average network transmission rate R in the time window ΔT, and adaptively selects the message interaction mode based on their ratio: if $S_{\delta t}/R < \lambda \cdot \mu/\tau$, the proxy will execute in eager mode and otherwise switch to lazy mode. Here λ is a configurable parameter, which is set to 0.6 in our experiments.

The structure of the engine is shown in Fig. 4. The engine contains the proxies for message interaction between D_k and D_l. Proxies on different links in the data center can exhibit different eager/lazy modes. The message interaction engine of each proxy includes a *detector* and a *switcher*. The detector is responsible for recording $S_{\delta t}$ and R, while the switcher decides which mode to use and notifies the remote proxy. Note that the intra-region networks use the eager mode by default.

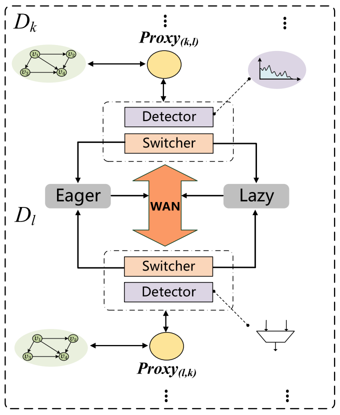


Fig. 4. Adaptive hierarchical interaction engine structure.

Algorithm 2: Discrepancy-Aware Message Filtering.

```
input : Messages $M$ that sequentially come
output: Cached messages partitioned in buckets
1 $\delta B_1 = \delta B_1^*$, $\delta B_2 = \delta B_2^*$;
2 for $m$ in $M$ do
3     Assign $m$ to $B_i$ if value of $m$ falls in range of $B_i$;
4     if $|B_3| \leq \gamma \sum_{i=1}^{3} |B_i|$ then
5         if $|B_2| \geq \sigma |B_1|$ then
6             $\delta_k = \frac{\delta B_1^{k-1} + \delta B_2^{k-1}}{2\Delta \overline{x}_k}$;
7             $\delta B_i = \frac{\delta B_i}{\delta_k}$ $(i = 1, 2)$;
8             Reassign cached messages based on $\delta B_1$, $\delta B_2$;
```

B. Discrepancy-Aware Message Filtering

Due to the heterogeneity of WAN networks, imbalanced communications occur in geo-distributed graph processing, where a vertex may receive highly discrepant messages generated at different iterations. Some of these important messages can make a large change to the vertex state, producing more *significant updates* thus advancing convergence. However, the range of message variability changes dynamically as the process proceeds, making it difficult to capture the current important messages.

Basic Idea: RAGraph employs buckets with adaptive ranges to filter important messages to reduce the impact of network status on message filtering. Specifically, each proxy maintains the messages to be propagated and assigns them to different buckets according to their values. Those unimportant messages (*i.e.* with a small change in value) will be delayed until they have accumulated enough importance. With the values of overall messages decreasing along with iterations, the ranges of the buckets adaptively vary to capture current important messages.

Algorithm 2 illustrates the pseudocode of the discrepancy-aware message filtering strategy in a proxy. Each proxy in RAGraph maintains 3 buckets B_1, B_2, and B_3, storing *unimportant*, *lowly important*, and *highly important messages* respectively. The ranges of B_1, B_2, and B_3 are denoted as $(0, \delta B_1]$, $(\delta B_1, \delta B_2]$, and $(\delta B_2, \infty)$, respectively. Each message will be categorized into buckets based on its value (line 3). If the number of messages in B_3 is below a ratio of the total number of messages, the system will decrease the ranges of buckets because the highly important messages are rare (lines 6–8). The ranges of buckets will be divided by a unified variable δ_k, and thus the ratio of δB_1 to δB_2 stays invariable. Formally, we let

$$\delta_k = \frac{\delta B_1^{k-1} + \delta B_2^{k-1}}{2\Delta \overline{x}_k}, \quad \delta B_i^k = \frac{\delta B_i^{k-1}}{\delta_k} \text{ for } i = 1, 2$$

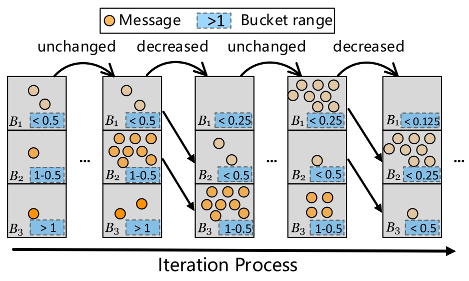


Fig. 5. Buckets for message filtering. Cycle: cached outward message (the darker the color, the larger the values).

TABLE II
DATASET DESCRIPTION

Graph	Vertices	Edges	Abbreviation
Web-Google [32]	916,428	6,078,250	GL
Enwiki-2013 [31]	4,203,323	101,311,614	WK
Arabic-2005 [34]	22,744,080	639,999,458	AB
UK-2005 [35]	39,459,925	936,364,282	UK
Twitter-2010 [13]	41,652,230	1,468,364,884	TW

where $\Delta\overline{x}_k$ denotes the average value of outgoing messages at time t_k. By dividing δ_k, the average value of messages is exactly at the middle of B_2. Thus, the distribution of message values can be well depicted. Fig. 5 illustrates the process of the strategy in a proxy.

Detection of Shifting Distribution: We may encounter a situation that $|B_3| \leq \gamma \sum_{i=1}^{3} |B_i|$ while at the same time $|B_2| \ll |B_1|$. Such a fluctuating distribution counters the intuition that B_2 should contain a number of messages due to the continuously decreasing process of the ranges. In practice, $|B_2| \ll |B_1|$ indicates that a considerable number of messages are still passing in the network and have not been received, which is caused by the gap between computation and communication. In such a case, we choose to make the buckets unchanged until the shifting stops. Thus, we additionally require $|B_2| \geq \sigma|B_1|$ (line 5 in Algorithm 2) to avoid shifting distribution from the decrease of ranges.

VI. EXPERIMENTAL EVALUATION

A. Experimental Setup

Datasets and Test Algorithms: We use five real-world datasets (see Table II) in our experiments, including Web-Google [32], Enwiki-2013 [31], Arabic-2005 [34], UK-2005 [35], and Twitter-2010 [13]. Graphs are partitioned in the common *uniform-chunk* strategy unless otherwise stated. That is, vertices are ordered in their local IDs and uniformly partitioned in different data centers. We use four typical monotonic graph algorithms in the experiments, including PageRank [14], Penalized Hitting Probability (PHP) [36], Single Source Shortest Path (SSSP) [37] and Connected Components (CC) [38].

Competitors: We compare RAGraph with a representative distributed graph processing system, GRAPE [6], and two state-of-the-art geo-distributed graph processing systems, Monarch [16] and GeoGraph [17]. All competitors and the corresponding test algorithms are implemented on top of libgrape-lite [39].

Environments: All algorithms are implemented in C++, and the average result of three runs is reported. AliCloud ECS clusters from five regions are chosen as geo-distributed data centers for evaluation, including Qingdao, China; Singapore; Sydney, Australia; Frankfurt, Germany; Virginia, USA. Each data center is allocated 16 AliCloud ecs.r5.2xlarge instances (8vCPU, 64GB memory).

B. Overall Performance

We first evaluate the overall performance of RAGraph, including running time and WAN cost, by comparing it with competitors.

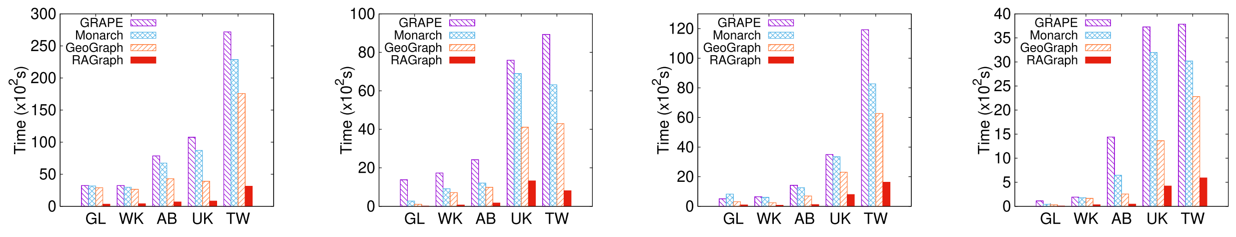
Running time: Fig. 6 shows the running time of PageRank, PHP, SSSP, and CC algorithms in the compared systems. As can be seen from the results, RAGraph outperforms others in all cases. Specifically, RAGraph achieves $4.38\times -98.42\times$ ($15.61\times$ on average) speedup over GRAPE, $4.18\times -19.59\times$ ($8.65\times$ on average) speedup over Monarch, and $2.88\times -10.51\times$ ($5.3\times$ on average) speedup over GeoGraph. RAGraph does perform iterative graph algorithms efficiently in geo-distributed environments. This is attributed to RAGraph's unique message interaction and communication optimization designs, which accelerate global message interaction, eliminate coordinated waiting times, and reduce the data transmission between data centers.

WAN cost: We measure the transmitted data size across data centers via WANs for each system. Fig. 7 shows the WAN cost of each system. As can be observed, RAGraph incurs the smallest WAN cost on all tested conditions. Specifically, RAGraph reduces WAN cost by 64.3% -97.3% (84.4% on average) compared with GRAPE, 58.3% -96.8% (81.3% on average) compared with Monarch, and 50% -92% (70.4% on average) compared with GeoGraph. The communication gains arise from the message filtering optimization and effective graph data placement strategies proposed in RAGraph.

C. Performance Gain Analysis

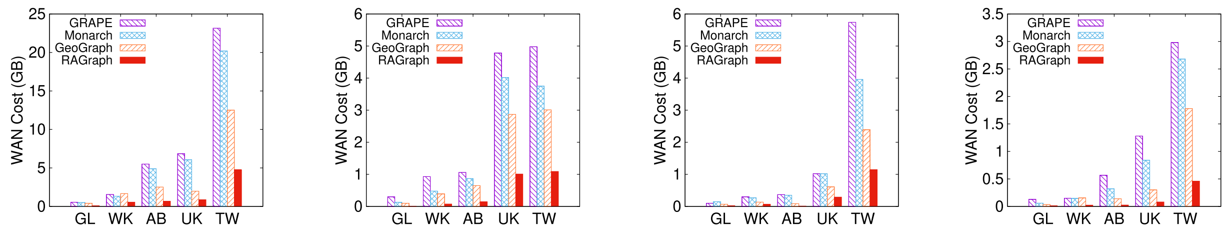
The performance of RAGraph mainly comes from the flexible Region-Aware framework, two runtime optimizations for heterogeneous-aware message passing, and graph data placement optimizations. In this subsection, we quantitatively analyze the gain from the above strategies. Specifically, we report the running time and WAN cost of the test algorithms by successively enabling RAGraph components, including Region-Aware framework in Section III, adaptive hierarchical message interaction, discrepancy-aware message filtering (introduced in Section V), and data placement optimizations in Section IV, denoted RA, RA + Hi, RA + Hi + Fi, and RAGraph respectively. The results are compared with the traditional synchronous graph processing system libgrape-lite [39] and its asynchronous version modified based on Maiter [14], which we denote Sync and Async, respectively.

The *normalized* running time and WAN cost of Sync, Async, RA, RA + Hi, and RAGraph for PageRank and SSSP are reported in Fig. 8. The results for PHP and CC show a similar



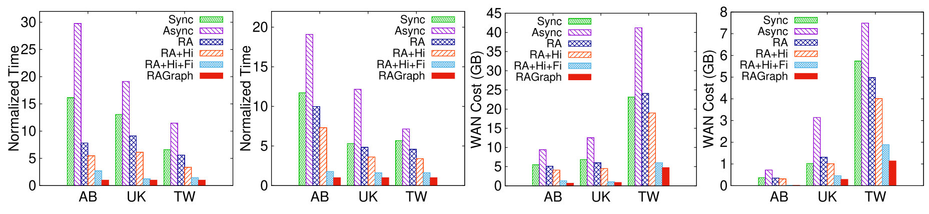
(a) PageRank (b) PHP (c) SSSP (d) CC

Fig. 6. Running time comparison.



(a) PageRank (b) PHP (c) SSSP (d) CC

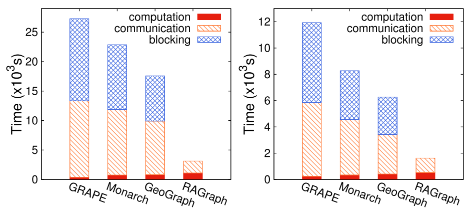
Fig. 7. WAN cost comparison.



(a) PageRank running time (b) SSSP running time (c) PageRank WAN cost (d) SSSP WAN cost

Fig. 8. Performance gain from RAGraph.

trend, and we omit the figures due to space limitations. The running time of the RAGraph is reported as the unit time (i.e., 1). From Fig. 8, one can find that the running time and WAN cost are reduced after applying each component of RAGraph in turn. Specifically, Region-Aware framework can achieve $1.09\times -2.06\times$ speedup compared with Sync, and $1.56\times -3.8\times$ speedup compared with Async. By further enabling the adaptive hierarchical message interaction, RA + Hi achieves $1.33\times -1.67\times$ speedup and reduces $16\% -40.1\%$ WAN cost compared with RA. The discrepancy-aware message filtering method (RA + Hi + Fi) achieves a $2.03\times -6.58\times$ speedup and a $43.7\% -94.7\%$ reduction in WAN cost. Finally, startup graph data placement optimizations (RAGraph) lead to $1.23\times -2.7\times$ speedup and $14.7\% -49.4\%$ WAN cost reduction. This validates the efficacy of the proposed Region-Aware framework and optimization strategies. Another observation is that Async produces the largest running time and WAN cost in most cases. This verifies our claim in Section I that traditional distributed graph processing systems cannot solve the problems in geo-distributed environments well. Besides, compared with Sync, the gain of running time from the Region-Aware architecture (i.e., the gap between RA and Sync) is more significant than that of WAN cost.



(a) PageRank (b) SSSP

Fig. 9. Performance Breakdown.

This indicates that the Region-Aware framework can largely eliminate coordinated waiting times in Sync.

D. Performance Breakdown

As discussed in Example 1, the overall runtime consists of computation, communication, and blocking time. To study the effect of each component on RAGraph, we run PageRank and SSSP on the TW graph and profile the running time of each component recorded in the data center located in Singapore. The result is shown in Fig. 9. We can see that the communication and

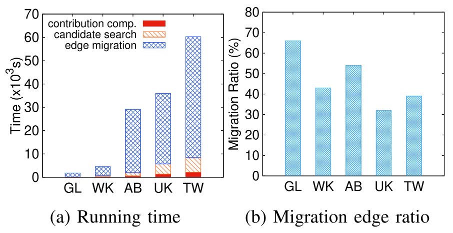


Fig. 10. Edge migration cost.

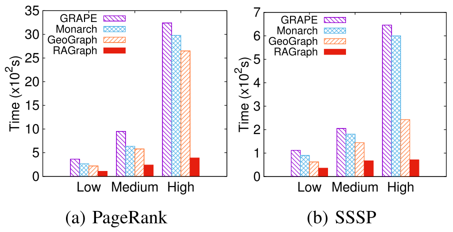


Fig. 11. Sensitivity to network status.

blocking take up most of the running time, while computation is lightweight. Compared with competitors, RAGraph eliminates the blocking time and generates the least communication.

E. Edge Migration Cost

We evaluate the cost of the edge migration algorithm, Fig. 10 shows the running time of the algorithm and the ratio of edges to be migrated on all five test graphs. The overall running time during the algorithm's execution is composed of three parts: the computation of vertex contribution, the search of candidate edges for migration, and the execution of edge migration. The proportion of running time for each component is shown in Fig. 10(a). In geo-distributed environments, the limitations of network bandwidth significantly impact the efficiency of transmitting migrated edges, making it the primary cost factor. This cost is directly proportional to the ratio of migrated edges. Besides, the ratio of edge migration is related to the raw edge positions. A poor raw edge allocation leads to a higher ratio of edge migration to optimize the boundary vertex contribution and balance the load with a more significant performance increase.

F. Sensitivity to Network Heterogeneity

This subsection evaluates the impact of network heterogeneity on the systems. We use different data center locations around the world to build low/medium/high-heterogeneity networks. Specifically, the low-heterogeneity network is constructed based on data centers in China (including Beijing, Shanghai, Qingdao, Hangzhou, and Guangzhou); the medium-heterogeneity network is based on Asia-wide data centers (including Tokyo, Japan; Singapore; Seoul, Korea; Beijing, China; and Mumbai, India); and high-heterogeneity network is based on worldwide data centers (configuration see Section VI-A). Fig. 11 shows the result of PageRank and SSSP in different systems on the WK graph. Compared with the competitors, RAGraph achieves $1.73\times -3.4\times$ speedup on the low-heterogeneity network, $2.16\times -3.95\times$ speedup on the medium-heterogeneity network, and $3.42\times -9.09\times$ speedup on the high-heterogeneity network. RAGraph shows substantial superiority on the high-heterogeneity network, which validates the effectiveness of the Region-Aware framework.

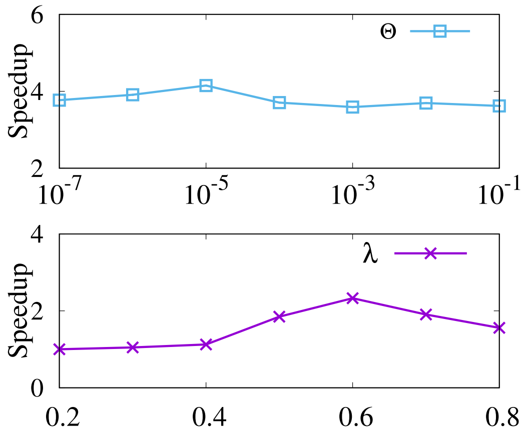


Fig. 12. Sensitivity to Θ and λ.

G. Sensitivity to Parameter Settings

We evaluate the impact of the two configurable parameters, i.e., λ and Θ, which control the eager/lazy mode switching in Section V-A and the algorithm convergence, respectively. We run PageRank on WK graph, varying Θ from 10^{-7} to 10^{-1} and λ from 0.2 to 0.8. For the experiment associated with λ, we normalize the running time of all cases with $\lambda = 0.2$ as unit time. As shown in the lower part of Fig. 12, as λ increases, more proxies turn into eager mode but may suffer high latency networks resulting in inefficiencies, and RAGraph reaches its best performance when λ is set to 0.6. For the experiment associated with Θ, we run Pagerank on RAGraph and GRAPE and report the speedup of RAGraph over GRAPE under different Θ. As shown in the upper part of Fig. 12, the convergence threshold change has less effect on the effectiveness of RAGraph.

H. Scalability

We finally test the scalability of RAGraph by enlarging the number of data centers. As the number of data centers increases, more cross-datacenter messages will be triggered, which limits the performance of the systems. To evaluate the impact of scaling the number of data centers on RAGraph, we run PageRank with the number of geo-distributed data centers varying from 2 to 8. Using the TW graph, the whole is partitioned into the corresponding number of parts placed in each data center using the uniform-chunk method. We take the running time on 2 data centers as the baselines, and Fig. 13 shows the result from RAGraph and the competitors. As the number of data centers increases, GeoGraph and RAGraph grow slower than GRAPE and Monarch, and RAGraph performs the best. GeoGraph derives scaling gain from the clustering of data centers. While the more independent region computation and communication optimization allow RAGraph to gain better scalability.

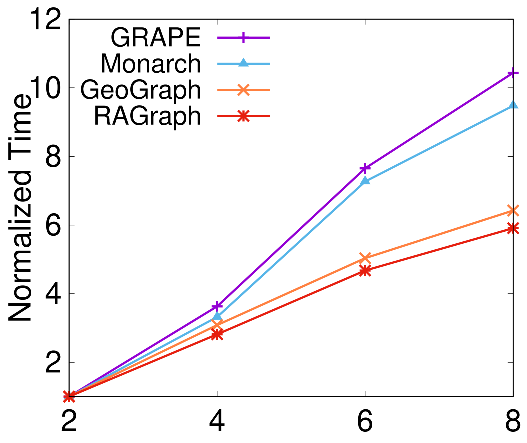


Fig. 13. Scalability.

VII. Related Works

Graph-Structured Data Processing: A large number of traditional graph processing systems have been developed for large-scale graph data analysis. The Bulk Synchronous Parallel model [15] is introduced into graph processing by Pregel [4] and adopted by most distributed graph processing systems [1], [3], [6]. While some systems, such as GraphLab [5], Maiter [14], and GRAPE+ [40], use the asynchronous parallel model (AP) to eliminate synchronization overhead. PowerSwitch [41] uses a hybrid mode for adaptive switching between sync and async during computation. Galois [42] and Priter [29] design priority scheduling from the algorithmic perspective, but may be limited by the impact of network transmission status on data scheduling. LazyGraph [43] involves replicas in local computation and sets a global sync data coherency stage to get a global view of replicas for lazy data consistency. Additionally, POCLib [44] introduces orthogonal processing on compression, enabling efficient data analysis directly on compressed data regardless of the processing type. This offers new opportunities to further reduce network costs in geo-distributed environments through data compression. On the other hand, Monarch [16] and GeoGraph [17] are designed for geo-distributed graph processing. Both of them exhibit excellent performance, but inevitably coordinate with other workers on the WANs and fail to consider the impact of network fluctuation.

RAGraph strives to optimize the heterogeneous and fluctuating network in geo-distributed environments, eliminating coordinated waiting times and reducing communication costs through unique message interactions and communication optimization designs. By implementing a Region-Aware framework and two runtime optimizations, it achieves a 7.6× speedup and an 83.6% reduction in WAN cost on average compared to traditional Sync and Async systems and a 3.91× speedup and a 58.7% reduction in WAN cost on average compared to Monarch and GeoGraph.

Monotonic Data Analysis: The monotonicity idea has been widely used in many fields. PowerLog [20] proposes monotonic recursive aggregate evaluation in Datalog, which provides theoretical guarantees for incremental and asynchronous execution of recursive aggregation programs. RisGraph [45] supports analysis for monotonic algorithms on evolving graphs to achieve high throughput and low latency simultaneously. KickStarter [46] proposes an incremental graph computing model for monotonic algorithms to produce correct results and converge quickly. GoGraph [47] leverages monotonicity to reorder the vertex processing sequence, thereby reducing the number of graph computation iterations. RAGraph exploits monotonicity to accelerate cross-region execution efficiency and enable coordination-free iterative processing.

Geo-Distributed Data Analysis: Several works focus on designing more efficient big-data analysis frameworks in geo-distributed environments. For example, Medusa [48] allows geo-distributed computation without modifying the Hadoop semantics. GeoDis [49] optimizes data-intensive jobs by considering data localization and migration. Both of them are MapReduce-based. Lube [50] and Tetrium [51] are Spark-based frameworks. Lube reduces the response time by optimizing runtime bottlenecks, and Tetrium considers network and computational resources to achieve multiple resource allocation. Zhou et al. [52] develop an online control framework based on Lyapunov optimization, achieving dynamic balance among electricity costs, carbon emissions, and SLA in geo-distributed data centers. λGrapher [53] introduces an innovative serverless computing framework for GNN serving, which achieves resource efficiency through graph sharing and fine-grained resource allocation. Volley [54] employs an iterative optimization algorithm to adjust data placement based on data access patterns and the geographical locations of clients. Yugong [28] optimizes bandwidth usage across Alibaba's geo-distributed data centers through strategic project placement, table replication, and job outsourcing. RAGraph introduces an edge migration strategy that adapts to the iterative nature and complex dependencies of the graph algorithms, taking into account contribution metrics and network heterogeneity. This further achieves an average system performance improvement of 1.74× speedup and a 35% reduction in WAN cost.

VIII. Conclusion

We design and implement RAGraph, which consists of a Region-Aware framework, an edge migration algorithm, and two runtime optimizations for geo-distributed graph processing. First, we design a Region-Aware framework based on three helpful observations: the ping-pong effect optimization for accelerating inefficient global updates, a two-layer view for coordination-free message interaction, and a replaceable communication strategy for network congestion. Additionally, we introduce an edge migration algorithm based on vertex contribution to effectively utilize network resources. Furthermore, we develop the adaptive hierarchical message interaction, allowing RAGraph to adaptively choose between two message interaction modes based on network status and message traffic. Finally, we propose a discrepancy-aware message filtering strategy to adaptively filter important messages in a discrepancy range of messages.

References

[1] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 301–316.

[2] R. Chen, J. Shi, Y. Chen, B. Zang, H. Guan, and H. Chen, "PowerLyra: Differentiated graph computation and partitioning on skewed graphs," *ACM Trans. Parallel Comput.*, vol. 5, no. 3, pp. 13:1–13:39, 2018.

[3] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 17–30.

[4] G. Malewicz et al., "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.

[5] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," in *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[6] W. Fan et al., "Parallelizing sequential graph computations," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 1889–1892.

[7] W. Fan et al., "Parallelizing sequential graph computations," *ACM Trans. Database Syst.*, vol. 43, no. 4, pp. 18:1–18:39, 2018.

[8] "Facebook daily active users (DAUS)," 2021. [Online]. Available: https://investor.fb.com/investor-events/event-details/2021/Facebook-Q2--2021-Earnings/default.aspx

[9] Y. Yuan, D. Ma, Z. Wen, Z. Zhang, and G. Wang, "Subgraph matching over graph federation," in *Proc. VLDB Endowment*, vol. 15, no. 3, pp. 437–450, 2021.

[10] Q. Pu et al., "Low latency GEO-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.

[11] A. Rabkin, M. Arye, S. Sen, V. Pai, and M. J. Freedman, "Making every bit count in {Wide-Area} analytics," in *Proc. 14th Workshop Hot Topics Operating Syst.*, 2013, pp. 1–6.

[12] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (GDPR)," in *A Practical Guide*, vol. 10, 1st ed. Berlin, Germany: Springer, 2017, pp. 10–5555.

[13] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proc. Int. Conf. World Wide Web*, 2011, pp. 587–596.

[14] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2091–2100, Aug. 2014.

[15] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[16] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Monarch: Gaining command on GEO-distributed graph analytics," in *Proc. 10th USENIX Conf. Hot Topics Cloud Comput.*, 2018, Art. no. 5.

[17] Y. Yuan, D. Ma, Z. Wen, Y. Ma, G. Wang, and L. Chen, "Efficient graph query processing over GEO-distributed datacenters," in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 619–628.

[18] F. Yao et al., "RAGraph: A region-aware framework for GEO-distributed graph processing," in *Proc. VLDB Endowment*, vol. 17, no. 3, pp. 264–277, 2024.

[19] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 25:1–25:39, 2015.

[20] Q. Wang et al., "Automating incremental and asynchronous evaluation for recursive aggregate data processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2439–2454.

[21] S. Gong et al., "Automating incremental graph processing with flexible memoization," in *Proc. VLDB Endowment*, vol. 14, no. 9, pp. 1613–1625, 2021.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[23] D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," in *Graph Theory and Computing*, San Francisco, CA, USA: Academic Press, 1972, pp. 109–122.

[24] M. Al Hasan and V. S. Dave, "Triangle counting in large networks: A review," *Wiley Interdiscipl. Rev.: Data Mining Knowl. Discov.*, vol. 8, no. 2, 2018, Art. no. e1226.

[25] X. Tao, K. Ota, M. Dong, W. Borjigin, H. Qi, and K. Li, "Congestion-aware traffic allocation for geo-distributed data centers," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1675–1687, Jul.-Aug., 2022.

[26] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Proc. Stabilization, Saf. Secur. Distrib. Syst.: 13th Int. Symp.*, 2011, pp. 386–400.

[27] P. S. Almeida, A. Shoker, and C. Baquero, "Efficient state-based crdts by delta-mutation," in *Proc. Int. Conf. Netw. Syst.*, 2016, pp. 62–76.

[28] Y. Huang et al., "Yugong: Geo-distributed data and job placement at scale," in *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2155–2169, 2019.

[29] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Priter: A distributed framework for prioritized iterative computations," in *Proc. Proc. 2nd ACM Symp. Cloud Comput.*, 2011, pp. 1–14.

[30] Y. Zhang, X. Liao, H. Jin, L. Gu, G. Tan, and B. B. Zhou, "Hotgraph: Efficient asynchronous processing for real-world graphs," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 799–809, May 2017.

[31] P. Boldi and S. Vigna, "The webgraph framework I: Compression techniques," in *Proc. Int. Conf. World Wide Web*, 2004, pp. 595–602.

[32] "Web-Google," 2002. [Online]. Available: https://www.cise.ufl.edu/research/sparse/matrices/SNAP/web-Google.html

[33] K. Bogdanov, M. P. Quirós, G. Q. M. Jr, and D. Kostic, "Toward automated testing of Geo-distributed replica selection algorithms," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2015, pp. 89–90.

[34] "Arabic-2005," 2015. [Online]. Available: https://law.di.unimi.it/webdata/arabic-2005/

[35] "Uk-2005," 2005. [Online]. Available: https://law.di.unimi.it/webdata/uk-2005/

[36] Y. Wu, R. Jin, and X. Zhang, "Fast and unified local search for random walk based k-nearest-neighbor query in large graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1139–1150.

[37] V. T. Chakaravarthy, F. Checconi, P. Murali, F. Petrini, and Y. Sabharwal, "Scalable single source shortest path algorithms for massively parallel systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2031–2045, Jul. 2017.

[38] T. Hsu, V. Ramachandran, and N. Dean, "Parallel implementation of algorithms for finding connected components in graphs," in *Proc. Parallel Algorithms, DIMACS Workshop*, 1994, vol. 30, pp. 23–41.

[39] "libgrape-lite," 2020. [Online]. Available: https://github.com/alibaba/libgrape-lite

[40] W. Fan et al., "Adaptive asynchronous parallelization of graph algorithms," *ACM Trans. Database Syst.*, vol. 45, no. 2, pp. 1–45, 2020.

[41] C. Xie, R. Chen, H. Guan, B. Zang, and H. Chen, "SYNC or ASYNC: Time to fuse for distributed graph-parallel computation," in *Proc. 20th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2015, pp. 194–204.

[42] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proc. 24th ACM Symp. Operating Syst. Princ.*, 2013, pp. 456–471.

[43] L. Wang et al., "Lazygraph: Lazy data coherency for replicas in distributed graph-parallel computation," *ACM SIGPLAN Notices*, vol. 53, pp. 276–289, 2018.

[44] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and X. Du, "POCLib: A high-performance framework for enabling near orthogonal processing on compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 459–475, Feb. 2022.

[45] G. Feng et al., "RisGraph: A real-time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions ops/s," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 513–527.

[46] K. Vora, R. Gupta, and G. Xu, "KickStarter: Fast and accurate computations on streaming graphs via trimmed approximations," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 237–251.

[47] Y. Zhou et al., "Fast iterative graph computing with updated neighbor states," in *Proc. IEEE 40th Int. Conf. Data Eng.*, 2024, pp. 2449–2462.

[48] P. A. Costa, X. Bai, F. M. Ramos, and M. Correia, "Medusa: An efficient cloud fault-tolerant MapReduce," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2016, pp. 443–452.

[49] M. W. Convolbo, J. Chou, C.-H. Hsu, and Y. C. Chung, "GEODIS: Towards the optimization of data locality-aware job scheduling in geo-distributed data centers," *Computing*, vol. 100, pp. 21–46, 2018.

[50] L. Zhao, Y. Yang, A. Munir, A. X. Liu, Y. Li, and W. Qu, "Optimizing Geo-distributed data analytics with coordinated task scheduling and routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 279–293, Feb. 2019.

[51] C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proc. Proc. 13th Eur. Conf.*, 2018, pp. 1–16.

[52] Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, and H. Jin, "Carbon-aware online control of GEO-distributed cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2506–2519, Sep. 2016.

[53] H. Hu, F. Liu, Q. Pei, Y. Yuan, Z. Xu, and L. Wang, "λGrapher: A resource-efficient serverless system for GNN serving through graph sharing," in *Proc. Int. Conf. World Wide Web*, 2024, pp. 2826–2835.

[54] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, Art. no. 2.

Feng Yao received the MS degree in computer science from Northeastern University, China, in 2021. He is currently working toward the PhD degree in computer science from Northeastern University, China. His research interests include cloud computing and distributed graph processing.

Qian Tao recieved the PhD degree in computer science and technology from Beihang University, China, in 2021. He is currently an engineer in Alibaba Group, China. His research interests include graph neural networks, graph computations, and large language models.

Shengyuan Lin is currently working toward the undergraduation degree majoring in computer science with Northeastern University, China. His research interests include distributed and parallel computation, distributed graph processing.

Yanfeng Zhang received the PhD degree in computer science from Northeastern University, China, in 2012. He is currently a professor with Northeastern University, China. His research consists of distributed systems and big data processing. He has published many papers in the above areas. His paper in SoCC 2011 was honored with "Paper of Distinction".

Wenyuan Yu received the PhD degree from the University of Edinburgh. He is a senior staff engineer and director with Alibaba Group. At Alibaba, he leads the Fusion Computing team, the Institute for Intelligent Computing, focusing on machine learning systems and graph computing. He is the founder and project lead of GraphScope, Alibaba's open-source large-scale graph computing system, and the CNCF's data sharing system, Vineyard. His research, published in top-tier international conferences and journals, has earned him recognition including Best Paper at SIGMOD 2017 and VLDB 2010, and the SIGMOD Research Highlight Award, in 2018. Prior to Alibaba, Wenyuan was a founding member of 7Bridges Ltd. and a Research Scientist at Facebook.

Shufeng Gong (Associate Member, IEEE) received the PhD degree in computer science from Northeastern University, China, in 2021. He is currently a lecturer with Northeastern University, China. His research interests include cloud computing, distributed graph processing, and data mining.

Qiange Wang received the PhD degree in computer science from Northeastern University, China, in 2022. He is currently working toward the postdoctoral research fellow with the National University of Singapore. His research interests include distributed graph processing, learning, and management systems.

Ge Yu (Senior Member, IEEE) received the PhD degree in computer science from the Kyushu University of Japan, in 1996. He is now a professor with Northeastern University, China. His current research interests include distributed and parallel systems, cloud computing, Big Data management, and blockchain techniques and systems. He has published more than 200 papers in refereed journals and conferences. He is the CCF fellow and the ACM member.

Jingren Zhou (Fellow, IEEE) is chief technology officer with Alibaba Cloud, where he spearheads cutting-edge technology innovation and product development. Prior to this role, he led the development of advanced techniques for personalized search, product recommendation, and advertisement with Alibaba's e-commerce platform and Alipay's online payment platform. He is an esteemed researcher in cloud-computing, databases, and large-scale machine learning. He has served as committee chairs for many prestigious academic conferences and published more than 100 papers in renowned journals and conferences. He received his PhD in Computer Science from Columbia University.