Efficient Graph Query Processing over Geo-Distributed Datacenters

Ye Yuan$^\sharp$, Delong Ma†, Zhenyu Wen$^\perp$, Yuliang Ma†, Guoren Wang$^\sharp$, Lei Chen‡
$^\sharp$Beijing Institute of Technology †Northeastern University, China $^\perp$Newcastle University
‡Hong Kong University of Science and Technology

ABSTRACT

Graph queries have emerged as one of the fundamental techniques to support modern search services, such as PageRank web search, social networking search and knowledge graph search. As such graphs are maintained globally and very huge (e.g., billions of nodes), we need to efficiently process graph queries across multiple geographically distributed datacenters, running geo-distributed graph queries. Existing graph computing frameworks may not work well for geographically distributed datacenters, because they implement a Bulk Synchronous Parallel model that requires excessive inter-datacenter transfers, thereby introducing extremely large latency for query processing. In this paper, we propose GeoGraph–a universal framework to support efficient geo-distributed graph query processing based on clustering datacenters and meta-graph, while reducing the inter-datacenter communication. Our new framework can be applied to many types of graph algorithms without any modification. The framework is developed on the top of Apache Giraph. The experiments were conducted by applying four important graph queries, i.e., shortest path, graph keyword search, subgraph isomorphism and PageRank. The evaluation results show that our proposed framework can achieve up to 82% faster convergence, 42% lower WAN bandwidth usage, and 45% less total monetary cost for the four graph queries, with input graphs stored across ten geo-distributed datacenters.

KEYWORDS

Graph search; Geo-distributed; Datacenters

ACM Reference Format:
Ye Yuan$^\sharp$, Delong Ma†, Zhenyu Wen$^\perp$, Yuliang Ma†, Guoren Wang$^\sharp$, Lei Chen‡. 2020. Efficient Graph Query Processing over Geo-Distributed Datacenters. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR' 20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401157

1 INTRODUCTION

Currently, most search engines offer geographically distributed (geo-distributed) search services to users in multiple geographic locations. For example, in support of global product searching, Amazon currently has 15 service regions geo-distributed around the world [1], Bing operates from eight regions [4], Google provides search services from 20 geo-distributed regions [2], and Facebook has built four geo-distributed datacenters to manage social search [5]. The popular multi-site web search engine is composed of a number of search sites geo-distributed around the world [18].

On the other hand, graph queries serve as the foundation of many popular search services, including PageRank web search [9], social networking search [11], knowledge graph search [29], and navigation map search [33]. In these graph queries, the processed graphs are typically as huge as billions of nodes and trillions of edges [10, 37], taking unlimited storage and computational resources. For instance, it is reported that the Google search engine needs to handle a web-scale graph consisting of more than 70 billions of websites and one trillion hyperlinks between them [10]. Such huge graphs with rapid rates of change [45], coupled with high costs of Wide-Area Network (WAN) data transfers [36] and regulatory constraints [46], make it extreme expensive or infeasible to move the entire dataset to a central location.

Therefore, it is critical to design an efficient *geo-distributed graph query processing* mechanism for search services across multiple datacenters. In particular, the fundamental challenge of processing the graph queries in a geo-distributed scenario is the high latency caused by the large number of communications via wide-area networks (WANs). Unfortunately, existing distributed graph computing frameworks are not sufficiently competent to address this challenge. The state-of-the-art frameworks or systems, e.g., Pregel [31], PowerGraph [19] and GraphX [20], are solely designed and optimized for processing graphs *within a single datacenter* with 100 Gbps of bandwidth capacity between worker nodes in the datacenter. Such high network bandwidth is far beyond the reality in inter-datacenter WANs, whose typical capacity is hundreds of Mbps [21].

The inefficiency of geo-distributed graph queries stems from the Bulk Synchronous Parallel (Pregel) model, which is the dominating synchronization model implemented by most of the popular graph computing frameworks [31]. Pregel runs processes in a sequence of "supersteps", which apply updates on nodes and edges iteratively. Each superstep typically allows communication among neighboring nodes only, and it usually takes many supersteps until the algorithm converges [7]. The message exchanges among neighboring nodes may incur excessive inter-datacenter communications if two nodes are allocated in different datacenters. To our best knowledge, only Monarch [23] modifies Pregel to support geo-distributed graph analytics. However, as shown in our experiments, Monarch still surfers large inter-datacenter transfer time and overhead due to that Monarch still inherits the synchronous parallel computation from Pregel. As a result, Monarch is not suitable for online search services that require short response time.

To solve the problems, in this paper, we propose GeoGraph–a universal framework to support efficient geo-distributed graph query processing. The inputs of GeoGraph are a geo-distributed graph G across m datacenters, each of which stores a fragment (F_i) of G, and a graph query Q. The output of GeoGraph is the answer of Q in G, $Q(G)$. GeoGraph works with a coordinator M_c and m datacenters.

GeoGraph consists of offline and online phases. In the offline phase, GeoGraph groups all the datacenters into k ($k < m$) clusters $\{C\}$ based on the WAN latency, and selects a head datacenter DC_h in each cluster. In the online phase, GeoGraph advocates a *evaluation–assembling* framework to process Q within each C. Specifically, GeoGraph first computes local answer $Q(F_i)$ for every F_i in C. $Q(F_i)$ is sent to DC_h. DC_h assembles all $Q(F_i)$ to compute the answer $Q(C)$ of the cluster C. Every $Q(C)$ is then sent to M_c that assembles all $Q(C)$ to compute the final answer $Q(G)$. For assembling at every DC_h, we construct a *meta-graph* $MG(C)$ of cluster C to compute $Q(C)$. For assembling at M_c, we construct a *meta-graph* $MG(G)$ *of* G based on all $MG(C)$ to compute $Q(G)$. The design of GeoGraph is to minimize the inter-datacenter communication to process Q over G.

In the application of GeoGraph, we process four typical graph queries by GeoGraph, i.e., source-destination shortest path (SDSP), graph keyword search (GKS), subgraph isomorphism and PageRank, which are important to search services. Nevertheless, GeoGraph can process all graph queries that can be run on Pregel. We implement GeoGraph on Apache Giraph [32] which is an open-source Pregel-based graph processing system. Take the SDSP query as an example: we query a graph (with 40.3M nodes and 180M edges) geographically distributed across ten datacenters on Amazon Cloud. The experimental results show that, compared to Pregel, GeoGraph reduces the running time by 82%, the WAN bandwidth usage by 42% and the monetary cost by 45%, respectively.

2 PROBLEM STATEMENT

In this section, we first present some backgrounds of geo-distributed datacenters and challenges of running graph queries across geo-distributed datacenters. Then, we formalized our research problem.

2.1 Geo-distributed Datacenters

Many search engine providers and large companies are deploying their services globally to guarantee low latency to users around the world. For example, Microsoft currently deploys eight regions [4] for cloud service, and Google has tens of datacenters distributed in four different regions [2].

Let U_i (resp. D_i) denote as the uplink (downlink) bandwidth from a datacenter DC_i to an endpoint of the WAN. There are three observations for computations over geo-distributed datacenter [16, 34].

Table 1: Uplink/downlink bandwidths (GB/s) of cc2.8xlarge instances from three Amazon EC2 regions to the Internet.

	US East	AP Singapore	Sydney
Uplink Bandwidth	0.52	0.55	0.48
Downlink Bandwidth	2.8	3.5	2.5

(1) *The WAN latency is a significant bottleneck.* The large WAN communication latency is orders of magnitude larger than the local network latency within a datacenter. For example, the local round-trip time (RTT) is 1ms on average, whereas the average WAN RTT is 100ms. (2) *The uplink/downlink bandwidths of a single datacenter can be heterogeneous.* As shown in Table 1, the downlink bandwidths of all the three regions are several times higher than their uplink bandwidths. (3) *The uplink/downlink bandwidths among datacneters are very various.* For example, the uplink and downlink bandwidths of the Singapore region are 17% and 40% higher than those of the Sydney region, respectively.

Computing Model. In this paper, we study graph query processing over geo-distributed datacenters. We assume that there are unlimited computation resources in each single datacenter, and the inter-datacenter data communication is the bottleneck to graph processing in geo-distributed datacenters, since the WAN bandwidth is much more scarce than the computation resources such as CPU and memory [48].

2.2 Problem Definition

Data Graphs. A data graph is a node-labeled, directed graph $G = (V, E, L)$, where (1) V is a finite set of data nodes; (2) $E \subseteq V \times V$, where $(v, w) \in E$ denotes a directed edge from node v to w; and (3) $L()$ is a function such that for each node v in V, $L(v)$ is a label from an alphabet Σ. Intuitively, $L()$ specifies e.g., interests, social roles, ratings [30].

To simplify the discussion, we do not explicitly mention edge labels. Nonetheless, our techniques can be readily adapted for edge labels.

Geo-distributed Graph. Given m geo-distributed datacenters, a strategy P partitions a data graph $G = (V, E, L)$ into disjoint fragments $F = (F_1, ..., F_m)$ such that each $F_i = (V_i, E_i)$ is a subgraph of G, $E = \bigcup_{i\in[1,m]} E_i$, $V = \bigcup_{i\in[1,m]} V_i$, and F_i resides at a datacenter DC_i. Denote by

- $F_i.I$ the set of *in-nodes* $v \in V_i$ such that there is an edge (v', v) incoming from a node v' in F_j $(i \neq j)$;
- $F_i.O$ the set of *out-nodes* $v \in V_i$ such that there is an edge (v, v') with a node v' in F_j $(i \neq j)$; and

- $F_i.O'$ the set of *virtual nodes* v' such that there exists an edge (v, v') in E, $v \in V_i$, and v' is in some $F_j (i \neq j)$.

We refer to the nodes in $F_i.B = F_i.I \bigcup F_i.O$ as the *border nodes* of F_i w.r.t. P. For an edge $e = (v, v')$, $v \in F_i.O$ and $v' \in F_i.O'$, we refer to e as a *cross edge* of F_i, denoted as cE_i. □

Intuitively, each fragment F_i is specified by $(V_i \bigcup F_i.O, E_i \bigcup cE_i)$. Specifically, $V_i \bigcup F_i.O$ of F_i consists of (a) those nodes in V_i and (b) each node in V_i that has an edge to another fragment. The edge set $E_i \bigcup cE_i$ consists of (a) the edges in E_i and (b) cross edges in cE_i, i.e., edges to other fragments. F_i maintains its virtual nodes $F_i.O'$ so that F_i can communicate with other fragments.

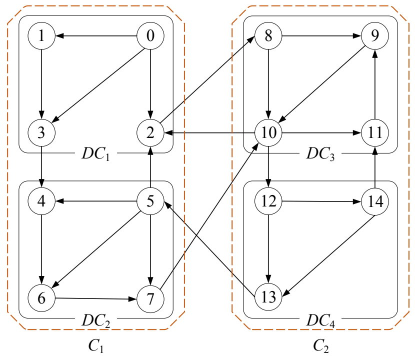


Figure 1: Example of geo-distributed graphs

Figure 1 shows a geo-distributed graph G across four datacenters DC_1, DC_2, DC_3 and DC_4 which maintain fragments of G, F_1, F_2, F_3 and F_4, respectively. For simplicity, we choose F_1 and F_2 to illustrate the above definitions. Node v_2 is an in-node of F_1, i.e., $v_2 \in F_1.I$, since there is an edge incoming from node v_5 in F_2. Node v_3 is an out-node of F_1 i.e., $v_3 \in F_1.O$, since there is an edge from v_3 to v_4 in F_2. Node v_4 is a virtual node of F_1, i.e., $v_4 \in F_1.O'$. Nodes v_3 and v_2 are border nodes of F_1, and edge (v_3, v_4) is a cross edge from F_1 to F_2.

We can advocate state-of-the-art strategies (e.g., [43, 48]) to partition data graphs to different datacenters as a set of geo-distributed graphs. In the following of the paper, we will alternate the two terms *fragment* and *datacenter* in a clear situation.

Based on the geo-distributed graphs, we can define geo-distributed graph processing problems. We refer to a graph computation problem as a class Q of graph queries. Denote Alg_Q as an algorithm to process Q over geo-distributed graphs. We then can formally define our problem.

Definition 2.1 ***(Geo-distributed Graph Processing)****.* Given a query Q and a fragmentation $F = \{F_i\}$ of a geo-distributed graph G, we compute the answer $Q(G)$ of Q in G and *minimize* the total processing time $Time_Q$ of Alg_Q. For a cross edge (v, v') of F_i, let S_v denote as the size of data that needs to be transferred from v to v', and let $Y_{v'}$ denote as the number of times of Alg_Q visiting v' from v. $Time_Q$ is given as the time consumption of Alg_Q transferring data among datacenters:

$$Time_Q = MAX_{i=1}^{k}\{\sum_{v \in F_i.O}\sum_{(v,v') \in v.cE} Y_{v'}X_v(\frac{S_v}{U_i} + \frac{S_v}{D_{v'}})\} \quad (1)$$

Where X_v is a binary, if Alg_Q visits v, $X_v = 1$, otherwise $X_v = 0$. U_i is the uplink bandwidth of F_i; $D_{v'}$ is the downlink bandwidth of the fragment $F_{v'}$ containing v' as an in-node; and $v.cE$ is the set of cross edges incident to v. □

Equation 1 can be explained as follows. The item $Y_v X_v(S_v /U_i + S_v/D_{v'})$ computes the time $T_{(v,v')}$ of Alg_Q transferring data through the cross edge (v, v'). This is because any current data in F_i is transferred through a cross edge of F_i, and because the datacenters are connected with a congestion-free network as stated in the computing model. For the node $v \in F_i.O$, there is a set of cross edges $v.cE$ incident to v. After summarizing $T_{(v,v')}$ over $v.cE$, we get the time T_v of transferring data through node v. After summarizing T_v over all out nodes v in $F_i.O$, we obtain the total time of Alg_Q transferring values from F_i. Alg_Q can be performed across m fragments in a parallel. Therefore, $Time_Q$ is the longest computation time among the m fragments.

In Section 3, we will propose a universal framework to process Q. In Section 4, we will present how to implement the framework by advocating two typical Q, i.e.,*graph traversal* (source-destination shortest path, SDSP) and *graph keyword search.*

3 FRAMEWORK

This section will present the general framework of geo-distributed graph processing, denoted as GeoGraph. We first show the high-level structure of GeoGraph and then its detailed steps.

3.1 Idea of GeoGraph

The inputs of GeoGraph are a geo-distributed graph G and a query Q. The output of GeoGraph is the answer of Q in G, i.e., $Q(G)$. The idea of GeoGraph is to employ the *local evaluation–assembling* framework to compute its local answer $Q(F_i)$ and then assemble all local answers $Q(F_i)$ at a coordinator M_c to compute the final answer $Q(G)$.

When there are too many datacenters, M_c may be bottleneck due to the large number of inter-datacenter communications transferred to M_c. To remedy the shortcoming, GeoGraph groups m datacenters into k ($m \geq k$) clusters, and selects a head datacenter DC_h in each cluster. GeoGraph then applies the local evaluation–assembling framework within each cluster C. Specifically, GeoGraph computes local answer $Q(F_i)$ for every F_i in C. $Q(F_i)$ is sent to DC_h. DC_h assembles all $Q(F_i)$ to compute the local answer $Q(C)$ of the cluster C. Every $Q(C)$ is then sent to M_c that assembles all $Q(C)$ to compute the final answer $Q(G)$. The k head datacentes can share the inter-datacenter communications to M_c.

Based on the above discussion, GeoGraph contains two assembling phases, i.e., assembling at DC_h and M_c. While assembling at DC_h, we need a *meta-graph* $MG(C)$ *of cluster* C to compute $Q(C)$. Similarly, while assembling at M_c, we need a *meta-graph* $MG(G)$ *of* G to compute $Q(G)$

*Definition 3.1 (**Meta-graphs**).* After we obtain k clusters of datacenters, the fragments $F = (F_1, ..., F_m)$ of G are clustered into k disjoint *cluster subgraphs* $CS = (CS_1, ..., CS_k)$. For a cluster subgraph $CS_i \in CS$, denote by $CS_i.I$ the set of *in-nodes* $v \in CS_i$ such that there is an edge (v', v) from a node v' in CS_j $(i \neq j)$, and $CS_i.O$ the set of *out-nodes* $v \in CS_i$ such that there is an edge (v, v') with a node v' in CS_j $(i \neq j)$. Denote by $CS_i.B = CS_i.I \bigcup CS_i.O$ the set of *border nodes* of CS_i. For an edge $e = (v, v')$ with $v \in CS_i.O$ and $v' \in CS_j.I$, we refer to e as a *cross edge* of CS_i, denoted as $CS_i.cE$.

Based on these definitions, the **meta-graph of** G is $MG(G) = (V_{mg}, E_{mg})$, where $V_{mg} = \bigcup_{i=1}^{k} CS_i.B$ and $E_{mg} = \bigcup_{i=1}^{k} CS_i.\ cE$. Assume that cluster C_i consists of c_i fragments F_{ij} $(1 \leq j \leq c_i)$ of F. The **meta-graph of** C is $MG(C) = (V_{mc}, E_{mc})$, where $V_{mc} = \bigcup_{j=1}^{c_i} F_{ij}.B$ and $E_{mc} = \bigcup_{j=1}^{c_i} F_{ij}.cE \setminus CS_i.cE$. □

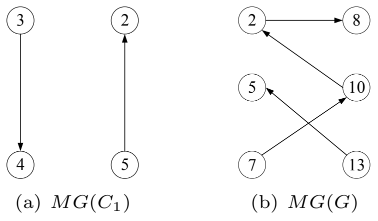


Figure 2: Meta-graphs of C_1 and G in Figure 1

For example, in Figure 1, assume that F_1 and F_2 are in cluster C_1, and F_3 and F_4 are in another cluster C_2. Figure 2(a) shows the meta-graph of C_1, since edges (v_3, v_4) and (v_5, v_2) are cross edges between F_1 and F_2. Figure 2(b) shows the meta-graph of the geo-distributed G, since edges (v_2, v_8), (v_{10}, v_2), (v_{13}, v_5) and (v_7, v_{10}) are cross edges between C_1 and C_2.

3.2 Detailed Steps of GeoGraph

Based on the meta-graphs $MG(G)$ and $MG(C)$, Figure 3 illustrates the detailed steps of GeoGraph. GeoGraph consists of the offline and online phases, introduced as follows.

3.2.1 Offline Phase. The offline phase is to cluster datacenters, and to construct $MG(G)$ and $MG(C)$, which consists of the following three steps.

(1) GeoGraph groups m datacenters into k $(k \leq m)$ clusters based on the metric of the transferring time $t_{i,j}$ between datacenters (line 1). Here we can advocate any clustering algorithm, e.g., k-means [35]. Let $t_{i,j}$ be the transferring time between datacenters DC_i and DC_j. Let $S(F_i.O)$ be the size of $F_i.O$, and $t_{i\to j}$ (resp. $t_{j\to i}$) be the transferring time from DC_i to DC_j. We then have $t_{i,j}$ =min$(t_{i\to j}, t_{j\to i})$, where $t_{i\to j} = S(F_i.O)/U_i + S(F_i.O)/D_j$ and $t_{j\to i} = S(F_j.O)/U_j + S(F_j.O)/D_i$. The meaning of the estimation is as follows. When DC_i communicates with DC_j, Alg_Q usually transfers the local answers associated with nodes in $F_i.O$ from DC_i to DC_j. Thus, we adopt the size of $F_i.O$, $S(F_i.O)$, to estimate $t_{i\to j}$. We then obtain the calculation of $t_{i\to j}$ similar to Equation 1.

```
Procedure GeoGraph_Framework {
  Input: $G$, $Q$
  Output: $Q(G)$
  //The offline phase
  (1) Group all datacenters into $k$ clusters based on the
 inter-datacenter transferring time;
  (2) Within each cluster $C$, select a head datacenter $DC_h$
 to maintain the meta-graph $MG(C)$ of $C$;
  (3) Select a head cluster $C_h$ to maintain the meta-graph $MG$
 $(G)$ of $G$;
  //The online phase
  (4) Post $Q$ to every datacenter $DC_i$;
  (5) Every $DC_i$ computes its local answer $Q(F_i)$ of $F_i$
 that is sent to its $DC_h$;
  (6) $DC_h$ assembles $Q(F_i)$ based on $MG(C)$ to compute
 the local answer $Q(C)$ of $C$. $Q(C)$ is then sent to $C_h$;
  (7) $C_h$ assembles $Q(C)$ based on $MG(G)$ to compute
 the global answer $Q(G)$ of $G$;
      (7.1)$C_h$ sends $Q(G)$ to every $DC_h$ to update $Q(C)$ if
 necessary;
      (7.2)$DC_h$ sends $Q(C)$ to every $DC_i$ to update $Q(F_i)$ if
 necessary;}
```

Figure 3: Framework of Graph Query Processing over Geo-distributed Datacenters.

(2) Within each cluster C, GeoGraph selects a head datacenter DC_h to maintain the meta-graph $MG(C)$ of C (line 2). The principle of selecting DC_h is as follows. Recall that C has c_i fragments. For a datacenter DC_i, we calculate the average transferring time t_i between DC_i and other $c_i - 1$ datacenters as: $t_i = \sum_{j=1}^{c_i-1} t_{i,j}/(c_i - 1)$. DC_h is then the datacenter with the smallest t_i. This principle shows that DC_h can use the least time to receive all $F_i.O$. To construct $MG(C)$, DC_i sends $F_i.B$ and $F_i.cE$ to DC_h.

(3) GeoGraph selects a head cluster C_h to maintain the meta-graph $MG(G)$ of G (line 3). The principle of selecting C_h is similar to that of selecting DC_h. For a cluster C_i, denote $DC_{i,h}$ as its head datacenter. We calculate the average transferring time t_i between $DC_{i,h}$ and other $c_i - 1$ head datacenters as: $t_i = \sum_{j=1}^{c_i-1} t_{i,j}/(c_i - 1)$. Here, $t_{ij} = S(C_j.B)/U_j + S(C_j.B)/D_i$. C_h is then the cluster with the smallest t_i. After receiving all $C_j.B$ and $C_j.cE$, C_h can construct $MG(G)$. Note that DC_h in C_h is responsible for constructing and maintaining $MG(G)$.

3.2.2 Online Phase. The online phase is to compute the answer $Q(G)$ of Q in G based on $MG(G)$ and $MG(C)$, which consists of the following four steps.

(1) Query Q is posted to every datacenter DC_i (line 4).

(2) Every DC_i computes its local answer $Q(F_i)$ of Q in F_i (line 5). Computations on graph data are often iterative. e.g., the algorithm like PageRank and the shortest path that require to update graph data repeatedly until a fixed point. $Q(F_i)$ is the answer of graph algorithms for Q that finishes iterations on F_i. For a query Q, $Q(F_i)$ usually contains the

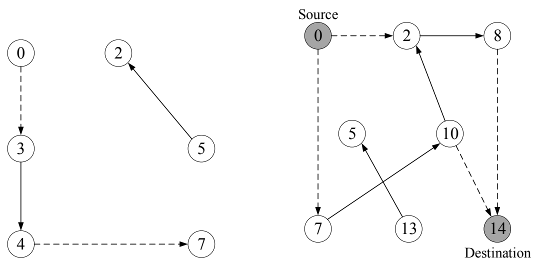
values associated with $F_i.O$. Therefore, only values of nodes in $F_i.O$ are sent to DC_h, which is lightweight. Note that any existing graph computing model (e.g., Pregel) can be advocated to compute $Q(F_i)$. In this step, once $Q(F_i)$ is obtained, it can be sent to DC_h, without waiting for the outcome or messages from any other datacenter.

(3) After DC_h receives all $Q(F_i)$, DC_h assembles these $Q(F_i)$ based on $MG(C)$, to compute the local answer $Q(C)$ of C. DC_h then sends $Q(C)$ to C_h (line 6).

(4) C_h assembles $Q(C)$, based on $MG(G)$, to compute the global answer $Q(G)$ (line 7). This step is similar to the third step. However, for some graph queries Q, (e.g., PageRank), $Q(G)$ includes all nodes of G. For these Q, the current $Q(G)$ only includes partial answers. To obtain a complete $Q(G)$, C_h sends $Q(G)$ to every C to update $Q(C)$. DC_h in C also sends $Q(C)$ to every DC_i to update $Q(F_i)$ if it is further necessary.

Remark. From the global view, GeoGraph employs an asynchronous parallel computation. That is, every datacenter (resp. every cluster) computes its local answers in an asynchronous parallel. The local computation can be efficient due to that nearby datacenters are clustered together. GeoGraph also utilizes meta-graphs to speed up the global computation. Moreover, unlike the existing models, our proposed framework can significantly reduce the inter-datacenter communication, thereby improving the performance of geo-distributed graph processing.

3.3 Running Example



(a) $AMG(C_1)$ (b) $AMG(G)$

Figure 4: Augmented meta-graphs of C_1 and G in Figure 2.

We take the *source-destination shortest path* problem (SDSP) as an example to present how GeoGraph works. The detailed solution to SDSP will be introduced in Section 4.1. Given a graph G and a pair of nodes (s, d) in G, the SDSP problem is to compute the shortest distance $dist(s, d)$ from s to d in G.

The idea to compute $dist(s, d)$ is as follows. Recall that all datacenters are grouped into k clusters C_i $(1 \le i \le k)$ and three cases are considered. Case 1: C_1 contains s. In this case, GeoGraph computes $dist(s, u)$ for each out-node u of C_1. Case 2: C_2 contains d. In this case, GeoGraph computes $dist(v, d)$ for each in-node v of C_2. Case 3: C_i $(3 \le i \le k)$ contains neither s nor d. In this case, GeoGraph computes $dist(u, v)$ for each pair of in-node u and out-node v of C_i. These distances are computed in parallel over clusters. The coordinator M_c assembles these partial distances to construct an augmented meta-graph $AMG(G)$ of G which composes of s, d and the meta-graph $MG(G)$. M_c then computes the $dist(s, d)$ by running the Dijkstra's algorithm over $AMG(G)$. Note that the distance $dist(s, u)$ (resp. $dist(u, v)$ and $dist(v, d)$) can also be calculated in parallel within cluster C_1 (resp. C_2 and C_i $(3 \le i \le k)$). The details are illustrated in Section 4.1.

The following compares our proposed algorithm with Pregel to process the geo-distributed graph G in Figure 1.

To compute $dist(v_0, v_{14})$, Pregel runs the single-source shortest path (SSSP) from v_0 to v_{14} as follows. Within a superstep, each node in G tries to update itself with the smallest distance from v_0 if it received a message from the last superstep. If the node has updated, it will send new messages along its outgoing edges. A node becomes inactive as soon as it cannot get further updates. Pregel needs 11 supersteps until converges, and incurs *21 inter-datacenter messages.*

In terms of GeoGraph, it **first** parallel computes $Q(F_i)$ in four datacenters according to the three cases classified as above. Take the cluster C_1 as an example. Since F_1 contains the source node v_0, $Q(F_1)$ is $dist(v_0, v_3)$. Since F_2 contains neither v_0 nor v_{14}, $Q(F_2)$ are $dist(v_4, v_7)$ and $dist(v_5, v_7)$. Similarly, $Q(F_3)$ are $dist(v_{11}, v_{10})$ and $dist(v_8, v_{10})$, and $Q(F_4)$ is $dist(v_{12}, v_{14})$. **Second**, GeoGraph assembles $Q(F_i)$ to compute $Q(C)$ based on $AMG(C)$. For C_1, GeoGraph obtains $AMG(C_1)$ in Figure 4(a), after it assembles $Q(F_1)$, $Q(F_2)$ and $MG(C_1)$ in Figure 2(a). That is $AMG(C_1) = MG(C_1) \cup (0, 3) \cup (4, 7)$, where edges $(0, 3)$ and $(4, 7)$ are dotted lines in Figure 4(a). GeoGraph then obtains $Q(C_1) = dist(v_0, v_7)$ based on $AMG(C_1)$. Similarly, GeoGraph obtains $Q(C_2) = \{dist(v_8, v_{14}), dist(v_{10}, v_{14})\}$. **Finally**, GeoGraph obtains $AMG(G)$ in Figure 4(b) by assembling $Q(C_1)$, $Q(C_2)$ and $MG(G)$ in Figure 2(b). Based on $AMG(G)$, we can compute the final answer $Q(G) = dist(v_0, v_{14})$ by running the Dijkstra's algorithm over $AMG(G)$ from v_0 to v_{14}.

Based on our calculation, Pregel incurs 21 inter-datacenter communications, but our GeoGraph only needs 4, reduced 5 times unnecessary messages. Also, the benefit will be amplified with the increase number of geo-distributed datacenters (see Section 5).

4 APPLICATION WITH FRAMEWORK

In this section, we feed GeoGraph with specific graph queries Q, i.e., SDSP and GKS. Recall that Alg_Q is an implementation of GeoGraph on a specific graph query. The offline phase of Alg_Q is to cluster datacenters, which is irrespective of Q. Therefore, to process a specific Q, we only need to give the detailed steps of *the online phase* of Alg_Q.

4.1 Graph Traversal

We start with the *source-destination shortest path* problem (SDSP). Consider a directed graph $G = (V, E, L)$ in which for each edge e, $L(e)$ is a positive number. The length of a path $(v_0, ..., v_k)$ in G is the sum of $L(v_{i-1}, v_i)$ for $i \in [1, k]$.

For a pair (s, d) of nodes, denote by $dist(s, d)$ the shortest distance from s to d. SDSP is stated as follows.

- Input: A directed graph G as above, and a pair of two nodes s and d in G.
- Output: Distance $dist(s, d)$ in G.

4.1.1 Steps of Alg_Q. The online phase of Alg_Q is to parallelize computations over clusters C and fragments F_i in every C. Since each F_i follows the same computation logic and is computed in parallel. The following illustrates one instance of applying Alg_Q to F_i

Alg_Q conducts local evaluation to compute $Q(F_i)$ at every F_i, and then assembles $Q(F_i)$ to compute $Q(C)$ at DC_h, which is introduced below.

(1) Local evaluation at F_i. In this step, we consider three cases.

- Case 1: F_i contains s. For each out node u of F_i, Alg_Q computes $dist(s, u)$. Alg_Q then sends the set $\{dist(s, u)\}$ to DC_h.
- Case 2: F_i contains neither s nor d. For each in-node u and out node v of F_i, Alg_Q computes $dist(u, v)$. Alg_Q then sends the set $\{dist(u, v)\}$ to DC_h.
- Case 3: F_i contains d. For each in-node v of F_i, Alg_Q computes $dist(v, d)$. Alg_Q then sends the set $\{dist(v, d)\}$ to DC_h.

(2) Assembling at DC_h. In this step, we need an *augmented meta-graph* of CG, denoted by $AMG(C) = (V_{amc}, E_{amc}, W_{amc})$, where V_{amc} is the node set, E_{amc} is the edge set, and W_{amc} is the set of edge weights.

Assume that C has c fragments $F = \{F_1, F_2, ..., F_c\}$, where F_1 contains s, F_2 contains d, and F_i ($i \in [3, c]$) contains neither s nor d. $AMG(C)$ is constructed from $MG(C) = (V_{mc}, E_{mc})$ as follows.

- $V_{amc} = \{s\} \bigcup \{d\} \bigcup V_{mc}$;
- $E_{amc} = E_{s,u} \bigcup E_{v,d} \bigcup E_{mc}$, where $E_{s,u} = \{(s, u)|\ u \in F_1.O\}$ and $E_{v,d} = \{(v, d)|v \in F_2.I\}$;
- $W_{amc} = \{dist(s, u)\} \bigcup \{dist(u, v)\} \bigcup \{dist(v, d)\}$.

Based on $AMG(C)$, we can obtain local answers $Q(C)$ of C, i.e., $\{dist(s, a)\}$, $\{dist(a, b)\}$ and $\{dist(b, d)\}$, where a and b are border nodes of C. These $Q(C)$ are sent to C_h to construct an *augmented meta-graph* $AMG(G)$ of G similar to $AMG(C)$. Based on $AMG(G)$, we can compute the final answer $Q(G) = dist(s, d)$ by running the Dijkstra's algorithm over $AMG(G)$ from s to d.

For the complexity of inter-datacenter transfers, we have the following conclusion.

- Alg_Q incurs $O(m)$ inter-datacenter data transfers, where m is the number of datacenters.
- Pregel incurs $O(|F.B| \cdot m) \cdot D$ inter-datacenter data transfers, where D is the diameter of G.

4.2 Graph Keyword Search

We consider graph keyword search (GKS) with distinct roots in the same setting of [8, 25]. A keyword query Q is of the form $(k_1, ..., k_w)$, where each k_i is a keyword. Given a directed graph G and a bound b, a match to Q in G at node r is a tree $T(r, l_1, ..., l_w)$ such that (a) T is a subgraph of G, and r is the root of T, (b) for each $i \in [1, w]$, l_i is a node in T containing k_i, i.e., it matches keyword k_i, (c) $dist(r, l_i) \leq b$, and (d) the sum $\sum_{i \in [1,a]} dist(r, l_i)$ is the smallest among all such trees, called *answer trees*. Here $dist(r, l_i)$ denotes the shortest distance from r to l_i, i.e., the length of a shortest path from r to l_i. GKS is as follows.

- Input: A directed graph G, a keyword query $Q = (k_1, ..., k_w)$, and a positive integer b.
- Output: The set $Q(G)$ of answer trees to Q at node r in G within b hops, for r ranging over all nodes in G.

4.2.1 Steps of Alg_Q. Similar to SDSP, the procedure consists of local evaluation at fragments F_i and the assembling at head datacenter DC_h.

(1) Local evaluation at F_i. Alg_Q enumerates the answer trees using a backward search algorithm from the nodes containing keywords, called keyword nodes. Given a set of w keywords, Alg_Q first finds the set of keyword nodes, S_i, for each keyword k_i in F_i. This step can be accomplished efficiently using an inverted list index. Let $S = \bigcup_i S_i$. Accordingly, the backward search algorithm concurrently runs $|S|$ copies of Dijkstra's single source shortest path algorithm, one for each keyword node v in S with node v as the source. The $|S|$ copies of Dijkstra's algorithm run concurrently using iterators. All the Dijkstra's single source shortest path algorithms traverse graph F_i in a reverse direction. When an iterator for keyword node v visits a node u, it finds a shortest path from u to the keyword node v. The concurrent backward search is to discover a common node from which there exists a shortest path to at least one node in each set S_i. Such paths will define a rooted directed tree with the common node as the root and the corresponding keyword nodes as the leaves. The answer trees computed by Alg_Q are approximately sorted in an increasing weight order.

Once Alg_Q finds an answer tree in F_i, Alg_Q returns it to users. When the backward search reaches border nodes of F_i, Alg_Q finds a set of paths (from keyword nodes) that are potential parts of answer trees, called *partial trees*. The partial trees are also sent to DC_h to obtain local answer $Q(C)$.

Also, Alg_Q specifies the b-neighbor $N_b(v)$ of each node $v \in F_i.B$, where $N_b(v)$ is the subgraph of F_i induced by the nodes within b hops of v. Note that $N_b(v)$ consists of inner nodes of F_i but not nodes in another F_j ($i \neq j$). Let $G(N_i.B) = (V(N_i.B), E(N_i.B))$ denote as the graph consisted of all $N_b(v)$, i.e., $G(N_i.B) = \bigcup_{b \in F_i.B} N_b(v)$. In addition to the partial trees, $G(N_i.B)$ is sent to DC_h.

(2) Assembling at DC_h. In this step, we need an *augmented meta-graph* of C, denoted by $AMG(C) = (V_{amc}, E_{amc})$, where V_{amc} is the node set and E_{amc} is the edge set. $|C|$ is the number of datacenters in cluster C. $AMG(C)$ is constructed from $MG(C) = (V_{mc}, E_{mc})$ as follows.

- $V_{amc} = \bigcup_{i=1}^{|C|} V(N_i.B) \bigcup V_{mc}$;
- $E_{amc} = \bigcup_{i=1}^{|C|} E(N_i.B) \bigcup E_{mc}$.

Alg_Q then continues the backward search from the partial trees over $AMG(C)$ to find $Q(C)$. All $Q(C)$ from clusters are sent to C_h to construct an *augmented meta-graph* $AMG(G)$ of G similar to $AMG(C)$. Over $AMG(G)$, Alg_Q computes the set $Q(G)$ of all answer trees.

For the complexity of inter-datacenter transfers, we have the following conclusion.

- Alg_Q incurs $O(m)$ inter-datacenter data transfers.
- Pregel incurs $O(m \cdot b \cdot w)$ inter-datacenter data transfers.

5 PERFORMANCE EVALUATION

We evaluate the effectiveness and efficiency of GeoGraph by comparing with other two state-of-the-art algorithms. All experiments are conducted by using real-world graph datasets and running over Amazon EC2.

5.1 Experimental Settings

Table 2: Dataset

Graph Dataset	Node Size	Edge Size
GoogleWeb(GW)	0.8 M	4.6 M
IMDB(ID)	1.3 M	6.2 M
liveJournal(LJ)	3.5 M	46.1 M
Dbpedia(DP)	28 M	33.4 M
Freebase(FB)	40.3 M	180 M
Synthesized Graph (SG)	620 M	24 B

Datasets. Five real-world graphs and one synthesized graph are advocated in our experiments to represent social networks, web graphs and knowledge graphs. Table 2 shows the number of nodes and edges for these graphs [3].

Graph queries. We test four graph queries which are widely used in search services. (1) *source-destination shortest path* (SDSP) is introduced in Section 4.1 and can find the relationships between nodes in a web search graph. (2) *graph keyword search* (GKS) is introduced in Section 4.2 and has been widely used in search engines [8]. (3) *PageRank* (PR) [9] is a graph algorithm widely used in search engines to measure the importance of webpages. The web is modeled as a graph, where each webpage is a node and the links between webpages are edges of the graph. Each node has a rank value, which indicates an importance of a particular page. The value of a node is defined recursively and depends on the number of all nodes that link to it. (4) *Subgraph Isomorphism* (SI) [14] is a type of graph pattern matching, which finds all matched subgraphs of the query pattern in a large graph. It has many applications in social networks, multimedia analysis and knowledge base which can be modeled by graph-structured data.

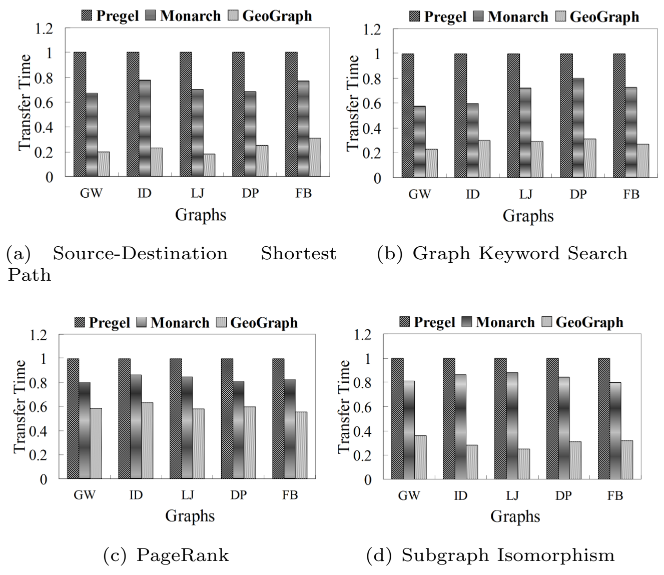
Compared approaches. We compare GeoGraph with Pregel, and also with Monarch [23] that modifies Pregel to support geo-distributed graph processing.

We have implemented GeoGraph, Pregel and Monarch on Apache Giraph 1.2.0 [32] which is an open-source graph computing platform. Also, the default graph partitioning strategy of Giraph is used to partition data graphs to obtain geo-distributed graphs.

Configuration details. We select ten regions of Amazon EC2 as the geo-distributed datacenters, namely *US East* (USE), *US West Oregon* (USW-O), *US West North California* (USW-NC), *EU Ireland* (EU-I), *EU France* (EU-F), *EU Germany* (EU-G), *Asia Pacific Singapore* (SIN), *Asia Pacific Tokyo* (TKY), *Asia Pacific Sydney* (SYD) and *South America* (SA). In each region, we construct a cluster of eight cc2.8xlarge instances [48]. The number of clusters is advocated as 2–10, and the default value is 6.

In the experiments, we compare the transfer time and WAN usage of graph queries supported by GeoGraph and the two compared algorithms. We also evaluate the monetary cost of inter-datacenter communications, using the real network prices charged by Amazon EC2. This is because WAN bandwidths are charged differently in different geographic locations.

5.2 Experimental Results



(a) Source-Destination Shortest Path (b) Graph Keyword Search

(c) PageRank (d) Subgraph Isomorphism

Figure 5: Normalized results of the inter-datacenter transfer time supported by the compared methods for the four graph queries.

Exp-1: transfer time. Figure 5 shows the normalized inter-datacenter transfer time supported by the compared methods for SDSP, GKS, PR and SI algorithms on the five real-world graphs. *All results are normalized to the result of Pregel.* From the results we observe that GeoGraph is able to obtain the lowest inter-datacenter transfer times under all settings. Compared to Pregel and Monarch, GeoGraph

Table 3: Actual WAN usages (GB) supported by the compared methods for the four graph queries.

(a) Shortest Path

Graph	Pregel (GB)	Monarch (GB)	**GeoGraph** (GB)
GW	0.67	0.53	**0.32**
ID	0.83	0.69	**0.46**
LJ	1.26	0.93	**0.69**
DP	1.73	1.35	**1.12**
FB	2.25	2.02	**1.51**

(b) Graph Keyword Search

Graph	Pregel (GB)	Monarch (GB)	**GeoGraph** (GB)
GW	2.06	1.36	**1.5**
ID	2.92	2.33	**2.18**
LJ	4.19	3.73	**3.51**
DP	5.81	5.52	**4.76**
FB	9.74	9.06	**7.87**

(c) PageRank

Graph	Pregel (GB)	Monarch (GB)	**GeoGraph** (GB)
GW	14.68	11.75	**8.22**
ID	20.38	17.52	**12.22**
LJ	23.53	19.77	**13.65**
DP	38.72	30.97	**23.24**
FB	56.28	46.73	**29.27**

(d) Subgraph Isomorphism

Graph	Pregel (GB)	Monarch (GB)	**GeoGraph** (GB)
GW	0.98	0.85	**0.86**
ID	1.57	1.43	**1.37**
LJ	3.39	3.22	**2.71**
DP	5.12	4.61	**4.52**
FB	8.32	6.91	**7.32**

Table 4: Actual monetary costs (US $) supported by the compared methods for the four graph queries.

(a) Shortest Path

Graph	Pregel $	Monarch $	**GeoGraph** $
GW	0.05	0.04	**0.02**
ID	0.05	0.05	**0.03**
LJ	0.09	0.07	**0.05**
DP	0.12	0.09	**0.08**
FB	0.16	0.14	**0.11**

(b) Graph Keyword Search

Graph	Pregel $	Monarch $	**GeoGraph** $
GW	0.14	0.1	**0.11**
ID	0.2	0.16	**0.15**
LJ	0.29	0.26	**0.25**
DP	0.41	0.39	**0.33**
FB	0.68	0.63	**0.55**

(c) PageRank

Graph	Pregel $	Monarch $	**GeoGraph** $
GW	1.03	0.82	**0.58**
ID	1.43	1.23	**0.86**
LJ	1.65	1.38	**0.96**
DP	2.71	2.17	**1.63**
FB	3.94	3.27	**2.05**

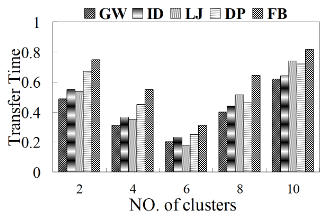
(d) Subgraph Isomorphism

Graph	Pregel $	Monarch $	**GeoGraph** $
GW	0.07	0.06	**0.06**
ID	0.11	0.1	**0.1**
LJ	0.24	0.23	**0.19**
DP	0.36	0.32	**0.32**
FB	0.58	0.48	**0.51**

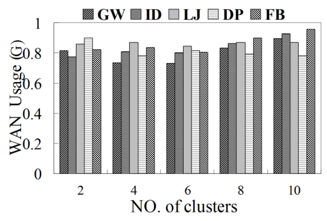
reduces the transfer times of the SDSP, GKS, PR and SI algorithms by 77% and 42%, 72% and 38%, 82% and 26%, and 80% and 42%, respectively. This is because GeoGraph advocates the meta-graphs which are very effective to accelerate graph queries. Pregel does not take into the account of the WAN impact. Monarch still needs many inter-datacenter rounds, though it transfers inter-datacenter data in a lightweight manner. This is because Monarch inherits the synchronous parallel computation from Pregel.

Exp-2: WAN usage. Table 3 shows the WAN usages obtained by the compared methods for the SDSP, SI, GKS and PR queries on the five real-world graphs. All results are the actual WAN usages measured by the transfer data sizes. As shown in the table, GeoGraph consumes the lowest WAN usages under all settings, and needs average 0.82G, 3.36G, 3.96G and 17.32G for the SDSP, SI, GKS and PR algorithms, respectively. PR costs much more WAN usages than other three graph queries, due to that PR visits every datacenters more times. Compared to Pregel and Monarch, GeoGraph reduces the WAN usages of the SDSP, GKS, PR and SI algorithms by 42% and 26%, 21% and 12%, 40% and 25%, and 14% and 6%, respectively. The reductions of the WAN usages are obvious similar as the transfer times. This shows that GeoGraph is both effective at the low inter-datacenter transfer time and traffics.

Exp-3: monetary cost. Table 4 shows the actual monetary cost results obtained by the compared methods for inter-datacenter data transfer. The calculation follows the Amazon Cloud pricing model. As reported in the table, GeoGraph obtains lower monetary cost than Pregel in all settings. Comparing with Monarch, GeoGraph is able to obtain lower monetary cost in most cases due to its low WAN usages. In some cases, such as for GW graphs running GKS and SI queries, the monetary cost of Monarch is lower than GeoGraph. The reason is that the WAN usages of the two algorithms in these cases are very close and Monarch can distribute most communications to less expensive datacenters. The above observations show that GeoGraph is able to achieve good monetary costs, although it is not specifically designed for monetary cost optimizations.

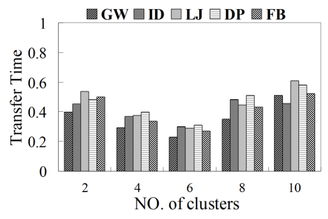


(a) Normalized transfer time

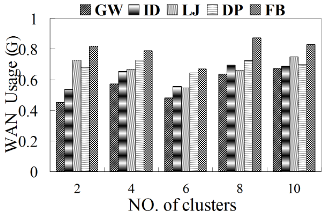


(b) Actual WAN Usage

Figure 6: Impact of different numbers of clusters on GeoGraph for source-destination shortest path.



(a) Normalized transfer time



(b) Actual WAN Usage

Figure 7: Impact of different numbers of clusters on GeoGraph for graph keyword search.

Exp-4: impact of the cluster number. Finally, we examine the performances of GeoGraph with respect to the cluster number, i.e., $k = 2, 4, 6, 8, 10$. Figure 6 shows the transfer times and WAN usages of the SDSP query on the five real-world graphs. Figure 7 shows the results of the GKS query. As reported in the figures, when the cluster number is 6, GeoGraph consumes both the smallest transfer times and WAN usages under all settings. This is because the 6 clusters correspond to 6 regions (i.e., US, EU, SIN, TK, SYD and SA) which are best clusters according to the distance measure. We also observe that, the transfer times and WAN usages decrease from $k = 2$ to $k = 6$ and increase from $k = 6$ to $k = 10$. The reason is that, for 10 datacenters, GeoGraph has the same procedures for both 10 clusters and 1 cluster.

6 RELATED WORK

We categorize the related work into two parts reviewed as follows.

Graph Processing System. A large number of graph processing systems have been proposed [26][31][19][41][40]. Most of them allow users to provide a node program which is run repeated on node (in parallel) via the system. These systems are based on different frameworks or computation models, such as MapReduce, BSP and GAS.

(1) The typical MapReduce-based systems include PEGASUS [26], HaLoop [13] and Twister [17]. PEGASUS [26] is a large-scale graph mining algorithm library proposed to solve the scalability problem of large-scale graph data mining. Through adding caching mechanism and modifying MapReduce's task scheduling framework, HaLoop [13] optimizes the iterative processing for graphs. Different from HaLoop, Twister [17] assumes that all data resides in memory and the task process is saved through the buffer pool for graph processing. (2) Most popular systems are based on the Bulk Synchronous Parallel (BSP) model [31][12][49]. Pregel [31] is the most typical BSP-based graph computation system. A typical Pregel computation consists of input, when the graph is initialized, followed by a sequence of supersteps separated by global synchronization points until the algorithm terminates, and finishing with output. GPS [7] is one of the most popular BSP-based systems that is a completely open-source project and designed for large-scale and fault-tolerant graph processing. (3) The works [40] [50] [19] are based the GAS model which iteratively executes user-defined node computations until convergence. The GAS model, which proposed in PowerGraph [19], divides the node computation into three stages, namely Gather, Apply and Scatter.

However, due to no consideration of the high latency and transmission cost under the geo-distributed background, none of them support geo-distributed graph processing. As shown in the experiments, Pregel incurs 82% and 45% higher inter-datacenter time and traffics than our proposed system GeoGraph.

Geo-Distributed Analytics. A number of recent works have made the case for geo-distributed analytics [47][46][36][6] [45] which focus on data processing, SQL-support, streaming processing, scalability and so on. For traditional distributed data analytics, data is assumed to be processed in a single, centralized datacenter. However, in a geo-distributed environment, data is generated and stored at geographically distributed datacenters. Hence, the traditional strategies are not suitable for the new cases.

(1) For data processing and analytics, like G-Hadoop [47], G-MR [24], Nebula [39] and Medusa [15], all these systems are map-reduced based framework, require to distribute a job over multiple clouds and then aggregation of outputs, but they need many MapReduce rounds to obtain query answers and in each round it produces a large number of intermediate results that are iterated into next round. (2) Geode [46], Google's F1 [42] and Spanner [6] are the SQL-style processing frameworks. Unfortunately, these works only focus on simple queries and largely ignored iterative workloads like distributed graph processing. (3) Iridium [36], Pixida [28] and JetStream [38] consider the bandwidth influence during the geo-distributed stream processing. They either move data out of the site with low uplink bandwidth before query arrives or place many of query's reduce tasks in it. However, simple task placement techniques do not work well as there is a need to deal with affinity and these approaches may soon be rendered impossible by sovereignty regulations governing data movement. (4) Reseal [27], Flutter [22] and Tudoran [44] aim at how to assign tasks to each datacenter reasonably, so as to obtain faster job completion time, but the challenge comes in finding a straggler process. WANalytics [45] and Geode [46] studied data replication and result caching, but it has additional costs to maintain the caching of results. (5) G-Cut [48] is proposed to partition data graphs across geo-distributed datacentres. G-Cut is orthogonal to our work, i.e., GeoGraph can process geo-distributed graphs partitioned by G-Cut. Particularly, G-Cut aims at minimizing the inter-datacenter data transfer time of the graph partitioning in geo-distributed datacenters while satisfying the WAN usage budget, but it still needs edge migrations to diminish the data traffic in bottleneck datacenters to further reduce the inter-datacenter data transfer time. Monarch [23] modifies Pregel to support geo-distributed graph analytics. However, as shown in our experiments, Monarch still surfers large inter-datacenter transfer time and overhead due to that Monarch still inherits the synchronous parallel computation from Pregel.

7 CONCLUSION

We introduce GeoGraph, a universal framework that is designed to process graph queries over geographically distributed datacenters efficiently. Based on meta-graph and clustered datacenters, GeoGraph behaves synchronous computations within a datacenter and asynchronous computations across datacenters. Such mechanism of GeoGraph guarantees fast convergence and correctness of existing graph queries that serve as the foundation of search services. Our prototype implementation and evaluation show that GeoGraph can reduce the running time and WAN bandwidth usage by up to 82% and 45%, respectively, while significantly reducing the monetary cost for running graph queries on multiple clouds. We conclude that GeoGraph is a general, efficient, and readily implementable framework that can benefit geo-distributed graph query processing. In the future, we will adapt GeoGraph to processing more complex graph analytics, e.g., graph convolutional network and graph embedding.

8 ACKNOWLEDGMENT

Ye Yuan is supported by the NSFC (Grant No. 61932004) and the Fundamental Research Funds for the Central Universities (Grant No. N181605012 and N2016009), and the Liao Ning Revitalization Talent Program (Grant No. XLY-C1807216). Guoren Wang is supported by the NSFC (Grant

No. 61732003) and the National Key Research and Development Program of China (Grant No. 2018YFB1004402). Lei Chen is partially supported by the China NSFC No. 61729201, the Hong Kong RGC GRF Project 16207617, CRF Project C6030-18GF and AOE Project AoE/E-603/18.

REFERENCES

[1] Aws global infrastructure, https://aws.amazon.com/about-aws/global-infrastructure/.

[2] Google datacenter locations, https://www.google.com/about/datacenters/ inside/ locations/ index.html.

[3] Stanford large network dataset collection, https://snap.stanford.edu/ data/.

[4] Windows azure regions, https://azure.microsoft.com/ en-us/regions/.

[5] O. Alonso, V. Kandylas, and S.-E. Tremblay. Social knowledge graph explorer. In *Proceedings of SIGIR*, pages 1317–1320, 2019.

[6] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, et al. Spanner: Becoming a sql system. In *SIGMOD*, pages 331–343, 2017.

[7] N. T. Bao and T. Suzumura. Towards highly scalable pregel-based graph processing platform with x10. In *Proceedings of WWW*, pages 501–508, 2013.

[8] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE*, pages 431–440, 2002.

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[10] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1-6):309–320, 2000.

[11] N. Bronson, Z. Amsden, G. Cabrera, Chakka, et al. Tao: Facebook's distributed data store for the social graph. In *Usenix Conference on Technical Conference*, pages 49–60, 2013.

[12] Y. Bu, V. Borkar, J. Jia, M. J. Carey, and T. Condie. Pregelix: Big (ger) graph analytics on a dataflow engine. *Proceedings of the VLDB Endowment*, 8(2):161–172, 2014.

[13] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. Haloop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1-2):285–296, 2010.

[14] L. P. Cordellaand, P. Foggia, and C. Sansone. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):1367–1372, 2004.

[15] P. A. Costa, X. Bai, F. M. Ramos, and M. Correia. Medusa: An efficient cloud fault-tolerant mapreduce. In *CCGrid*, pages 443–452. IEEE, 2016.

[16] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer. A survey on geographically distributed big-data processing using mapreduce. *IEEE Transactions on Big Data*, 5(1):60–80, 2017.

[17] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *HPDC*, pages 810–818. ACM, 2010.

[18] G. Francès, X. Bai, B. B. Cambazoglu, and R. Baeza-Yates. Improving the efficiency of multi-site web search engines. In *WSDM*, pages 3–12, 2014.

[19] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

[20] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: graph processing in a distributed dataflow framework. In *OSDI*, 2014.

[21] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, Ganger, et al. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI 17*, pages 629–647, 2017.

[22] Z. Hu, B. Li, and J. Luo. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. In *INFOCOM*, pages 1–9. IEEE, 2016.

[23] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica. Monarch: gaining command on geo-distributed graph analytics. In *HotCloud*, 2018.

[24] C. Jayalath, J. Stephen, and P. Eugster. From the cloud to the atmosphere: Running mapreduce across data centers. *IEEE transactions on computers*, 63(1):74–87, 2013.

[25] V. Kacholia, S. Pandit, S. Chakrabarti, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB*, pages 505–516, 2005.

[26] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *ICDM*, pages 229–238. Washington, DC, USA, 2009.

[27] R. Kettimuthu, G. Agrawal, P. Sadayappan, and I. Foster. Differentiated scheduling of response-critical and best-effort wide-area data transfers. In *IPDPS*, pages 1113–1122. IEEE, 2016.

[28] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues. Pixida: optimizing data parallel jobs in wide-area data analytics. *Proceedings of the VLDB Endowment*, 9(2):72–83, 2015.

[29] X. Lu, S. Pramanik, R. Saha Roy, A. Abujabal, et al. Answering complex questions by joining multi-document evidence with quasi knowledge graphs. In *SIGIR*, pages 105–114, 2019.

[30] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. In *VLDB*, pages 310–321, 2012.

[31] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[32] C. Martella, R. Shaposhnik, D. Logothetis, C. Martella, R. Shaposhnik, and D. Logothetis. Practical graph analytics with apache giraph. 2015.

[33] K. Miyaki. Route search method in navigation system, Apr. 1 2003. US Patent 6,542,817.

[34] Z. Niu, B. He, C. Zhou, and C. T. Lau. Multi-objective optimizations in geo-distributed data analytics systems. In *ICPADS*, pages 519–528, 2017.

[35] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *SIGKDD*, pages 277–281, 1999.

[36] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, et al. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review*, 45(4):421–434, 2015.

[37] L. Qin, J. X. Yu, L. Chang, H. Cheng, C. Zhang, and X. Lin. Scalable big graph processing in mapreduce. In *Proc. of SIGMOD*, pages 827–838, 2014.

[38] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *NSDI*, pages 275–288, 2014.

[39] M. Ryden, K. Oh, A. Chandra, and J. Weissman. Nebula: Distributed edge cloud for data intensive computing. In *IC2E*, pages 57–66. IEEE, 2014.

[40] D. Sengupta, N. Sundaram, X. Zhu, T. L. Willke, et al. Graphin: An online high performance incremental graph processing framework. In *Euro-Par*, pages 319–333. Springer, 2016.

[41] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, pages 505–516. ACM, 2013.

[42] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, et al. F1: A distributed sql database that scales. *Proceedings of the VLDB Endowment*, 6(11):1068–1079, 2013.

[43] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs. In *KDD*, pages 1222–1230. ACM, 2012.

[44] R. Tudoran, A. Costan, R. Wang, L. Bougé, et al. Bridging data in the clouds: An environment-aware system for geographically distributed data transfers. In *CCGrid*, pages 92–101. IEEE, 2014.

[45] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. Wanalytics: Analytics for a geo-distributed data-intensive world. In *CIDR*, 2015.

[46] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *NSDI*, pages 323–336, 2015.

[47] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, 29(3):739–750, 2013.

[48] A. C. Zhou, S. Ibrahim, and B. He. On achieving efficient data transfer for graph processing in geo-distributed datacenters. In *ICDCS*, pages 1397–1407. IEEE, 2017.

[49] C. Zhou, J. Gao, B. Sun, and J. X. Yu. Mocgraph: Scalable distributed graph processing using message online computing. *Proceedings of the VLDB Endowment*, 8(4):377–388, 2014.

[50] S. Zhou, R. Kannan, H. Zeng, and V. K. Prasanna. An fpga framework for edge-centric graph processing. In *Proceedings of CF 2018*, pages 69–77. ACM, 2018.