RAGraph: A Region-Aware Framework for Geo-Distributed Graph Processing

Feng Yao
Northeastern Univ., China
yaofeng@stumail.neu.edu.cn

Qian Tao
Alibaba Group, China
qian.tao@alibaba-inc.com

Wenyuan Yu
Alibaba Group, China
wenyuan.ywy@alibaba-inc.com

Yanfeng Zhang
Northeastern Univ., China
zhangyf@mail.neu.edu.cn

Shufeng Gong
Northeastern Univ., China
gongsf@mail.neu.edu.cn

Qiange Wang
Northeastern Univ., China
wangqiange@stumail.neu.edu.cn

Ge Yu
Northeastern Univ., China
yuge@mail.neu.edu.cn

Jingren Zhou
Alibaba Group, China
jingren.zhou@alibaba-inc.com

ABSTRACT

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. 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. Finally, the experiments show that RAGraph can achieve 1.69× - 40.53× speedup and 20.9% - 97% WAN cost reduction compared with state-of-the-art systems.

PVLDB Reference Format:
Feng Yao, Qian Tao, Wenyuan Yu, Yanfeng Zhang, Shufeng Gong, Qiange Wang, Ge Yu, and Jingren Zhou. RAGraph: A Region-Aware Framework for Geo-Distributed Graph Processing. PVLDB, 17(3): 264 - 277, 2023.
doi:10.14778/3632093.3632094

PVLDB Artifact Availability:
The source code of this research paper has been made publicly available at https://www.github.com/farisyao/RAGraph.

1 INTRODUCTION

Iterative graph processing has been regarded as a significant paradigm in many fields. Recently, with the size of graph-structured data blowing up promptly, research efforts have been made to extend graph processing to distributed environments to handle computation over large-scale graphs, and have made great progress from both theory perspective and system perspective [19, 24, 25, 30, 44, 45, 78]. These works have significantly enhanced the processing of iterative graph algorithms from many aspects, such as graph partition, processing model design, parallel algorithm design, *etc.*

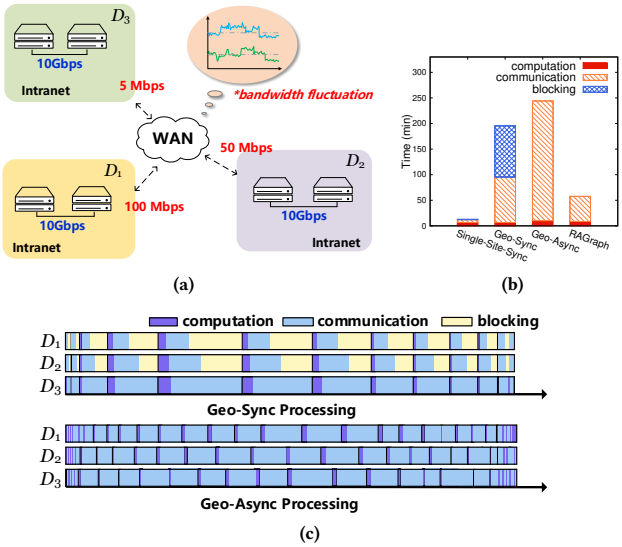


Figure 1: An example of geo-distributed graph processing. (a) The bandwidth of geo-distributed networks; (b) performance of geo-distributed and single-site iterative graph processing; (c) performance breakdown of sync/async parallel processing modes on geo-distributed networks.

Unfortunately, most of the existing frameworks assume that the graph-structured data is distributed to multiple machines within a single-site data center, which is 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 [5]. Another typical application with geographically distributed graphs is federated graph computation [10, 72], 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

Proceedings of the VLDB Endowment, Vol. 17, No. 3 ISSN 2150-8097.
doi:10.14778/3632093.3632094

connected by Wide Area Networks (WANs), which results in scarce and heterogeneous network bandwidth [51, 52]. The increasing data protection requirements [62] also make redistributing data among data centers impossible. Therefore, traditional graph partitioning methods [15, 42, 60] and parallel processing model designs [24, 74] are 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: Figure 1a 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 [11]. Traditional distributed graph processing systems treat the worker as peer 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 [12] on two clusters with different configurations. The first cluster consists of 24 AliCloud ECS instances (8vCPU and 64GB memory), all located within a single-site data center. These instances are interconnected via a 10Gbps Ethernet network. The second cluster consists of 24 ECS instances with identical configurations, and the instances are deployed across three geo-distributed data centers. Within each data center, there are 8 ECS instances connected via a 10Gbps local Ethernet network. The three data centers are interconnected by WANs. Figure 1a illustrates the bidirectional bandwidths of each data center. We test a state-of-the-art synchronous parallel system, GRAPE [24], on both clusters (*i.e.,* Single-Site-Sync and Geo-Sync) and, additionally, an advanced asynchronous parallel processing system, Maiter [74], on the geo-distributed cluster (*i.e.,* Geo-Async). The overall computation, communication, and blocking time are reported in Figure 1b. Figure 1c 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, which the inefficient and heterogeneous WANs cause. □

Example 1 raises 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 Figure 1b. 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 [61], require coordinated computation (*i.e.,* local) and communication (*i.e.,* global) among vertices in each iteration in order to simplify the parallel semantic [45]. 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 Figure 1c, 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 [44, 74] 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 the various graph processing systems, Monarch [39], GeoGraph [71], and PGPregel [77] 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. On the other hand, PGPregel focuses on efficient geo-distributed privacy protection by integrating differential privacy into graph processing.

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. 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 buckets 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.

Region-Aware framework and runtime optimizations can improve efficiency and guarantee correctness for a considerable scope of iterative graph algorithms aligned with the *monotonic property*. To integrate all these effective methods together, we have developed a new geo-distributed graph processing system, called RAGraph. To summarize, we make the following contributions:

- **Region-Aware Graph Processing Framework.** We design a Region-Aware framework for geo-distributed graph processing based on three helpful observations. By employing proxy, the framework allows advancing inefficient global updates locally, enables coordination-free message interaction, and implements replacement communication on congested networks.
- **Adaptive Hierarchical Message Interaction Engine.** Following the Region-Aware graph processing framework, we design and implement a geo-distributed message interaction engine that can guide global message interaction in eager/lazy modes and adaptively switch between the two modes to adapt to complicated inter-region networks.
- **Adaptive Message Filtering Strategy.** We filter globally transmitted discrepant messages to reduce communication cost and design and implement an adaptive buckets structure to adjust the filtering range of important messages adaptively.

2 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.

Geo-Distributed Graphs. Given a graph $G = (V, E, C)$, a *geo-distributed partition* of G, denoted by $S = \{G_1, G_2, ..., G_h\}$ with $G_i = (V_i, E_i, C_i)$, is a set of induced subgraphs of G. satisfying that: (i) $\cup_{i=1}^{h} E_i = E$; and (ii) $E_i \cap E_j = \emptyset$ for $\forall i \neq j$. If a node $v \in G_i$ has edges connected to another subgraph G_j $(j \neq i)$, v is defined as a *boundary vertex*. *W.l.o.g.*, we assume data centers $\{D_1, ..., D_h\}$ are geo-distributed and data center D_i stores the subgraph G_i.

Vertex-Centric Model. Many graph processing systems adopt the "think like a vertex" programming model [47], which uses a vertex-centric program $\mathcal{P}$ to abstract graph algorithm operators for the input graph G. $\mathcal{P}$ is executed on each vertex iteratively, and the program executed on v, say $\mathcal{P}_v$, will interact with the programs on v's neighbors in each iteration. $\mathcal{P}$ is performed for iterations 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 i-th 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 $\mathsf{N}_{out}(v) = \{w \mid e(v, w) \in E\}$.

Table 1: 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 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 Equation (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*.

Monotonic Conditions. $\mathcal{A}$ and $\mathcal{I}$ 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 [66, 74] 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 [28]. Table 1 summarizes some typical iterative graph algorithms that satisfy the monotonic property.

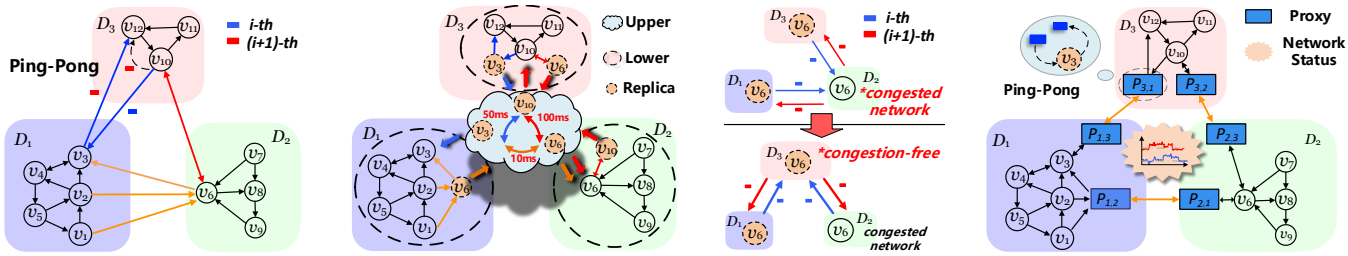
The monotonic property makes it possible for more flexible iterative processing and has been regarded in traditional graph analysis [24, 26, 29, 54, 63, 66, 74]. This paper focuses on algorithms with the monotonic property in geo-distributed environments. Henceforth, we denote the set of *partial messages* from the element group $*$ as M_*^i and use $\mathcal{A}$ and $\mathcal{U}$ interchangeably.

Example 2: We take an iterative graph algorithm, delta-based PageRank [74], 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)**. Intuitively, vertex v can use messages from part of neighbors to complete one iterative operation, which is friendly to eliminating coordinated operations.

□



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

Figure 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.

3 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, and ends with the theoretical analysis.

3.1 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 Figure 1a, a sample graph is given with 12 vertices distributed, as shown in Figure 2a, of which 6 are boundary vertices through whose messages sent to other data centers will suffer from inefficient and heterogeneous WANs. In comparison, the subgraph within the data center is executed on efficient LANs.

Observation 1: Ping-Pong Effect. Consider the execution of PageRank at the blue arrows in Figure 2a. 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 occurs in other communication-intensive boundary structures. As the red bidirectional arrow in Figure 2a 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 first aggregate the messages from v_1 and v_2 locally, *i.e.,* $m_{(*,6)} = \text{sum}(m_{(1,6)}, m_{(2,6)})$, and then locally generate the message $m_{(6,3)}$ in advance using the aggregated message $m_{(*,6)}$ for the subsequent computation of v_3.

Observation 2: Inefficient Local-Global 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. Specifically, all workers in each iteration coordinated perform global message interaction (*e.g.,* BSP model [61]), limited by the imbalanced global network resulting in coordinated waiting. 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 Figure 2b, 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.,* $d \times \text{sum}(M_{*,v})/N_v$, 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 coordinating with other parties.

Observation 3: Replica Replaceable Communication. In practice, the bandwidth between data centers is allocated based on typical or average usage rather than peak usage [59], 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, the time of which fluctuates from hundreds of milliseconds to seconds or more [16], making transmission inefficient.

Based on the message passing of v_6 in the upper layer of Figure 2b with the original vertex v_6 in D_2 and replicas of v_6 in D_1 and D_3. Consider the typical communication pattern shown in the upper part of Figure 2c, and still take PageRank as an example. Both replicas of v_6 send messages to original v_6 in D_2, then original v_6 aggregates the messages from both replicas and local neighbors, *i.e.,* $x_{v_6} = \text{sum}\{m_{(D_1,v_6)}, m_{(D_2,v_6)}, m_{(D_3,v_6)}\}$, 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 the network links of D_2.

Intuitively, as long as any replica of v_6 knows the v_6's out-degree N_{v_6}, the aggregation (*e.g.,* sum$\{m_{(D_1,v_6)}, m_{(D_2,v_6)}, m_{(D_3,v_6)}\}$) and generation (*e.g.,* $d \times x_{v_6}/N_{v_6}$) of messages can be performed. So we can find a substitute to share the congested communication. As shown in the lower part of Figure 2c, 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 centrally handle communications and message interaction from v_6 in D_1 and D_2.

3.2 Region-Aware Message Management

The observations in Section 3.1 inspire us to design a *Region-Aware* message management framework. Figure 2d illustrates the structure of the Region-Aware framework. 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 $v \in V_k$:

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

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

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

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

$$m_{v,w}^i = \mathcal{I}\Big(\mathcal{A}\big(lx_v^i, \{m_{u,v}^{i-1} \mid u \in V \setminus V_l\}\big), C_E(v,w)\Big) \text{ for } w \in V_l (l \neq k). \tag{6}$$

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

$$m_{P_k,w}^i = \overline{\mathcal{I}}\Big(\mathcal{A}\big(\{m_{v,w}^i \mid v \in V_k\}\big)\Big) \text{ for } w \in V_l, \tag{7}$$

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

For any vertex v, in each iteration, v in data center D_k first aggregates the messages received from its local neighbors (Equation (2)) and other data centers (Equation (3)), then updates its state in the i-th round based on the aggregation results (Equation (4)). It finally generates messages to its local neighbors (Equation (5)) and remote neighbors (Equation (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 (Equation (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 $\overline{\mathcal{I}}$ (Equation (7)) without coordinating with other proxies. At the lower layer, considering

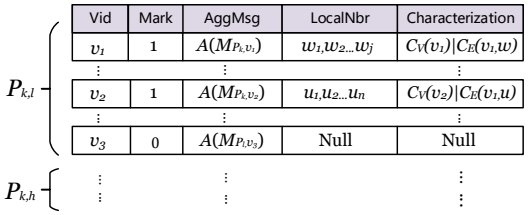


Figure 3: InterTable structure for proxy.

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 $\mathcal{I}$ (Equation (8)).

Design of Proxy. Multiple proxies are deployed in each data center, which play a key role in Region-Aware framework. Each proxy maintains a table for region-aware interaction, called *InterTable*. The *InterTable* for $P_{k,l}$ maintains the necessary information for local and remote boundary vertices between data centers D_k and D_l. As shown in Figure 3, each row maintains the cached information for a vertex identified by `Vid`. The `Mark` column marks whether the vertex is outside (*resp.* inside) the data center with value 1 (*resp.* 0). The column `AggMsg` is used for caching the aggregated messages. For example, when messages targeted at a remote vertex v_1, *i.e.,* M_{P_k,v_1}, they are aggregated and cached in the `AggMsg` column before being sent to the remote data center, which is illustrated in Equation (7). For the incoming message, when a message targeted at a local vertex v_3 is received from data center D_l, the proxy $P_{k,l}$ first caches the message in column `AggMsg`, and then v_3 will pull the cached messages from $P_{k,l}$ to avoid multi-proxy write conflicts to v_3. For the rows associated with remote vertices, a `LocalNbr` column storing the neighbors located in D_k and a column `Characterization` storing the vertex or edge associated property are designed, which are used to optimize the computation of the ping-pong effect as shown in Equation (8).

Design of Communication Module. All replicas can directly communicate with the original vertex using Equation (7) at the upper layer through proxies. In order to implement the replaceable communication described in Section 3.1, we need to get the global network status and redirect the proxy's communication to specify where the new "original" vertex is. Specifically, the proxy monitors the current network status and shares global information between the proxies on a time window ΔT. When congestion occurs on D_k, the data center with the lowest average round-trip delays for that time interval, say D_l, will take over the corresponding task based on the replicas associated with D_k. Accordingly, the `Mark` of the replicas in the InterTable is modified to 0 (*i.e.,* inside vertex) on $P_{l,k}$, and then performed as a local vertex. Note that subject to the shared global network status, all proxies know whom to send messages to without coordination. The proxies only need to adjust the communication route from $P_{*,k}$ to $P_{*,l}$.

Example 3: The delta-based PageRank algorithm in the Region-Aware framework can be implemented as follows:

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

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.

□

3.3 Theoretical Analysis

This subsection provides a theoretical analysis for the proper execution of the Region-Aware framework.

Conditional Equivalence of the Ping-Pong Effect. We illustrate that optimizing the ping-pong effect is conditionally equivalent to vertex-centric synchronous processing. Following the vertex-centric model, *i.e.,* Equation (1), we simplify $\mathcal{I}(x_v^i, C_E(v,w))$ as $\mathcal{I}(x_v^i)$ and reorganize $\mathcal{I}$ in a monotonic fashion:

$$m_{v,w}^i = \mathcal{I}(\mathcal{A}(M_v^{i-1})) \tag{9}$$

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

$$= \mathcal{A}\big(\cup_{k=1}^{j} \mathcal{I}(m_{k,v}^{i-1})\big) \tag{11}$$

where $x_v^i = \mathcal{A}(M_v^{i-1})$ and $M_v^{i-1} = \cup_{k=1}^{j} m_{k,v}^{i-1}$. Equation (11) is true when obeying condition (**C2**), and it indicates that $\mathcal{I}$ is allowed to be applied to partial messages independently. Thus, the ping-pong effect can be optimized locally using local partial messages. If condition (**C1**) holds, the complete message $m_{v,w}^i$ of the i-th iteration can be obtained by splicing the partial messages $\mathcal{I}(m_{k,v}^{i-1})$ using $\mathcal{A}$. Additionally, the vertex state can be updated using $\mathcal{A}$ monotonically applying partial results from $\mathcal{I}$. In summary, the ping-pong effect optimization can obtain the same results as the vertex-centric synchronous processing under monotonic conditions.

Feasibility of Coordination-free Interaction. Coordinating the consistency among data centers with heterogeneous networks is inefficient, and a 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) [7, 55] 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:

$$s_v^i = \mathcal{A}(s_v^{i-1}, \mathcal{A}(M_v^{i-1})) \tag{12}$$

$$= \mathcal{A}\big(s_v^{i-1}, \cup_{k=1}^{j} m_{k,v}^{i-1}\big) \tag{13}$$

$$= \mathcal{A}(s_v^{i-1} \cup \mathcal{I}(x_{1,v}^{i-1}) \cup \ldots \cup \mathcal{I}(x_{j,v}^{i-1})) \tag{14}$$

where $m_{k,v}^{i-1} = \mathcal{I}(x_{k,v}^{i-1})$. Following the conditions (**C1**) and (**C2**), Equation (12) can be reorganized to obtain Equation (14). Equation (14) 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:

$$\begin{aligned} s^n =& \mathcal{A}\Big(s^{n-1}, \mathcal{A}(M^{n-1})\Big) \\ =& \mathcal{A}\Big(\mathcal{A}(s^{n-2}, \mathcal{A}(M^{n-2})) \cup \mathcal{A} \circ \mathcal{I}(\mathcal{A}(M^{n-2}))\Big) \\ =& \mathcal{A}\Big(s^0 \cup (\mathcal{A} \circ \mathcal{I})(M^0) \cup \ldots \cup (\mathcal{A} \circ \mathcal{I})^n(M^0)\Big) \end{aligned} \tag{15}$$

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.,* $s^n = \mathcal{A}((s^0 \cup (\mathcal{A} \circ \mathcal{I})(\cup_j m^0) \cup \ldots \cup (\mathcal{A} \circ \mathcal{I})^n(\cup_j m^0))$). 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 in Section 2, we know that the result of the complete message obtained by vertex v in round i-th is $\mathcal{A}(\{m_{u,v}^{i-1} \mid e(u,v) \in E\})$. Based on Equation (2) and (3), the intermediate result of v in the i-th 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. □

Based on the above analysis, the Region-Aware framework satisfies the three conditions of δ-CRDT based on monotonic property guaranteeing coordination-free interactions on the replicas. It also indicates the capabilities' equivalence of all replicas, which provides theoretical support for replica replaceable communication for Observation 3 in Section3.1.

Correctness of Two-layer Message Interaction. In the Region-Aware framework, two-layer coordination-free message interaction view can be seen as message passing at different paces between the lower layer (*i.e.,* local messages) and the upper layer (*i.e.,* global messages). The following demonstrates the convergence of two-layer message interaction under the overall view.

Divide local and global messages into consecutive time periods in asynchronous paces and define $M^t = (lx \cup gx)^t$ as the set of messages received from the two layers at the t-th time period. For asynchronous message passing, it will converge to:

$$R_{async} = \mathcal{A}\big(X_{init} \cup \mathcal{I} \circ \mathcal{A}(M^0) \cup \ldots \cup \mathcal{I} \circ \mathcal{A}(M^\infty)\big) \tag{16}$$

where X_{init} represents the initial states of the vertices. Define $M^{t,q}$ as the set of messages that have passed through a q-hop path in M^t and we have $M^t = \cup_{q=0}^{\infty} M^{t,q}$. Then, we have:

$$R_{async} = \mathcal{A}(X_{init} \cup \cup_{t=0}^{\infty} \mathcal{A}(\cup_{q=0}^{\infty} \mathcal{I} \circ \mathcal{A}(M^{t,q}))). \tag{17}$$

If the monotonic conditions (**C1**), (**C2**) hold, we reorganize the messages according to the number of hops:

$$R_{async} = \mathcal{A}(X_{init} \cup \cup_{q=0}^{\infty} \mathcal{A}(\cup_{t=0}^{\infty} \mathcal{I} \circ \mathcal{A}(M^{t,q}))). \tag{18}$$

Notice that $\mathcal{A}(\cup_{t=0}^{\infty} \mathcal{I} \circ \mathcal{A}(M^{t,q}))$ is exactly the intermediate result by the messages passing through q hops. We have shown that the value of a message varies from m to $\mathcal{I}(m)$ every time it goes through one more path (no matter the lower layer or the upper layer message passing). This means $\mathcal{A}(\cup_{t=0}^{\infty} \mathcal{I} \circ \mathcal{A}(M^{t,q}))$ is exactly the complete result at q-th round in synchronous paces. Thus, the Region-Aware framework can converge correctly in finite time under two-layer message interaction.

The intuition behind our discussion is that the two-layer interaction view can treat local and remote neighbor messages independently. Thus, the vertex may obtain only partial results on different hops at different moments. Still, in a finite time, by splicing the complete messages on all hops, the vertex can obtain the same results as synchronization.

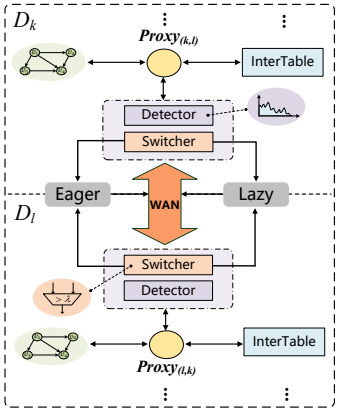


Figure 4: Adaptive hierarchical interaction engine structure.

RAGraph Application Scope. RAGraph focuses on iterative graph algorithms aligned with the monotonic property. Besides the examples listed in Table 1, some classic iterative graph algorithms are well-supported, such as MinimalSpanningTree [32], FacilityLocation [49], WidestPath [8], and Min/Max Label Propagation [27], *etc.* On the contrary, some graph algorithms do not obey the monotonic property. For example, GCN-Forward [41], whose aggregation function $\mathcal{A}$ is sum, satisfies condition **(C1)**, but the activation functions like ReLU, Sigmoid, *etc.*, as interaction functions do not satisfy condition **(C2)**. In addition, Graph Coloring [46], Triangle Counting [6], *etc.*, require the complete neighbor messages to be obtained simultaneously and do not satisfy the monotonic conditions. In this regard, RAGraph can support their correct execution with a generic vertex-centric model.

4 HETEROGENEOUS-AWARE MESSAGE PASSING MANAGEMENT

This section presents two important runtime optimizations based on the Region-Aware framework, *i.e.,* the adaptive hierarchical message interaction and discrepancy-aware message filtering strategy.

4.1 Adaptive Hierarchical Message Interaction

RAGraph adopts the two-layer message interaction view to cast off the coordination overhead. To adapt to the heterogeneous and fluctuating networks in geo-distributed environments, we design an adaptive hierarchical message interaction engine for the upper-lower layer, which takes into account the network status on the basis of coordination-free message interaction. 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 scarcer than that of LANs and highly heterogeneous. As a result, the commonly adopted real-time message passing in the asynchronous model will generate frequent cross-datacenter communication and cause an intolerant overhead. In contrast, an alternative way is to focus on the computation in the lower-layer subgraphs 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.

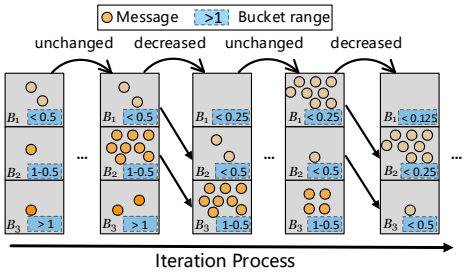


Figure 5: Buckets for message filtering. Cycle: cached outward message (the darker the color, the larger the values).

Second, the data transmission rate between data centers *fluctuates significantly* [11, 35]. 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 computed. 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 record in *InterTable*. During the execution, each proxy counts the average transmission data size $S_{\delta t}$ and average network transmission rate $R_{\delta t}$ in the time window ΔT, and adaptively selects the message interaction mode based on their ratio: if $S_{\delta t}/R_{\delta t} < \lambda \cdot \mu/\tau$, 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 Figure 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 $R_{\delta t}$, 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.

Algorithm 1: 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** $|B_3| \le \gamma \sum_{i=1}^{3} |B_i|$ **then**
5 **if** $|B_2| \ge \sigma |B_1|$ **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 δB_1, δB_2;

4.2 Discrepancy-Aware Message Filtering

Due to the skewed distribution of practical graph structures [29, 73] and heterogeneity of the WAN networks, the values of messages are highly discrepant: (1) different vertices generate messages with different importance. The larger the change in the vertex state, the more possibly it produces *significant updates*; (2) due to imbalanced communications, a vertex may receive discrepant messages generated at different iterations, leading to inefficient communications.

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 1 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

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

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. Figure 5 illustrates the process of the strategy in a proxy.

Detection of Shifting Distribution. We may encounter a situation that $|B_3| \le \gamma \sum_{i=1}^{3} |B_i|$ 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 1) to avoid shifting distribution from the decrease of ranges.

5 SYSTEM

We design and implement RAGraph, a graph processing system for geo-distributed environments. It is developed based on libgrape-lite [4], an open-source version of GRAPE [24]. Below we discuss the implementation details of distinctive components.

Data Preprocessing. Graph data is preprocessed to fit the Region-Aware framework before the computation task starts. RAGraph utilizes the CSC/CSR format to classify and store interior subgraph information and constructs *InterTable* in proxy to quickly distinguish the roles of vertices. During loading data, each data center loads the local subgraph and globally shares the characterizations cached in *InterTable* for subsequent computation by proxy. For example, PageRank needs to provide the vertex's out-degree for the proxy to compute the optimization of the ping-pong effect.

Homomorphic Encryption. RAGraph provides homomorphic encryption (HE) [70] interfaces to protect the users' data from other parties. To get the results without leaking each party's original messages and graph topology, HE allows a third party to compute on encrypted data without knowing the explicit values in advance. We use an open-source software library HElib [33] to perform RAGraph's homomorphic encryption operators. The Region-Aware framework enables each proxy to uniformly manage the message interaction on one party's boundary, which well supports the execution of HE. RAGraph's runtime optimizations on cross-datacenter messages in Section 4 can reduce the number of transmission messages, thus reducing the computation cost caused by HE.

Termination Checker. The termination check module is executed in the lazy interaction mode detection phase and global convergence. A convergence threshold Θ determines if the vertex states are close enough to the stable states. The module checks the convergence of each data center using a dedicated thread. Since each data center executes without coordination, the module uses the *AllReduce* operation to perform global synchronous statistics and distribute the results to each data center.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Datasets and Test Algorithms. We use five real-world datasets (see Table 2) in our experiments, including Web-Google [1], Enwiki-2013 [13], Arabic-2005 [2], UK-2005 [3], and Twitter-2010 [12]. 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 [74], Penalized Hitting Probability (PHP) [67], Single Source Shortest Path (SSSP) [14] and Connected Components (CC) [37].

Table 2: Dataset Description

Graph	Vertices	Edges	Abbreviation
Web-Google [1]	916,428	6,078,250	GL
Enwiki-2013 [13]	4,203,323	101,311,614	WK
Arabic-2005 [2]	22,744,080	639,999,458	AB
UK-2005 [3]	39,459,925	936,364,282	UK
Twitter-2010 [12]	41,652,230	1,468,364,884	TW

Competitors. We compare RAGraph with a representative distributed graph processing system, GRAPE [24], and two state-of-the-art geo-distributed graph processing systems, Monarch [39] and GeoGraph [71]. All competitors and the corresponding test algorithms are implemented on top of libgrape-lite [4].

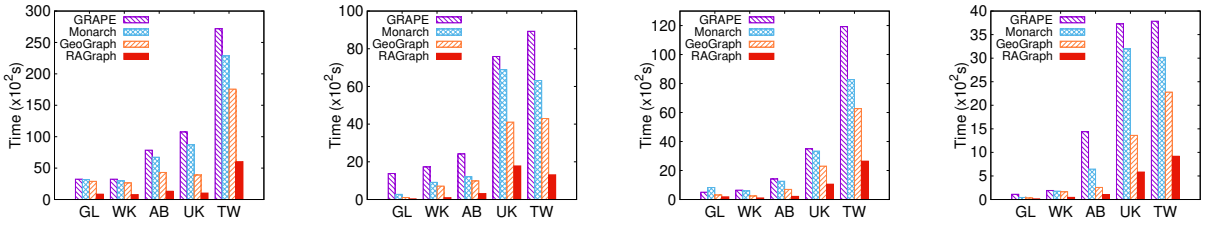
Environments. All algorithms are implemented in C++, and the average result of three runs is reported. AliCloud ECS clusters from five regions are chosen as geo-distributed data centers for evaluation, including Qingdao, China; Singapore; Sydney, Australia; Frankfurt, Germany; Virginia, USA. Each data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

6.2 Overall Performance

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

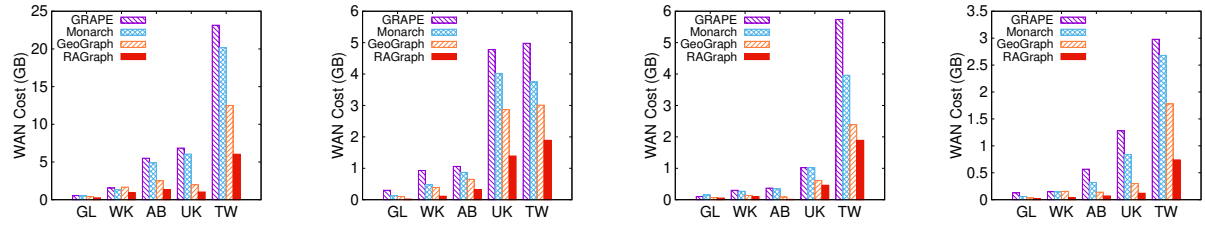
Running time. Figure 6 shows the running time of PageRank, PHP, SSSP, and CC algorithms in the compared systems. As can be seen from the results, RAGraph outperforms others in all cases. Specifically, RAGraph achieves 2.72× - 40.53× (8.13× on average) speedup over GRAPE, 2.26× - 9.31× (4.86× on average) speedup over Monarch, and 1.69× - 7.3× (2.97× 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. Notably, the gap between RAGraph and other competitors on PageRank and PHP is more significant than that on CC and SSSP. This is because PageRank and PHP require more iterations to reach convergence, and RAGraph can avoid the coordinated waiting times between iterations.

WAN cost. We measure the transmitted data size across data centers via WANs for each system. Figure 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 40.2% - 97% (73% on average) compared with GRAPE, 30% - 96.8% (67.8% on average) compared with Monarch, and 20.9% - 87.2% (49.6% on average) compared with GeoGraph. Besides, RAGraph has less improvement compared with GeoGraph on SSSP than on other algorithms. This is probably because, in SSSP, only a few important messages (those leading to a shorter distance) activate the update of the vertex state. Therefore, in SSSP, the proposed message filtering optimization is not as effective as other test algorithms.



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

Figure 6: Running time comparison.



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

Figure 7: WAN cost comparison.

6.3 Performance Gain Analysis

The performance of RAGraph mainly comes from the flexible Region-Aware framework and two runtime optimizations for heterogeneous-aware message passing. In this subsection, we quantitatively analyze the gain from the framework and the runtime optimizations. Specifically, we report the running time and WAN cost of the test algorithms by successively enabling RAGraph components, including Region-Aware framework, adaptive hierarchical message interaction, and discrepancy-aware message filtering, which we denote RA, RA +Hi, and RAGraph respectively. The results are compared with the traditional synchronous graph processing system libgrape-lite [4] and its asynchronous version modified based on Maiter [74], 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 Figure 8. The results for PHP and CC show a similar 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 Figure 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× - 2.06× speedup compared with Sync, and 1.56× - 3.8× speedup compared with Async. By further enabling the adaptive hierarchical message interaction, RA +Hi achieves 1.33× - 1.67× speedup and reduces 16% - 40.1% WAN cost compared with RA. Finally, the discrepancy-aware message filtering method lead to a 2.03× - 6.58× speedup and a 43.7% - 94.7% reduction in WAN cost. 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 1 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. This indicates that the Region-Aware framework can largely eliminate coordinated waiting times in Sync.

Figure 9 shows Sync, Async, and RAGraph with and without discrepancy-aware message filtering method in terms of WAN cost on the GL graph. The blue columns represent the improvement caused by the discrepancy-aware message filtering method. Specifically, the discrepancy-aware message filtering method reduces 20% - 25% WAN cost for Sync, 47% - 55% WAN cost for Async, and 55% - 59% WAN cost for RAGraph. RAGraph obtains a more significant gain from the discrepancy-aware message filtering method than Sync and Async. In addition, Async also has significant benefits, but is still limited by the frequent message transmission in execution.

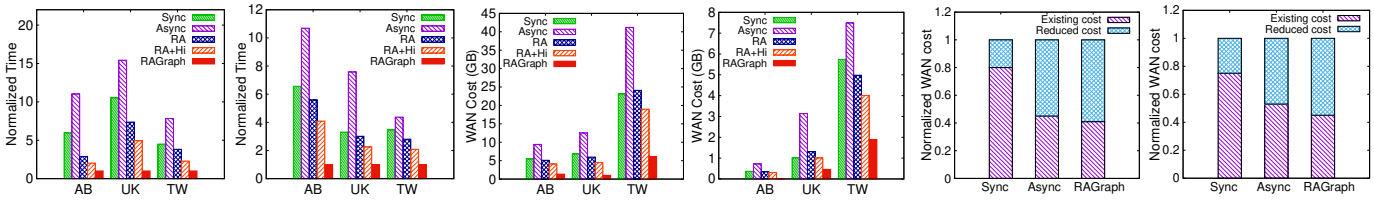
Table 3: Cost of Region-Aware Framework

Region-Aware Framework Applied/Not Applied		
Dataset	**Memory Usage**	**Computation Cost**
GL	2.02x	1.24x
WK	1.55x	1.15x
AB	1.37x	1.22x
UK	1.63x	1.17x
TW	1.41x	1.38x

6.4 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 in Singapore. The result is shown in Figure 10. We can see that the communication and blocking take up most of the running time. Compared with competitors, RAGraph eliminates the blocking time and generates the least communication.

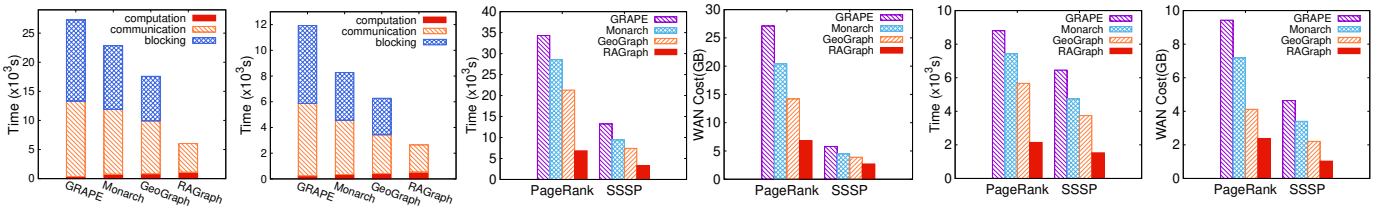
We further study the additional memory and computation cost in RAGraph, which comes from vertex replica maintenance on InterTable and ping-pong effect optimization. We run PageRank on all five graphs and record the memory usage and running time before and after applying the InterTable and Region-Aware optimizations. As reported in Table 3, InterTable and Region-Aware optimizations lead to an average of 1.59× memory usage and 1.23× computation cost, respectively. However, compared with the benefit of communication and blocking time reduction (as shown in Figure 10), the additional computation is lightweight. On the other hand, the memory requirement can be solved by adding instances as the computation resources are sufficient in geo-distributed data centers. In summary, it is worthwhile to adopt InterTable and Region-Aware optimizations in geo-distributed graph processing.



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

Figure 8: Performance gain from RAGraph.

Figure 9: Sensitivity to message filtering.

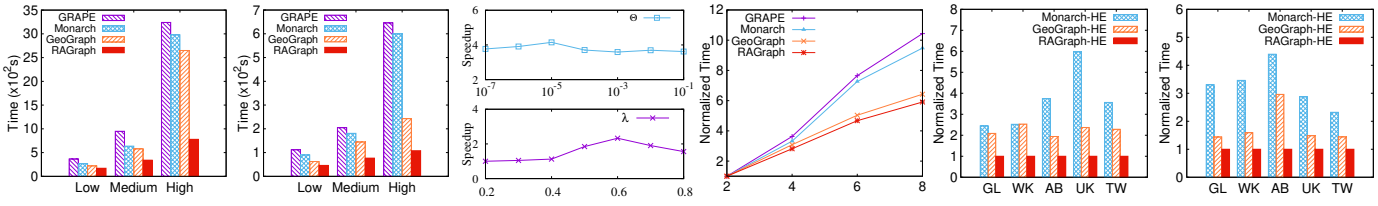


(a) PageRank (b) SSSP (a) Runing time (b) WAN cost (a) Runing time (b) WAN cost

Figure 10: Performance Breakdown.

Figure 11: Performance under skewed chunk.

Figure 12: Performance under Fennel.



(a) PageRank (b) SSSP

Figure 13: Sensitivity to network status.

Figure 14: Sensitivity to Θ and λ.

Figure 15: Scalability.

(a) PageRank (b) SSSP

Figure 16: Performance on HE.

6.5 Effect of Data Partition

We explored the performance impact of different data partition cases in geo-distributed environments. We compare the performance of systems on the TW graph under different data partition strategies to evaluate RAGraph. For the *skewed chunk partitioning strategy*, vertices are partitioned into h parts: the i-th part contains a ratio $i/(\sum_{j=1}^{h} j)$ of vertices. As shown in Figure 11, RAGraph can achieve 2.23× - 5.02× (3.56× on average) speedup and 41% - 74.9% (57.3% on average) WAN cost reduction compared with competitors. In addition, the speedup is more significant under the skewed chunk strategy than uniform-chunk (*i.e.,* the results in Figure 6), with a 9.8% - 24.8% improvement. This validates the efficiency of the RAGraph under a skewed partitioning strategy.

For the *uniform partitioning strategy*, we additionally test the performance under the advanced partitioning strategy Fennel [60]. Figure 12 reports the results of the different systems on the TW graph. RAGraph outperforms others and achieves 2.45× - 4.22× speedup and 42% - 77% WAN cost reduction compared with competitors. Notably, compared with uniform-chunk strategy (*i.e.,* the results in Figure 7), it saves 60% of WAN cost on PageRank and 45% on SSSP. The results on different partitioning strategies indicate that RAGraph is feasible for data with various partitions.

6.6 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 (see Section 6.1). Figure 13 shows the result of PageRank and SSSP in different systems on the WK graph. Compared with the competitors, RAGraph achieves 1.22× - 2.43× speedup on the low-heterogeneity network, 1.7× - 2.79 × speedup on the medium-heterogeneity network, and 2.25× - 5.98 × speedup on the high-heterogeneity network. Besides, RAGraph shows substantial superiority on the high-heterogeneity network, which validates the effectiveness of the Region-Aware framework.

6.7 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 4.1 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 Figure 14, 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 Figure 14, the convergence threshold change has less effect on the effectiveness of RAGraph. Users can choose arbitrary convergence threshold according to their requirements.

6.8 Scalability

We 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 Figure 15 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.

6.9 Performance on Homomorphic Encryption

We finally evaluate the performance of RAGraph and other competitors under RAGraph's homomorphic encryption (HE) module. Since the competitors do not contain encryption functions, we port the HE module to Monarch and GeoGraph such that they can perform geo-distributed computation under HE. The total *computation time* of HE is reported. Figure 16 shows the *normalized time* of HE in different systems, and RAGraph can achieve a 2.32× - 5.97× speedup over Monarch and a 1.43× - 2.96× speedup over GeoGraph. We can see that RAGraph requires a shorter running time on the HE module than Monarch and GeoGraph, and a similar trend can be observed in WAN cost (see Figure 7). This is reasonable as each cross-datacenter data transmission always causes a computation and transmission of encrypted data. Hence, the total computation cost is proportional to the WAN cost of the RAGraph without HE.

7 RELATED WORKS

Distributed Graph Data Processing. A large number of traditional distributed graph processing systems [15, 19, 20, 24, 30, 31, 42, 44, 45, 56, 57, 68, 73–75, 78] have been developed for large-scale graph data analysis. The Bulk Synchronous Parallel model [61] is introduced into graph processing by Pregel [45] and adopted by most distributed graph processing systems [24, 30, 78]. While some systems, such as GraphLab [44], Maiter [74], and GRAPE+ [21], use the asynchronous parallel model (AP) to eliminate synchronization overhead. PowerSwitch [68] uses a hybrid mode for adaptive switching between sync and async during computation. Galois [48] and Priter [73] design priority scheduling from the algorithmic perspective, but may be limited by the impact of network transmission status on data scheduling. LazyGraph [65] involves replicas in local computation and sets a global sync data coherency stage to get a global view of replicas for lazy data consistency. The difference is that RAGraph does not require the proxy to do any global sync interaction and get the eventual consistency. Fan et al. [22, 23] perform a partial evaluation to compute local answers, and the coordinator assembles local answers to get the *complete query result*. Differently, RAGraph focuses on iterative graph algorithms, where the local partial evaluation of vertices spans multiple iteration steps and allows assembling an arbitrary number of partial messages instead of complete messages at different moments. Several other works [29, 34, 42, 50, 58, 60, 69] focus on optimizing graph partition and have been proven to significantly reduce communication cost, which are orthogonal to the optimization of the RAGraph.

RAGraph is inspired by previous distributed systems but proposes a more purposeful design to run over the heterogeneous and fluctuating networks in geo-distributed environments.

Geo-Distributed Data Analysis. Several works focus on designing more efficient big-data analysis frameworks in geo-distributed environments. For example, Medusa [18] allows geo-distributed computation without modifying the Hadoop semantics. GeoDis [17] optimizes data-intensive jobs by considering data localization and migration. Both of them are MapReduce-based. Lube [76], Tetrium [38], *etc.*, 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. Gaia [36] is a geo-distributed ML system that uses an approximate BSP model to eliminate useless communication across data centers. Besides, there are other systems that support database queries [43, 64] and stream data processing [40, 53]. However, the above systems are designed for general data analysis, query, or resource allocation [9], and cannot be adapted to the iterative nature and complex dependencies of the graph algorithms.

Some graph processing systems are designed for geo-distributed environments [39, 71, 77]. PGPregel [77] is a graph processing system focusing on geo-distributed privacy protection by integrating differential privacy. It assumes a certain tolerance for errors in the results. In contrast, RAGraph utilizes HE to obtain accurate results. Monarch [39] reduces the global communication cost by optimizing the local computation under the GAS model. GeoGraph [71] reduces WAN usage through hierarchical clustering. The difference is that RAGraph focuses on the layering and heterogeneity of the network while considering the effects of network fluctuation.

8 CONCLUSION

We design and implement RAGraph, which consists of a Region-Aware framework and two runtime optimizations for geo-distributed graph processing. Firstly, 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. Furthermore, we develop the adaptive hierarchical message interaction, in which two types of message interaction modes are allowed, and RAGraph could adaptively choose the message interaction mode based on network status and message traffics. Lastly, we introduce a discrepancy-aware message filtering strategy to adaptively filter important messages in a discrepancy range of messages.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (U2241212, 62072082, and 62202088), the 111 Project (B16009), the Fundamental Research Funds for the Central Universities (N2216015, N2216012), and a research grant from Alibaba Innovative Research (AIR) Program. Yanfeng Zhang is the corresponding author.

REFERENCES

[1] 2002. web-Google. https://www.cise.ufl.edu/research/sparse/matrices/SNAP/web-Google.html.
[2] 2005. Arabic-2005. https://law.di.unimi.it/webdata/arabic-2005/.
[3] 2005. UK-2005. https://law.di.unimi.it/webdata/uk-2005/.
[4] 2020. libgrape-lite. https://github.com/alibaba/libgrape-lite.
[5] 2021. Facebook Daily Active Users (DAUs). https://investor.fb.com/investor-events/event-details/2021/Facebook-Q2-2021-Earnings/default.aspx.
[6] Mohammad Al Hasan and Vachik S Dave. 2018. Triangle counting in large networks: a review. *WIRES DATA MIN KNOWL* 8, 2 (2018), e1226.
[7] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2016. Efficient state-based crdts by delta-mutation. In *NETYS*. 62–76.
[8] John S Baras and George Theodorakopoulos. 2010. Path problems in networks. *SLCN* 3, 1 (2010), 1–77.
[9] Mohammed Bergui, Said Najah, and Nikola S Nikolov. 2021. A survey on bandwidth-aware geo-distributed frameworks for big-data analytics. *Journal of Big Data* 8, 1 (2021), 40.
[10] Akash Bharadwaj and Graham Cormode. 2022. An Introduction to Federated Computation. In *SIGMOD*.
[11] Kirill Bogdanov, Miguel Peón Quirós, Gerald Q. Maguire Jr., and Dejan Kostic. 2015. Toward Automated Testing of Geo-Distributed Replica Selection Algorithms. In *SIGCOMM*.
[12] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*.
[13] Paolo Boldi and Sebastiano Vigna. 2004. The webgraph framework I: compression techniques. In *WWW*.
[14] Venkatesan T. Chakaravarthy, Fabio Checconi, Prakash Murali, Fabrizio Petrini, and Yogish Sabharwal. 2017. Scalable Single Source Shortest Path Algorithms for Massively Parallel Systems. *TPDS* 28, 7 (2017), 2031–2045.
[15] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. 2015. PowerLyra: differentiated graph computation and partitioning on skewed graphs. In *EuroSys*.
[16] Brian Cho and Marcos K Aguilera. 2012. Surviving Congestion in Geo-Distributed Storage Systems. In *ATC*. 439–451.
[17] Moïse W Convolbo, Jerry Chou, Ching-Hsien Hsu, and Yeh Ching Chung. 2018. GEODIS: towards the optimization of data locality-aware job scheduling in geo-distributed data centers. *Computing* 100 (2018), 21–46.
[18] Pedro ARS Costa, Xiao Bai, Fernando MV Ramos, and Miguel Correia. 2016. Medusa: An efficient cloud fault-tolerant mapreduce. In *CCGrid*.
[19] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Alex Brooks, Nikoli Dryden, Marc Snir, and Keshav Pingali. 2018. Gluon: a communication-optimizing substrate for distributed heterogeneous graph analytics. In *SIGPLAN*.
[20] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of Graph Partitioning Algorithms. *PVLDB* 13, 8 (2020), 1261–1274.
[21] Wenfei Fan, Ping Lu, Wenyuan Yu, Jingbo Xu, Qiang Yin, Xiaojian Luo, Jingren Zhou, and Ruochun Jin. 2020. Adaptive asynchronous parallelization of graph algorithms. *TODS* 45, 2 (2020), 1–45.
[22] Wenfei Fan, Xin Wang, and Yinghui Wu. 2012. Performance Guarantees for Distributed Reachability Queries. *PVLDB* 5, 11 (2012).
[23] Wenfei Fan, Xin Wang, Yinghui Wu, and Dong Deng. 2014. Distributed graph simulation: Impossibility and possibility. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1083–1094.
[24] Wenfei Fan, Jingbo Xu, Yinghui Wu, Wenyuan Yu, Jiaxin Jiang, Zeyu Zheng, Bohan Zhang, Yang Cao, and Chao Tian. 2017. Parallelizing Sequential Graph Computations. In *SIGMOD*.
[25] Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. 2018. Parallelizing Sequential Graph Computations. *ACM Trans. Database Syst.* 43, 4 (2018), 18:1–18:39.
[26] Guanyu Feng, Zixuan Ma, Daixuan Li, Shengqi Chen, Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2021. Risgraph: A real-time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions ops/s. In *SIGMOD*.
[27] Yasuhiro Fujiwara and Go Irie. 2014. Efficient label propagation. In *ICML*. 784–792.
[28] Shufeng Gong, Chao Tian, Qiang Yin, Wenyuan Yu, Yanfeng Zhang, Liang Geng, Song Yu, Ge Yu, and Jingren Zhou. 2021. Automating incremental graph processing with flexible memoization. *PVLDB* 14, 9 (2021), 1613–1625.
[29] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2020. HBP: Hotness Balanced Partition for Prioritized Iterative Graph Computations. In *ICDE*.
[30] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*.
[31] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI*.
[32] Ronald L Graham and Pavol Hell. 1985. On the history of the minimum spanning tree problem. *AHC* 7, 1 (1985), 43–57.
[33] Shai Halevi and Victor Shoup. 2020. Design and implementation of HElib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.* (2020), 1481.
[34] Loc Hoang, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. 2021. CuSP: A Customizable Streaming Edge Partitioner for Distributed Graph Analytics. *SIGOPS* 55, 1 (2021), 47–60.
[35] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *SIGCOMM*.
[36] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *NSDI*.
[37] Tsan-sheng Hsu, Vijaya Ramachandran, and Nathaniel Dean. 1994. Parallel implementation of algorithms for finding connected components in graphs. In *DIMACS*.
[38] Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. 2018. Wide-area analytics with multiple resources. In *EuroSys*.
[39] Anand Padmanabha Iyer, Aurojit Panda, Mosharaf Chowdhury, Aditya Akella, Scott Shenker, and Ion Stoica. 2018. Monarch: Gaining Command on Geo-Distributed Graph Analytics. In *HotCloud*.
[40] Albert Jonathan, Abhishek Chandra, and Jon B. Weissman. 2018. Multi-Query Optimization in Wide-Area Streaming Analytics. In *SoCC*.
[41] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[42] Dongsheng Li, Yiming Zhang, Jinyan Wang, and Kian-Lee Tan. 2019. TopoX: Topology refactorization for efficient graph partitioning and processing. *PVLDB* 12, 8 (2019), 891–905.
[43] Hangyu Li, Hong Xu, and Sarana Nutanong. 2017. Bohr: Similarity Aware Geo-distributed Data Analytics. In *HotCloud*.
[44] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *PVLDB* 5, 8 (2012), 716–727.
[45] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *SIGMOD*.
[46] David W Matula, George Marble, and Joel D Isaacson. 1972. Graph coloring algorithms. In *GTC*. 109–122.
[47] Robert Ryan McCune, Tim Weninger, and Greg Madey. 2015. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *ACM Comput. Surv.* 48, 2 (2015), 25:1–25:39.
[48] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. 2013. A lightweight infrastructure for graph analytics. In *SOSP*.
[49] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. 2018. Multi-level facility location problems. *EJOR* 267, 3 (2018), 791–805.
[50] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. 2015. HDRF: Stream-Based Partitioning for Power-Law Graphs. In *CIKM*.
[51] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *SIGCOMM* 45, 4 (2015), 421–434.
[52] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek Pai, and Michael J Freedman. 2013. Making Every Bit Count in {Wide-Area} Analytics. In *HotOS*.
[53] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. 2014. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *NSDI*.
[54] Shafiur Rahman, Nael B. Abu-Ghazaleh, and Rajiv Gupta. 2020. GraphPulse: An Event-Driven Hardware Accelerator for Asynchronous Graph Processing. In *MICRO*. 908–921.
[55] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *SSS*. 386–400.
[56] Julian Shun and Guy E. Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *PPoPP*.
[57] Shuang Song, Xu Liu, Qinzhe Wu, Andreas Gerstlauer, Tao Li, and Lizy K. John. 2018. Start Late or Finish Early: A Distributed Graph Processing System with Redundancy Reduction. *PVLDB* 12, 2 (2018), 154–168.
[58] Isabelle Stanton and Gabriel Kliot. 2012. Streaming graph partitioning for large distributed graphs. In *SIGKDD*.
[59] Xiaoyi Tao, Kaoru Ota, Mianxiong Dong, Wuyunzhaola Borjigin, Heng Qi, and Keqiu Li. 2020. Congestion-aware traffic allocation for geo-distributed data centers. *TCC* 10, 3 (2020), 1675–1687.
[60] Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. 2014. FENNEL: streaming graph partitioning for massive scale graphs. In *WSDM*.
[61] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111.
[62] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.

[63] Keval Vora, Rajiv Gupta, and Guoqing Xu. 2017. Kickstarter: Fast and accurate computations on streaming graphs via trimmed approximations. In *ASPLOS*. 237–251.
[64] Hao Wang, Di Niu, and Baochun Li. 2020. Turbo: Dynamic and Decentralized Global Analytics via Machine Learning. *TPDS* 31, 6 (2020), 1372–1386.
[65] Lei Wang, Liangji Zhuang, Junhang Chen, Huimin Cui, Fang Lv, Ying Liu, and Xiaobing Feng. 2018. Lazygraph: lazy data coherency for replicas in distributed graph-parallel computation. In *PPoPP*.
[66] Qiange Wang, Yanfeng Zhang, Hao Wang, Liang Geng, Rubao Lee, Xiaodong Zhang, and Ge Yu. 2020. Automating Incremental and Asynchronous Evaluation for Recursive Aggregate Data Processing. In *SIGMOD*.
[67] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2014. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*.
[68] Chenning Xie, Rong Chen, Haibing Guan, Binyu Zang, and Haibo Chen. 2015. SYNC or ASYNC: time to fuse for distributed graph-parallel computation. In *PPoPP*.
[69] Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. 2014. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. In *NeurIPS*.
[70] Xun Yi, Russell Paulet, and Elisa Bertino. 2014. Homomorphic encryption. In *Homomorphic encryption and applications*. 27–46.
[71] Ye Yuan, Delong Ma, Zhenyu Wen, Yuliang Ma, Guoren Wang, and Lei Chen. 2020. Efficient Graph Query Processing over Geo-Distributed Datacenters. In *SIGIR*.
[72] Ye Yuan, Delong Ma, Zhenyu Wen, Zhiwei Zhang, and Guoren Wang. 2021. Subgraph matching over graph federation. *PVLDB* 15, 3 (2021), 437–450.
[73] Yanfeng Zhang, Qinxin Gao, Lixin Gao, and Cuirong Wang. 2011. PrIter: a distributed framework for prioritized iterative computations. In *SOCC*.
[74] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. 2013. Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation. *TPDS* 25, 8 (2013), 2091–2100.
[75] Yu Zhang, Xiaofei Liao, Hai Jin, Lin Gu, Guang Tan, and Bing Bing Zhou. 2017. HotGraph: Efficient Asynchronous Processing for Real-World Graphs. *IEEE Trans. Computers* 66, 5 (2017), 799–809.
[76] Laiping Zhao, Yanan Yang, Ali Munir, Alex X Liu, Yue Li, and Wenyu Qu. 2019. Optimizing geo-distributed data analytics with coordinated task scheduling and routing. *TPDS* 31, 2 (2019), 279–293.
[77] Amelie Chi Zhou, Ruibo Qiu, Thomas Lambert, Tristan Allard, Shadi Ibrahim, and Amr El Abbadi. 2022. PGPregel: an end-to-end system for privacy-preserving graph processing in geo-distributed data centers. In *SOCC*. 386–402.
[78] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. 2016. Gemini: A Computation-Centric Distributed Graph Processing System. In *OSDI*.