



**HAL**  
open science

## Cost-Aware Partitioning for Efficient Large Graph Processing in Geo-Distributed Datacenters

Amelie Chi Zhou, Bingkun Shen, Yao Xiao, Shadi Ibrahim, Bingsheng He

► **To cite this version:**

Amelie Chi Zhou, Bingkun Shen, Yao Xiao, Shadi Ibrahim, Bingsheng He. Cost-Aware Partitioning for Efficient Large Graph Processing in Geo-Distributed Datacenters. *IEEE Transactions on Parallel and Distributed Systems*, 2019, pp.1707-1723. 10.1109/TPDS.2019.2955494 . hal-02389120

**HAL Id: hal-02389120**

<https://inria.hal.science/hal-02389120v1>

Submitted on 3 Dec 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Cost-Aware Partitioning for Efficient Large Graph Processing in Geo-Distributed Datacenters

Amelie Chi Zhou\* Bingkun Shen\* Yao Xiao\* Shadi Ibrahim† Bingsheng He‡

\*National Engineering Lab for Big Data System Computing Technology, Shenzhen University

†Inria, IMT Atlantique, LS2N ‡National University of Singapore

**Abstract**—Graph processing is an emerging computation model for a wide range of applications and graph partitioning is important for optimizing the cost and performance of graph processing jobs. Recently, many graph applications store their data on geo-distributed datacenters (DCs) to provide services worldwide with low latency. This raises new challenges to existing graph partitioning methods, due to the multi-level heterogeneities in network bandwidth and communication prices in geo-distributed DCs. In this paper, we propose an efficient graph partitioning method named *Geo-Cut*, which takes both the cost and performance objectives into consideration for large graph processing in geo-distributed DCs. *Geo-Cut* adopts two optimization stages. First, we propose a cost-aware streaming heuristic and utilize the one-pass streaming graph partitioning method to quickly assign edges to different DCs while minimizing inter-DC data communication cost. Second, we propose two partition refinement heuristics which identify the performance bottlenecks of geo-distributed graph processing and refine the partitioning result obtained in the first stage to reduce the inter-DC data transfer time while satisfying the budget constraint. *Geo-Cut* can be also applied to partition dynamic graphs thanks to its lightweight runtime overhead. We evaluate the effectiveness and efficiency of *Geo-Cut* using real-world graphs with both real geo-distributed DCs and simulations. Evaluation results show that *Geo-Cut* can reduce the inter-DC data transfer time by up to 79% (42% as the median) and reduce the monetary cost by up to 75% (26% as the median) compared to state-of-the-art graph partitioning methods with a low overhead.

**Keywords**—Graph Processing, Wide Area Network, Geo-distributed Datacenters.

## 1 INTRODUCTION

Graph processing is an emerging computation model for a wide range of applications, such as social network analysis [1], [2], natural language processing [3] and web information retrieval [4]. Graph processing systems such as Pregel [5], GraphLab [6] and PowerGraph [7] follow the “think-as-a-vertex” philosophy and encode graph computation as vertex programs which run in parallel and communicate through edges of the graphs. Vertices iteratively update their states according to the messages received from neighboring vertices. Thus, efficient communication between vertices and their neighbors is important to the cost and performance of graph processing algorithms. Graph partitioning, which distributes graph processing workload to multiple nodes for better parallelism, plays a vital role in reducing cross-partition data communication and ensuring load balancing.

Many graph applications, such as social networks, involve large sets of data spread in multiple geographically distributed (geo-distributed) datacenters (DCs). For example, Facebook receives terabytes of text, image and video data everyday from users around the world [8]. In order to provide reliable and low-latency services to the users, Facebook has built four geo-distributed DCs to maintain and manage those data. Sometimes, it is impossible to move data out of their DCs due to privacy and government regulation reasons [9]. Therefore, it is inevitable to process those data in a geo-distributed way. We identify a number of technical challenges for partitioning and processing graph data across geo-distributed DCs.

First, with different graph processing algorithms, the traffic pattern during graph processing can be highly heterogeneous [10]. On the one hand, the data sizes transferred by different vertices can be highly variant, mainly due to the different degrees of vertices. In natural graphs, the degrees of vertices usually follow power-law distribution [7], which cause a small portion of vertices to consume most of the traffic during graph processing [10]. On the other hand, the sent and received data sizes of a single vertex can also be heterogeneous according to the different features of graph processing algorithms. For example, in PageRank, the data sizes sent from and received by a vertex are proportional to the number of out and in neighbors of the vertex, respectively. Constructing a synthetic power-law graph [7] with  $\lambda = 2.1$ , the difference between the numbers of in and out neighbors of a vertex can be 500x. Thus, it is non-trivial to obtain a good graph partitioning which minimizes cross-partition data communication during graph processing.

Second, the geo-distributed DCs have highly heterogeneous network bandwidths [11], [12]. On the one hand, the uplink and downlink bandwidths of a DC can be highly heterogeneous due to the different link capacities and resource sharing among multiple applications [13]. We have found more than four times difference between the uplink and downlink bandwidths of the cc2.8xlarge Amazon EC2 instances. On the other hand, the network bandwidths of the same type of link in different DCs can also be heterogeneous due to different hardwares and workload patterns in the DCs. For example, it has been observed that the network bandwidth of the EU region

is both faster and more stable than the network bandwidth of the US region in Amazon EC2 [14]. Due to the heterogeneous traffic pattern during graph processing, even if the amount of data transferred across DCs is minimized, it can still result in long inter-DC data transfer time if the network bandwidth heterogeneity is not considered.

Third, the network prices in geo-distributed DCs are also highly heterogeneous. On the one hand, the data communication between graph partitions in geo-distributed DCs goes through the Wide Area Network (WAN), which is usually much more expensive than intra-DC data communication [13], [15]. Traditional graph partitioning methods which try to balance the workload among different partitions while reducing the vertex replication rate [7] can end up with large inter-DC data transfer size and hence high monetary cost on WAN usage. On the other hand, the inter-DC network prices differ from DC to DC. For example, on Windows Azure, the network prices of moving data out of the US East and the Japan East regions to WAN differ by 37%. Considering the heterogeneous traffic pattern between graph partitions, it requires careful design to obtain low inter-DC data communication cost.

Due to the above challenges, it is complicated to optimize the performance and cost objectives for geo-distributed graph processing at the same time. As will be shown in the evaluation (e.g., Figure 12), the solution space to this bi-objective optimization problem is quite large and we need an efficient and effective method to solve the problem. In this paper, we propose a geo-aware graph partitioning method named *Geo-Cut*. *Geo-Cut* considers the special features of graph traffic and geo-distributed DCs, and aims to optimize the performance of graph processing by minimizing the inter-DC data transfer time while satisfying the budget constraint on inter-DC data communication cost. To improve the optimization efficiency, *Geo-Cut* adopts a two-stage optimization approach. In the first stage, we propose a streaming heuristic which aims at minimizing the inter-DC data transfer cost, and utilize the one-pass streaming partitioning method to quickly assign edges onto different DCs. In the second stage, we propose two partition refinement heuristics which identify the performance bottlenecks of graph processing and refine the partitioning result obtained in the first stage to reduce the inter-DC data transfer time while guaranteeing the budget constraint. Thanks to the two-stage optimization, *Geo-Cut* exhibits a lightweight runtime overhead and can be easily and efficiently extended to partition dynamic graphs. We evaluate the effectiveness and efficiency of *Geo-Cut* with five real-world graphs and three different graph algorithms, and compare it with four state-of-the-art graph partitioning methods. Evaluation results show that *Geo-Cut* can reduce the inter-DC data transfer time by up to 79% (42% as the median) and reduce the monetary cost by up to 75% (26% as the median) compared to the other methods. Regarding the overhead, *Geo-Cut* takes less than three seconds to partition one million edges for small size graphs such as GoogleWeb ( $\sim 10^6$  edges) and around 100 seconds for large size graphs such as Twitter ( $\sim 10^9$  edges).

The following of this paper is organized as below. Section 2 introduces the background and related work of the pa-

TABLE 1: Uplink/downlink bandwidths of cc2.8xlarge instances from three Amazon EC2 regions to the Internet. Prices are for uploading data out of the regions to the Internet.

	US East	AP Singapore	AP Sydney
Uplink Band. (GB/s)	0.52	0.55	0.48
Downlink Band. (GB/s)	2.8	3.5	2.5
Price (\$/GB)	0.09	0.12	0.14

TABLE 2: Uplink/downlink bandwidths of A8 instances from three Windows Azure cloud regions to the Internet. Prices are for uploading data out of the regions to the Internet.

	US East	West Europe	Japan East
Uplink Band. (MB/s)	126	179	128
Downlink Band. (MB/s)	174	295	165
Price (\$/GB)	0.087	0.087	0.12

per. We formulate the graph partitioning problem in Section 3 and introduce our proposed techniques in Sections 4 and 5. We evaluate *Geo-Cut* in Section 6 and conclude this paper in Section 7.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Geo-distributed Datacenters

Many cloud providers and large companies are deploying their services globally to guarantee low latency to users around the world. In such environments, user data are stored and updated in local DCs to provide low-latency services to local users and are transferred through the WAN to other DCs for collective computations. We measure the uplink/downlink bandwidths from/to the US East, Asia Pacific Singapore and Sydney regions of Amazon EC2 to/from the WAN using cc2.8xlarge instances and have the following observations.

*Observation 1: the uplink/downlink bandwidths of a single DC can be heterogeneous.* As shown in Table 1, the downlink bandwidths of all the three regions are several times higher than their uplink bandwidths. This is mainly due to the different link capacities and resource sharing among multiple applications [13]. Our evaluation on Amazon EC2 shows that many instance types, such as c4.4xlarge, cc2.8xlarge and m4.10xlarge, all show significant differences between their uplink and downlink bandwidths.

*Observation 2: the bandwidths of different DCs are also heterogeneous.* For example, the uplink and downlink bandwidths of the Singapore region are 17% and 40% higher than those of the Sydney region, respectively. This is mainly due to the differences of hardware and the amount of workloads between different cloud regions [14].

*Observation 3: using WAN bandwidth can be pricy.* Using network bandwidth within a single DC is usually fast and cheap. However, it is not the case when using WAN in geo-distributed DCs. For example, data transfer within the same region of Amazon EC2 is usually free of charge, while sending data to the Internet can be very pricy as shown in Table 1. This is because the providers of geo-distributed DCs have to rent WAN bandwidth from Internet Service Providers (ISPs) and pay accordingly for the WAN usage.

*Observation 4: WAN prices are heterogeneous and dynamic.* First, prices of uploading and downloading data from/to the same DC can be different. For example, Amazon EC2 charges for transferring data from DC to the Internet while

downloading data from the Internet to DC is free of charge. Second, the inter-DC networking prices of different DCs are heterogeneous. As shown in Table 1, transferring data from Sydney region to the Internet is much more expensive than that from the US East region. Third, there is an increasing interest in using dynamic WAN pricing to improve the utilization of Internet bandwidths [16]. WAN prices can be lowered when using WAN at off-peak time or in less congested DCs. Facing the complicated WAN pricing models, it is non-trivial to optimize the monetary cost of WAN usage.

The same observations can be found on other clouds with geo-distributed DCs such as Windows Azure. As shown in Table 2, in the same DC, the downlink bandwidth of A8 instances in West Europe is 39% higher than the uplink bandwidth. Across different DCs, the downlink bandwidth of West Europe is 44% higher than that of Japan East. However, the network price of Japan East is the highest among the three regions compared, which is 37% higher than the other two. It is also interesting to note that there exists no strong correlation between the bandwidth and price heterogeneities, which makes the optimization of the performance and cost objectives at the same time more complicated.

## 2.2 Graph Partitioning Methods

Existing main graph processing engines adopt either edge-cut [5], [6] or vertex-cut [7], [17] method to partition graphs onto multiple machines by cutting them through edges or vertices, respectively. As shown in Figure 1, edge-cut replicates cross-partition edges which causes data communication between partitions. For example, when vertex 1 needs to receive data from vertex 2, there is data communication between partition  $p1$  and  $p2$ . On the other hand, vertex-cut replicates cross-partition vertices. When vertex 1 needs data from vertex 2, the data is sent to the mirror replica of vertex 1, which is located in the same partition as vertex 2. Although it also causes communication between replicas of vertex 1, it has the benefit of maintaining data locality for vertices. Existing studies find that vertex-cut is more efficient than edge-cut for natural graphs which has a small number of vertices with very high degrees [7]. As we focus on real-world graphs which usually follow power-law degree distribution, we focus on vertex-cut in this paper.

With the increasing sizes of real-world graphs, there have been many studies proposed to provide fast and efficient partitioning for large graphs. Some studies focus on distributed graph processing in order to reduce communication cost and obtain good load balance at the same time [18], [19], [20], [21]. Among those studies, the one-pass streaming-based partitioning method has been widely adopted [22]. Xu et al. [23] propose a heterogeneity-aware streaming graph partitioning method, which considers the heterogeneous computing and communication abilities when placing graph vertices to different machines. Mayer et al. [24] propose a window-based streaming graph partitioning method which selects the best edge from a set of edges during edge assignment to improve the quality of partitioning results. However, their method cannot be applied directly to the geo-distributed graph partitioning problem, due to the both

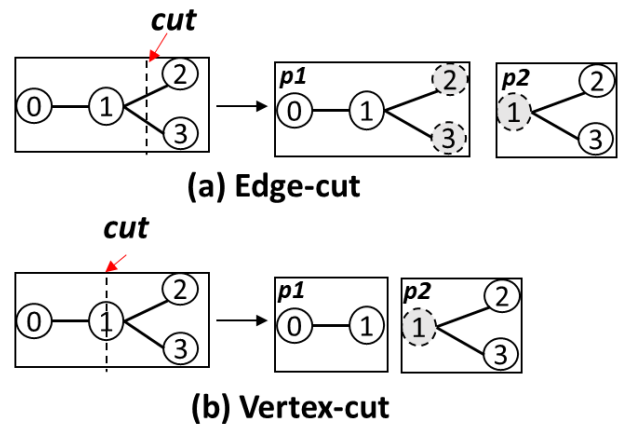


Fig. 1: Illustrations of (a) edge-cut and (b) vertex-cut. Shaded vertices represent ghosts and mirrors respectively.

inter- and intra-DC network heterogeneities in geo-distributed DCs. Mayer et al. [10] propose an adaptive streaming graph partitioning method named Graph, which considers the heterogeneity in vertex traffic and networking prices and aims at minimizing the communication cost during graph processing. However, Graph is not aware of the multi-level network bandwidth heterogeneities in geo-distributed DCs, and thus can lead to large inter-DC data transfer time. Zhou et al. [25] propose a graph partitioning method, which considers the multi-level network bandwidth heterogeneities in geo-distributed DCs to optimize inter-DC data transfer time while satisfying WAN usage constraint. However, this work does not consider the price heterogeneity of geo-distributed DCs and thus can lead to high inter-DC data transfer cost even with a low WAN usage.

This paper significantly extends a preliminary conference version [25] by considering the cost efficiency of geo-distributed graph processing. Although existing works have studied the graph partitioning problem with various criteria, none of them has considered the problem with both performance and cost as optimization objectives at the same time in geo-distributed DCs, where the network bandwidth and prices are both highly heterogeneous and quite different from those within a single data center. To tackle this challenging problem, we make extensions in three major aspects. First, we reformulate the graph partitioning problem with minimizing inter-DC data transfer time as the optimization goal and monetary budget on runtime inter-DC data communication cost as the constraint. Second, to solve the new problem efficiently, we propose a cost-aware streaming partitioning heuristic to optimize the inter-DC data communication cost and propose a new partition mapping strategy to more efficiently reduce the inter-DC data transfer time while satisfying the budget constraint. Third, we extend our experiments to an additional real-world cloud platform and using a new graph with larger size.

## 2.3 A Motivating Example

We give a simple example to demonstrate the importance of a geo-aware graph partitioning method. Figure 2 shows a graph with eight vertices, where the input data of vertices are initially

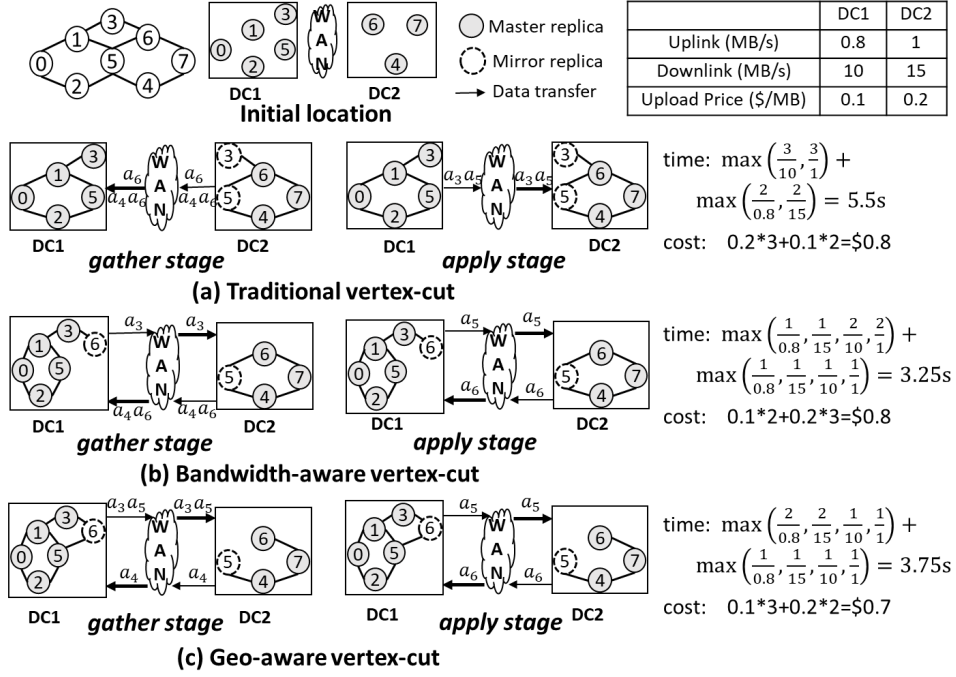


Fig. 2: A comparison between three graph partitioning methods: (a) traditional vertex-cut, (b) bandwidth-aware vertex-cut and (c) geo-aware vertex-cut. Widths of the up/down links indicate network bandwidths.

distributed in two DCs. The initial locations are selected as the master replica of each vertex. We consider three motivating partitioning methods, including a traditional vertex-cut [7], a bandwidth-aware vertex-cut [25] and the geo-aware vertex-cut method proposed in this paper.

We adopt the widely-used GAS model [7] as our graph processing model. With the vertex-centric abstraction of GAS, we can easily partition graphs using vertex-cut to parallelize vertex computations. One of the replicas for each vertex is selected as the *master* and all the other replicas are called *mirrors*. GAS model supports the bulk synchronous model of computation and there are three computation stages in each GAS iteration, namely **G**ather (Sum), **A**pply and **S**catter. In the gather stage, each replica gathers data from its local neighbors and sends the gathered data to the master. The master aggregates all received data into the gathered sum. In the apply stage, the master updates vertex data using the gathered sum and sends the updated data to all mirrors. In the scatter stage, each replica activates its local neighbors. In this example, we assume the sum function of the graph algorithm aggregates two messages by simply binding them together and the data sizes of all vertices are the same (i.e.,  $a_i = 1$  MB). The inter-DC data transfer time can be calculated as the sum of the time spent on the gather (from mirrors to master) and apply (from master to mirrors) stages. The inter-DC data transfer cost is calculated as the sum of the data uploading cost of each DC.

Assuming the budget constraint for the problem to be \$0.7, we obtain the partitioning results as shown in Figure 2. We have the following observations. First, compared to the traditional vertex-cut, geo-aware vertex-cut obtains both lower inter-DC data transfer time and cost. This is partly because that traditional vertex-cut is not aware of the bandwidth and

TABLE 3: Notation overview.

Symbol	Meaning
$M$	The number of geo-distributed DCs
$R_v$	The set of DCs containing at least one replica of $v$
$I_v^r$	A boolean value indicating whether the replica of vertex $v$ in DC $r$ is the master ( $I_v^r = 1$ ) or not ( $I_v^r = 0$ )
$d_v$	The input data size of vertex $v$
$g_v^r(i)$	The aggregated data size transferred from the mirror in DC $r$ to the master of vertex $v$ during the gather stage in iteration $i$
$a_v(i)$	The combined data size sent from the master of vertex $v$ to each mirror in the apply stage of iteration $i$
$U_r/D_r$	Uploading/Downloading bandwidth of DC $r$
$P_r$	Unit price of uploading data from DC $r$ to the Internet
$B$	The monetary budget of WAN usage

price heterogeneities, and tends to get a partition solution with good load balance as shown in Figure 2(a). This results in long inter-DC data transfer time at apply stage by transferring more data through the uplink of DC 1, and high inter-DC data communication cost at the gather stage by transferring more data via the pricy uplink of DC 2. Second, the bandwidth-aware method obtains shorter inter-DC data transfer time than geo-aware, but has higher inter-DC data communication cost due to its ignorance of the network price heterogeneity. Specifically, both methods transfer the same amount of data through WAN, while geo-aware sends less data using the pricy uplink of DC 2 and more with the cheap uplink of DC 1. This example demonstrates that it is important to consider both the network bandwidth and price heterogeneities to obtain a good trade-off between the cost and performance of geo-distributed graph processing.

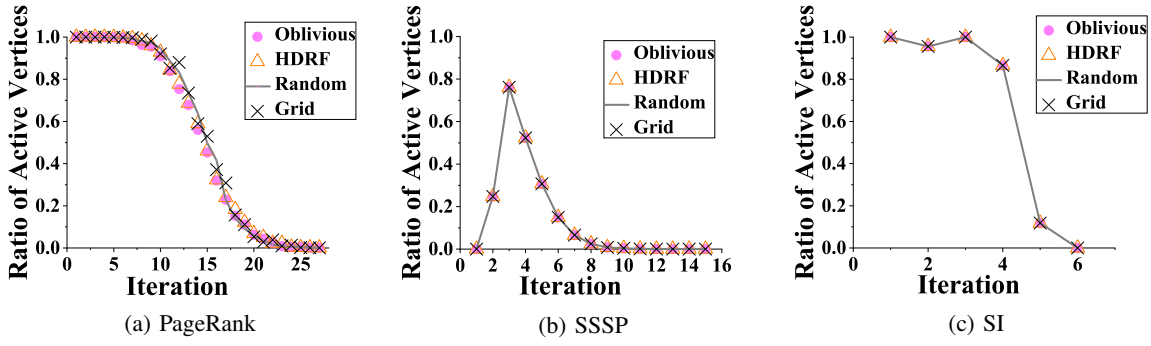


Fig. 3: Convergence analysis of PageRank, SSSP and SI under different strategies. The convergence speeds under the four partitioning strategies are exactly the same for SSSP and SI.

### 3 PROBLEM FORMULATION

#### 3.1 Model Overview

We study how to partition a large graph onto multiple geo-distributed DCs. The graph data are generated and stored geo-distributedly. For simplicity, we assume that the graph data are not replicated initially and each machine executes only one vertex replica at a time. We assume that there are unlimited computation resources in each single DC, and inter-DC data communication is the performance bottleneck of geo-distributed graph processing. This assumption is valid in the geo-distributed environment since the WAN bandwidth is much more scarce than the computation resources such as CPU and memory. We also assume that DCs are connected with a congestion-free network and the bottlenecks of the network are only from the uplinks/downlinks of DCs [13]. This assumption is based on the observation that many datacenter owners are expanding their services world widely and are very likely to build their own private WAN infrastructure [26]. We assume that only uploading data from a DC to WAN is charged. This assumption is based on the pricing scheme of most public clouds such as Amazon EC2 and Windows Azure. The graph partitioning algorithm is applied before executing graph processing algorithms.

Before formally defining the problem, we first study the impact of graph partitioning on the convergence of graph algorithms. We execute three graph algorithms, namely PageRank [27], Single Source Shortest Path (SSSP) [28] and Sub-graph Isomorphism (SI) [29], using PowerGraph engine and a synthetic power-law graph with 1,000,000 vertices ( $\lambda = 2.1$ ). We use four existing graph partitioning strategies implemented in PowerGraph to partition the graph, namely Random [7], Grid [7], HDRF [30] and Oblivious [7]. Figure 3 shows the convergence speeds of the three graph algorithms using different graph partitioning strategies. The convergence speed is evaluated using the ratio of active vertices among all. When a graph algorithm is converged, all vertices have finished their computation and become inactive. Interestingly, we observe that the convergence speeds are hardly affected by the partitioning strategies: the convergence speeds of each graph algorithm are almost the same when using different graph partitioning strategies. This means that we can optimize the performance of graph processing jobs by optimizing each processing

iteration. In the following, we model the graph processing performance using one iteration.

#### 3.2 Problem Definition

Table 3 summarizes the symbols used in problem formulation. Consider a graph  $G(V, E)$  with input data stored in  $M$  geo-distributed DCs, where  $V$  is the set of vertices and  $E$  is the set of edges in the graph. Each vertex  $v$  ( $v = 0, 1, \dots, |V| - 1$ ) has an initial location  $L_v$  ( $L_v \in [0, 1, \dots, M - 1]$ ), which is where the input data of vertex  $v$  is stored.

With the distributed GAS model, the inter-DC network traffic mainly comes from the gather stage and the apply stage. For a given iteration  $i$  and a vertex  $v$ , each mirror in DC  $r$  sends aggregated data of size  $g_v^r(i)$  to the master of  $v$  in the gather stage and the master sends the combined data of size  $a_v(i)$  to each mirror to update the vertex data in the apply stage. To simplify the calculation of data transfer time, we assume there is a global barrier between the gather stage and the apply stage. Thus, the inter-DC data transfer time in iteration  $i$  can be formulated as the sum of the data transfer times in gather and apply stages. In each DC, the data transfer finishes when it is finished on both up and down links. Thus, we have:

$$T(i) = T_G(i) + T_A(i) = \max_r T_G^r(i) + \max_r T_A^r(i) \quad (1)$$

$$T_G^r(i) = \max\left(\frac{\sum_v I_v^r \cdot \sum_{k \in R_v, k \neq r} g_v^k(i)}{D_r}, \frac{\sum_v (1 - I_v^r) \cdot g_v^r(i)}{U_r}\right) \quad (2)$$

$$T_A^r(i) = \max\left(\frac{\sum_v I_v^r \cdot a_v(i) \cdot (|R_v| - 1)}{U_r}, \frac{\sum_v (1 - I_v^r) \cdot a_v(i)}{D_r}\right) \quad (3)$$

where  $I_v^r$  is a boolean indicator showing whether the replica of vertex  $v$  in DC  $r$  is the master ( $I_v^r = 1$ ) or not ( $I_v^r = 0$ ).  $R_v$  is the set of DCs containing at least one replica of  $v$  and initially contains only  $L_v$ .  $U_r$  and  $D_r$  are the uplink and downlink bandwidths of DC  $r$ , respectively.

The inter-DC data communication cost is calculated as the sum of the cost on data uploading during the gather and apply stages. Denote the unit price of uploading data from DC  $r$  to the Internet as  $P_r$ , we have:

$$C_{comm}(i) = \sum_v \sum_{r \in R_v} P_r \cdot [I_v^r \cdot dm_v + (1 - I_v^r) \cdot g_v^r(i)] \quad (4)$$

where  $dm_v$  equals to  $a_v(i) \cdot (|R_v| - 1)$ , which represents the amount of data uploaded by the master copy of  $v$ .

Based on the above analysis, we formulate our geo-distributed graph partitioning problem as the following constrained optimization problem:

$$\min T(i) \quad (5)$$

$$\text{s.t.} \quad C_{comm}(i) \leq B \quad (6)$$

Note that the cost and performance objectives can be contradictory with each other. For example, as shown in Figure 2, the uplink of DC 2 has higher bandwidth and thus is useful for reducing the inter-DC data transfer time. However, the price of this link is also higher and hence has an adverse impact on the cost objective. Thus, it is more complicated to find a good graph partitioning solution for our problem than existing studies which consider only one of the two objectives [10], [25].

## 4 GEO-AWARE GRAPH PARTITIONING

In this paper, we propose a geo-aware graph partitioning algorithm named *Geo-Cut*, which incorporates two optimization stages to reduce optimization overhead. In the first stage, we study the traffic pattern during graph processing and propose a streaming heuristic which considers the network price heterogeneity to minimize inter-DC data transfer cost. In the second stage, we propose two partition refinement heuristics which consider the network bandwidth heterogeneity to reduce the inter-DC data transfer time without violating the budget constraint. In the following, we introduce the two optimization stages in detail.

### 4.1 Cost-Aware Streaming Graph Partitioning

Given the initial locations of vertices (i.e., where the input data of vertices are located), we adopt the streaming graph partitioning approach to quickly partition a graph. We view a graph as a stream of edges  $e_0, \dots, e_{|E|-1}$  to be assigned to a graph partition. The number of partitions is the same as the number of DCs. The order of edges in the set, which determines the edge that will be assigned first by the streaming partitioning method, can be decided randomly or using breadth- or depth-first traversals. Existing studies have shown that the random order of edges can produce nearly the same result as that produced by optimized edge orders [31]. Thus, we randomly order the edges in the stream. One important design parameter in streaming partitioning is the heuristic which decides where to place an incoming edge. In this paper, we design a cost-aware streaming heuristic that aims at minimizing the inter-DC data communication cost.

As only the uploading network traffic is charged in geo-distributed DCs, we study the additional uploading traffic introduced by edge assignment. We select the initial replica of each vertex as the master replica. When placing an edge  $(u, v)$  in a DC  $r$ , it can cause two types of additional uploading traffic. First, the replication of vertex  $u$  ( $v$ ) in DC  $r$  increases the uploading traffic in the initial location of  $u$  ( $v$ ) by  $a_u(i)$  ( $a_v(i)$ ), which is used to synchronize the updated vertex data between the master and the added mirror replica of  $u$  ( $v$ ) in the apply stage of the GAS model. Second, the data passed along

the edge  $(u, v)$  increases the uploading data size of DC  $r$  when the mirror replicas send local aggregated data of  $u$  and/or  $v$  to the master replicas in the gather stage. Denote the aggregated data size transferred from the mirror of a vertex  $v$  in DC  $r$  to the master replica during the gather stage in iteration  $i$  as  $g_v^r(i)$ . We calculate the increased gathering traffic size caused by placing an edge  $(u, v)$  in DC  $r$  in iteration  $i$  as follows. Without loss of generality, we assume graphs are directed and data are gathered from in-edges.

$$\Delta g_{u,v}^r(i) = \text{Sum}(g_v^r(i), a_u(i-1)) - g_v^r(i) \quad (7)$$

Based on the above analysis of graph traffic pattern, we derive the following streaming heuristic which always tries to place an incoming edge  $(u, v)$  to the DC that leads to the lowest increased inter-DC data transfer cost.

- 1) If  $R_v$  and  $R_u$  intersect, place edge  $(u, v)$  in any  $r \in R_v \cap R_u$  with the lowest  $C_{gather}^r$ , where

$$C_{gather}^r = (1 - I_v^r) \cdot P_r \cdot \Delta g_{u,v}^r(i) \quad (8)$$

- 2) If  $R_v$  and  $R_u$  do not intersect, place  $(u, v)$  in a DC  $r$  with the lowest sum of  $C_{gather}^r$  and the synchronization cost  $C_{sync}^r$ , where

$$C_{sync}^r = \begin{cases} P_{L_u} \cdot a_u(i) & \text{if } r \in R_v \\ P_{L_v} \cdot a_v(i) & \text{if } r \in R_u \\ P_{L_u} \cdot a_u(i) + P_{L_v} \cdot a_v(i) & \text{otherwise} \end{cases} \quad (9)$$

After the edge assignment, we build local subgraphs in each DC and create vertex replicas as needed. The time complexity of the streaming graph partitioning is  $O(|E|)$ . As our streaming heuristic prioritizes DCs which do not require replicating vertices, our partitioning method can also result in a low vertex replication rate and hence low data movement cost.

### 4.2 Performance-Aware Partition Refinement

After the cost-aware partitioning, in this stage, we consider the highly heterogeneous network bandwidth in geo-distributed DCs and propose two heuristics to reduce the inter-DC data transfer time while satisfying the budget constraint. First, we propose a simple yet effective partition mapping heuristic which iteratively tries to switch the mapping of partitions to reduce the inter-DC data transfer time while satisfying the budget constraint. Second, we consider the network bandwidth heterogeneity and use edge migrations to diminish the data traffic in bottlenecked DCs to further reduce the inter-DC data transfer time while satisfying budget constraint.

#### 4.2.1 Partition Mapping

Mapping the  $M$  graph partitions to  $M$  geo-distributed DCs is a classic combinatorial NP-hard problem and has a solution space of  $O(M!)$ . For small values of  $M$ , we can simply adopt BFS/DFS-based search algorithms to find the best mapping solution. Specifically, in our problem, a mapping solution is better than another when it obtains less inter-DC transfer time while satisfying monetary budget. For a large number of DCs, it is however non-trivial to design a heuristic for finding the best partition mapping due to the network heterogeneities in both bandwidths and prices.

We propose a simple yet effective algorithm for quickly finding a good graph partition mapping solution. The algorithm is inspired by the “power of two choices” [32] technique, which has been demonstrated effective in task scheduling problems for finding the approximately optimal scheduling solution with low latency [33]. Recent studies have further discovered that using  $d > 2$  choices can provide orders-of-magnitude improvements [34] when task sizes are heavy-tailed. Similar to task scheduling problems, our partition mapping problem is also a balls-to-bin scheduling problem with some tailed “tasks” due to network heterogeneities. Thus, we can expect the “power of many choices” technique to be effective for our problem. Our partition mapping algorithm is shown in Algorithm 1.

---

**Algorithm 1** Graph partition mapping algorithm.
 

---

**Require:**  $P_{init}$ : the initial partition mapping solution;  
**Ensure:**  $P_{opt}$ : the optimized partition mapping solution;  
 1:  $P_{opt} = P_{init}$ ;  
 2:  $continue = true$ ;  
 3: **repeat**  
 4: Randomly sample  $d$  pairs of partitions;  
 5: Let  $G_b = 0$  and  $p_b = 0$ ;  
 6: **for**  $i = 1$  to  $d$  **do**  
 7:  $(G_i, cost_i) = \text{EstimateGain}(pair_i, P_{opt})$ ;  
 8: **if**  $G_i > G_b$  and  $cost_i \leq B$  **then**  
 9: Let  $G_b = G_i$  and  $p_b = i$ ;  
 10: **if**  $G_b > 0$  **then**  
 11: Switch the mapping of partitions in  $pair_{p_b}$  to generate plan  $P_{opt}$ ;  
 12: **else**  
 13:  $continue = false$ ;  
 14: **until**  $!continue$   
 15: **return**  $P_{opt}$ ;

---



---

**Algorithm 2** EstimateGain( $pair, P$ )
 

---

**Require:**  
 $pair$ : the pair of partitions to be evaluated;  
 $P$ : the current partition mapping solution;  
**Ensure:**  
 $G$ : the estimated gain of switching the pair of partitions in  $P$ ;  
 $C$ : the inter-DC data communication cost after switching partitions in  $P$ ;  
 1: Let  $G = 0$  and  $C = 0$ ;  
 2: Generate plan  $P'$  by switching the partition mapping of  $pair$ ;  
 3:  $G = \text{EstimateTime}(P) - \text{EstimateTime}(P')$ ;  
 4:  $C = \text{EstimateCommCost}(P')$ ;  
 5: **return**  $G$  and  $C$ ;

---

Starting from an initial partition mapping solution, we randomly select  $d$  pairs of partitions (Line 4) and estimate the gain of switching their mapping. The initial partition mapping solution can be the partitioning result obtained by the first optimization stage or an optimized solution given by any other graph partitioning method. The gain of switching the mapping of two partitions is calculated as the reduced inter-DC data transfer time if the switched partition mapping still satisfies the budget constraint (otherwise the gain is zero) (Line 7). We select the pair with the best gain among the  $d$  pairs and switch its partition mapping (Line 8-9). Iteratively, we repeat the above process until no improvement can be obtained or a predefined number of iterations has been reached. The choice of  $d$  is related to the severity of network performance heterogeneity, which corresponds to the tail length in task scheduling problems. By default, we choose  $d = 2$ . We estimate the gain and cost of switching the mapping of two partitions using Algorithm 2. The gain is calculated as the difference between

the estimated data transfer time before and after the switching (Line 3). The data transfer time is estimated using Equation 1 which characterizes the performance of GAS model and the data transfer cost is estimated using Equation 4.

*Complexity Analysis:* According to Equations 1 and 4, the time complexity of Algorithm 2 is  $O(M)$ . Assuming the predefined maximum number of iterations to sample new pairs of partitions for switching is  $MaxIter$ , the overall time complexity of our partition mapping algorithm is  $O(MaxIter \times M)$ , which is much faster than the naive search algorithms.

#### 4.2.2 Edge Migration

Network bandwidth heterogeneity can cause performance bottlenecks even with a good partition mapping. For example, as shown in Table 1, both the uplink and downlink bandwidths of Sydney are lower than the other two cloud DCs. Thus, an equal partition of the graph workload will lead to performance bottleneck on the Sydney DC. To mitigate such performance bottlenecks, we propose to migrate edges out of the bottlenecked DCs after obtaining a good partition mapping.

According to Equations 1–3, we can identify the bottlenecks of inter-DC data communication by estimating the data transfer time in the gather and apply stages for each DC  $r$ , i.e.,  $T_G^r(i)$  and  $T_A^r(i)$ , respectively. The bottlenecks for the gather (apply) stage are the DCs which have the same data transfer time as  $T_G(i)$  ( $T_A(i)$ ). The bottleneck can be bounded either on the uplink or the downlink of the DC. Thus, there can be four types of bottlenecked links, namely uplink/downlink bounded in the gather stage and uplink/downlink bounded in the apply stage. If the gather and apply stages are bounded on the same link of the same DC, we migrate edges out of that DC considering the reduction to the inter-DC data transfer time. Otherwise, we select the link with more possible reduction to the data transfer time of the gather and apply stages to migrate. Note that, in both cases, the migration is allowed only when the budget constraint is not violated.

Based on Equations 1–3, we address a bottlenecked link  $l_r$  of DC  $r$  as follows. If  $l_r$  is uplink and bounds the gather stage, the network traffic on  $l_r$  is mainly caused by mirror replicas in the DC sending gathering data to the masters. Thus, we order the mirror replicas according to their gathering data sizes (i.e.,  $g_v^r(i)$  for the mirror of vertex  $v$  in DC  $r$ ) in a priority queue  $Q$ . We iteratively remove the vertices in  $Q$  until  $l_r$  is no longer the bottleneck or  $Q$  is empty. If  $l_r$  is downlink and also bounds the gather stage, the network traffic on  $l_r$  is mainly caused by master replicas receiving the gathering data from mirrors. Thus, we order the master replicas in DC  $r$  according to their non-local gathering data sizes. After removing the master replicas, we select the replica with the largest number of local neighbors in each DC as the master of a vertex, in order to further reduce the inter-DC network traffic. Similarly, for bottlenecks of the apply stage, if  $l_r$  is uplink (downlink), we order the master (mirror) replicas in DC  $r$  according to the vertex data sizes. After removing a vertex replica  $v$ , we migrate each edge connected to  $v$  to a DC which results in the shortest inter-DC data transfer time. We calculate the inter-DC data transfer time after migrating an edge using Equation 5 with the updated inter-DC network traffic. By default, we migrate



one edge at a time.

*Complexity Analysis:* The average time complexity of the edge migration can be calculated as  $O(|Q| \times \frac{|E|}{|V|} \times M)$ . As  $|Q|$  is on average  $\frac{|V|}{M}$ , we have the time complexity as  $O(|E|)$ . To balance the trade-off between the effectiveness and efficiency of edge migrations, we provide two optimizations. First, as introduced in Section 2, it is common in power-law graphs that a small portion of vertices are contributing to most of the data traffic. Thus, we can limit the length of  $Q$  to a small value  $L_Q$  while achieving similar migration results. Second, for each vertex candidate  $v$  in  $Q$ , we group the edges connected to  $v$  into  $C$  groups, where edges in the same group will be migrated at the same time to the same DC. We adopt the clustering method to group edges, and the distance between two edges is defined as the intersection size between the replication locations of the other end of the edges. For example, for two edges  $(u, v)$  and  $(w, v)$ , the distance between them is defined as  $|R(u) \cap R(w)|$ . In this way, we can minimize the number of additional vertex replications and hence the size of additional inter-DC data transfer caused by edge migrations. With the two optimizations, we are able to reduce the time complexity of edge migration to  $O(L_Q \times C \times M)$ .

### 4.3 A Concrete Example

To better illustrate our two-stage optimizations method, we present a concrete example to partition a small graph with seven vertices and six edges using Geo-Cut. The initial locations of the vertices are shown in Figure 4(a).

First, we apply our cost-aware streaming graph partitioning method to get an initial partitioning solution. Assume the first edge to be assigned is edge  $(0, 1)$ , we show how to place it in Figure 4(b). As  $R_0$  and  $R_1$  do not intersect, according to Equation 8 and 9, we can calculate the total increased cost of placing the edge in DC1 and DC2 to be \$0.3 and \$0.1, respectively. Thus, edge  $(0, 1)$  should be placed in DC2. After placing all edges using the same way, we get an initial partitioning solution for the graph.

Second, we apply the partition mapping refinement technique. In this example, switching the mapping of the two partitions does not bring any gain, so the partitioning solution remains unchanged.

Third, we identify the bottlenecks of inter-DC data communication and apply the edge migration refinement. Based on equations 1–3, we calculate the time of the apply and gather stages to be both  $2s$ , and we can identify the uplink of DC1 to be the bottleneck of the apply stage. To address this bottleneck, we order vertices 0, 2, 5 and 6 according to their data sizes. As all vertices have the same data size, we choose to remove vertex 0 and migrate edge  $(6, 0)$  from DC1 to DC2. After edge migration, the time of the apply and gather stages are reduced to  $1s$  and  $0s$ , respectively.

## 5 PARTITIONING DYNAMIC GRAPHS

With Geo-Cut, we are able to partition a static graph in  $O(|E|)$  time, assuming  $|E| \gg M$ . However, many real-world graphs such as social network graphs are dynamic, with frequent vertex and edge insertions/deletions (e.g.,

newly registered/deactivated users and following/unfollowing operations). These changes to a graph can greatly affect graph partitioning decisions. For example, a deactivated celebrity Facebook user, who usually has a large number of followers, can greatly affect the network traffic of the new social network graph. Repartitioning the entire graph once changes occur is not cost-effective due to its large overhead. In the following, we discuss how to use Geo-Cut to adaptively partition dynamic graphs with a low overhead.

Since adding and removing a vertex can be represented as adding and removing edges connected to this vertex, we abstract the changes to a dynamic graph as edge insertions and edge deletions, assuming that a graph has no isolated vertex. It has been pointed out by existing studies [35] that a dynamic graph can be viewed as an intermediate state of the streaming graph partitioning. Thus, we can adopt the streaming graph partitioning technique of Geo-Cut to directly assign inserted edges to DCs. The challenge is that, the inserted/deleted edges can change the data traffic of vertices and thus make the existing partitioning less effective. For example, when inserting a large number of edges connected to an originally low-degree vertex  $v$ , it is better to move  $v$  to a DC with high bandwidths.

To address this challenge, we periodically apply our partition refinement technique to the updated graph partitions. Specifically, we use partition mapping to relocate the graph partitions which have new edges inserted. We use the edge migration technique to migrate edges from the current bottlenecked DC to DCs which have deleted edges, if the inter-DC data transfer time can be reduced. It is costly to perform the refinement every time an edge is inserted/deleted. Thus, we need to decide the timing of partition refinement. Specifically, we calculate the increased/decreased data traffic size to a partition caused by edge insertions/deletions using the same way as introduced in streaming graph partitioning. We trigger the partition refinement whenever the number of changes to data traffic caused by edge insertions/deletions is larger than a threshold. By default, we set *threshold* to 10% of the overall data traffic size of a partition. The time complexity of inserting a set of edges  $E'$  is  $O(|E'| + MaxIter \times R)$  and that of deleting a set of edges is  $O(L_Q \times C \times R)$ , where  $R$  is the number of partition refinement operations applied. This is much faster than repartitioning the entire graph.

## 6 EVALUATION

We evaluate the effectiveness and efficiency of Geo-Cut using real-world graph datasets on both real clouds and a cloud simulator. We use Amazon EC2 and Windows Azure for real cloud experiments. To emulate the congestion-free network model, we limit the uplink and downlink bandwidths of the instances to be smaller than the WAN bandwidth. The limited bandwidths are proportional to their original bandwidths. We adopt the GAS-based PowerGraph [7] system to execute graph processing algorithms. The evaluated graph partitioning methods are implemented using C++ programming language in PowerGraph to partition graphs while loading. We adopt a multi-threaded implementation to parallelize the streaming graph partitioning and to reduce the ingress overhead.

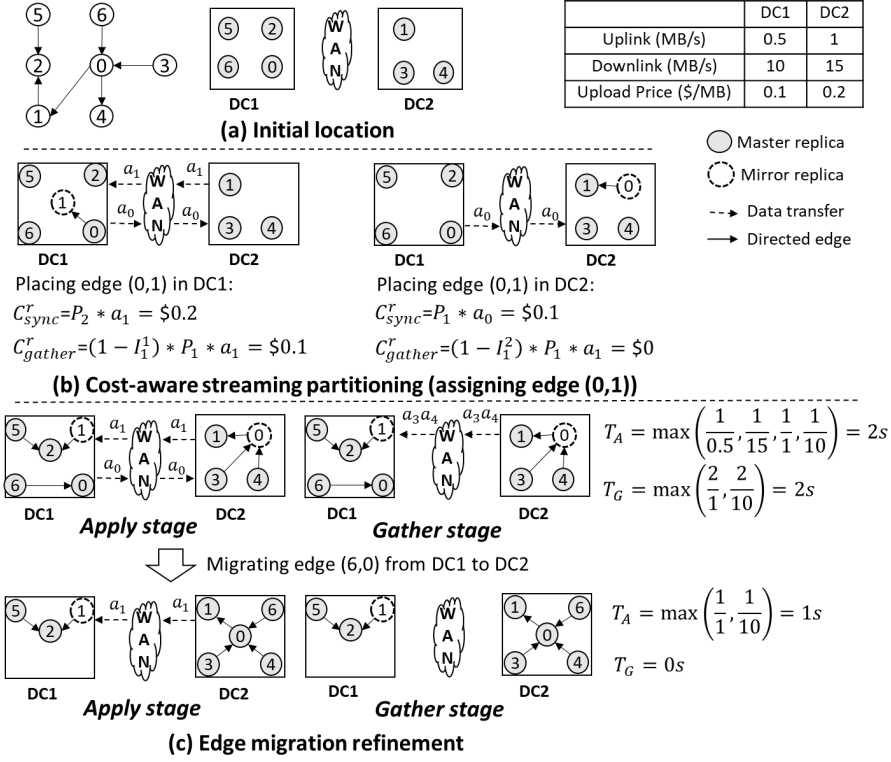


Fig. 4: A concrete example of applying the two-stage optimizations of Geo-Cut to partition a graph.

TABLE 4: Experimented real-world graphs.

Notation	#Vertices	#Edges	$\alpha_{in}$	$\alpha_{out}$
Gnutella (GN)	8,104	26,013	2.91	2.59
WikiVote (WV)	7,115	103,689	3.63	3.80
GoogleWeb (GW)	875,713	5,105,039	2.96	3.64
LiveJournal (LJ)	3,577,166	44,913,072	3.45	2.88
Twitter (TW)	41,652,230	1,468,365,182	3.29	3.55

## 6.1 Experimental Setup

**Graphs.** We select five real-world graphs for our experiments, which are representative graphs in P2P networks, social networks and web graphs. Table 4 gives details of the graphs [36].

**Graph algorithms.** We adopt three graph algorithms which are widely used in different areas, including PageRank (PR) [27], Single Source Shortest Path (SSSP) [28] and Subgraph Isomorphism (SI) [29].

PR is widely used in web information retrieval to evaluate the relative importance of webpages. SSSP finds the shortest paths starting from a single source to all other vertices in the graph. SI is used to find the subgraphs matching certain graph pattern in a large graph.

**Compared methods.** We compare Geo-Cut with the following state-of-the-art vertex-cut graph partitioning methods. We adopt the original implementation of these algorithms if open-sourced<sup>1</sup> and implement our own version if otherwise.

**Baseline** [7] uses random hashing to assign an edge to a DC containing the source or target node of the edge. If such a DC cannot be found, it assigns the edge to a random DC. Baseline is used as the baseline comparison, as what has been done in the other compared methods [7], [10], [25].

**Greedy** [7] iteratively places edges to the DCs which minimize the expected vertex replication factor.

**Graph** [10] considers the heterogeneity in vertex traffic and network pricing, and assigns edges to minimize the data communication cost.

**G-Cut** [25] is our previous work on geo-distributed graph partitioning, which considers the network bandwidth heterogeneity to optimize graph processing performance while satisfying WAN usage constraint.

**Configuration details.** We use both real cloud experiments and simulations to evaluate the effectiveness of Geo-Cut.

**Real cloud experiments.** We select eight regions of Amazon EC2 as the geo-distributed DCs, namely US East (USE), US West Oregon (OR), US West North California (NC), EU Ireland (EU), Asia Pacific Singapore (SIN), Asia Pacific Tokyo (TKY), Asia Pacific Sydney (SYD) and South America (SA). With Azure, we select four regions, including East US (US), West Japan (JP), West Europe (EU) and Southeast Australia (AUS). In each region, we construct a cluster of five c-c2.8xlarge EC2 instances or A8-v2 Azure instances. In all experiments, we compare the performance and monetary cost of graph algorithms optimized by Geo-Cut and the four compared methods. As Graph is designed to optimize the data communication cost for graph algorithms, we set the monetary budget of runtime data communication to the cost of Graph by default. All results are normalized to those of Baseline if not otherwise specified.

**Simulations.** We simulate 20 geo-distributed DCs using network performances and prices measured from Amazon EC2. We perform three sets of simulations using Livejournal graph. First, we construct three types of geo-distributed environ-

1. <https://github.com/rossojr92/PG-ObliviousGreedy-Partitioning>

TABLE 5: Bandwidth and price settings for the bandwidth heterogeneity experiments.

Bandwidth/Price	DC 0-9	DC 10-14	DC 15-19	Heter.
Low	1/1	1/0.5	0.5/0.5	0.74
Medium	1/1	0.5/0.5	0.25/0.5	0.89
High	1/1	0.5/0.5	0.125/0.5	0.96

TABLE 6: Bandwidth and price settings for the price heterogeneity experiments.

Bandwidth/Price	DC 0-9	DC 10-14	DC 15-19	Heter.
Low	1/1	0.5/1	0.25/1	0
Medium	1/1	0.5/0.75	0.25/0.5	0.26
High	1/1	0.5/0.25	0.25/0.125	0.71

ments with “Low”, “Medium” and “High” network heterogeneities and study the impact of network heterogeneities on the effectiveness of Geo-Cut. Specifically, we consider two types of heterogeneities in network bandwidths and prices. We present the network and price settings of the 20 DCs for the bandwidth and price heterogeneity experiments in Tables 5 and 6, respectively. Specifically, we identify the DCs as DC 0-19. We set the network bandwidth and prices of DC 0-9 to be the same as those measured from the US East Region of Amazon EC2 and change the settings of the rest proportionally to obtain different system heterogeneities. For example, for the “High” bandwidth heterogeneity environment, we change the upload/download bandwidths of DC 10-14 and 15-19 to one eighth and half of the measured bandwidths, respectively. We change the network prices of DC 10-19 to half of the original prices, to match the fact that lower bandwidths are usually cheaper in the cloud. To quantitatively define the heterogeneity of network bandwidth, we use the relative standard deviation of the bandwidths of all links (including uplinks and downlinks) in the network as the metric. Similarly, for the heterogeneity of network prices, we use the standard deviation of the prices of all uplinks in the network as the metric. The calculated metrics in Table 5 and 6 show that the bandwidths and prices are in accordance with the heterogeneity settings.

Second, we study the impact of budget constraint on the effectiveness of Geo-Cut. We vary the budget from 1.0, 0.9, ..., to 0.6 times of the default budget and compare Geo-Cut with G-Cut [25], which is a network bandwidth heterogeneity aware graph partitioning method, to show the effectiveness of the cost-aware design in Geo-Cut on satisfying the budget constraint. For fair comparison, we set the WAN usage budget of G-Cut to the runtime WAN usage of Graph by default. When varying the monetary budget of Geo-Cut, we vary the WAN usage budget for G-Cut accordingly in the same way. For all settings, the bandwidths and prices are set as the same as those in the “Medium” price heterogeneity environment.

Third, we study the effectiveness of Geo-Cut on partitioning dynamic graphs under the High network bandwidth heterogeneity. We first study the threshold parameter, which decides how often partition refinements are triggered when partitioning dynamic graphs. With the default threshold, we consider three types of graph changes, including edge insertions, edge deletions and dynamic partitions. For edge insertions, we take 7/8, 3/4 and 1/2 of the edges in a graph as the initial static graph and insert the rest of the edges. For edge deletions, we iteratively remove 1/8, 1/4 and 1/2 of the edges from the

TABLE 7: The absolute values of inter-DC data transfer time obtained by Baseline on Amazon EC2.

Time	GN	WV	GW	LJ	TW
SI	770s	2250s	7.5h	319.4h	575h
PR	3.8ms	11.7ms	136ms	5.3s	9.5s
SSSP	2ms	5.9ms	67.9ms	2.6s	4.8s

TABLE 8: The absolute values of inter-DC data transfer time obtained by Baseline on Windows Azure.

Time	GN	WV	GW	LJ	TW
SI	478s	1670s	5.1h	205.8h	375h
PR	2.3ms	7.3ms	87ms	3.2s	6.2s
SSSP	1.2ms	3.6ms	43.8ms	1.6s	3.1s

initial graph. We reduce the number of partitions from 20 to 15 and reassign the edges in the removed five DCs to the rest DCs. We evaluate the effectiveness and efficiency of dynamic partition refinement.

## 6.2 Real Cloud Results

### 6.2.1 Inter-DC Data Transfer Time

Figures 5 and 6 show the normalized inter-DC data transfer times of the compared algorithms on Amazon EC2 and Windows Azure, respectively. The absolute data transfer times of the baseline algorithm are shown in Table 7 and 8.

We have the following observations. First, on both clouds, Geo-Cut obtains the shortest data transfer time compared to the other methods. Specifically, on Amazon EC2, Geo-Cut reduces the inter-DC data transfer time by 46%-79%, 25%-64%, 19%-64% and 4%-24% compared to Baseline, Greedy, Graph and G-cut, respectively. On Windows Azure, the time reduction are 46%-76%, 17%-64%, 7%-64% and 2%-13% compared to Baseline, Greedy, Graph and G-Cut, respectively. Second, comparing the optimization results of different graphs, Geo-Cut obtains higher time reduction over other compared methods on graphs with more heterogeneous data communications. For example, as shown in Table 4, Gnutella and WikiVote graphs have similar size (i.e., number of vertices) while the latter has more skewed vertex degree distribution (i.e., larger  $\alpha$  value). As a result, the time reduction obtained by Geo-Cut for WikiVote is up to 79% (47% as the median value) on Amazon EC2 and 76% (45% as the median value) on Windows Azure, while the reduction for Gnutella is only up to 68% (34% as the median value) on Amazon EC2 and 66% (25% as the median value) on Windows Azure.

These observations demonstrate that the heterogeneity-aware graph partitioning of Geo-Cut is effective in reducing the inter-DC data transfer time for geo-distributed graph processing.

To better understand the effectiveness of each individual technique proposed in Geo-Cut, and to verify the analysis above, we further breakdown the optimization results of WikiVote graph on both Amazon EC2 and Windows Azure as shown in Figure 9, where Streaming represents optimization with the cost-aware graph partitioning technique only and Placement represents optimization with both Streaming and partition mapping technique. All results are normalized to those of Streaming. We have the following observations.

First, comparing the results obtained by Placement and Streaming, the partition mapping technique is more effective

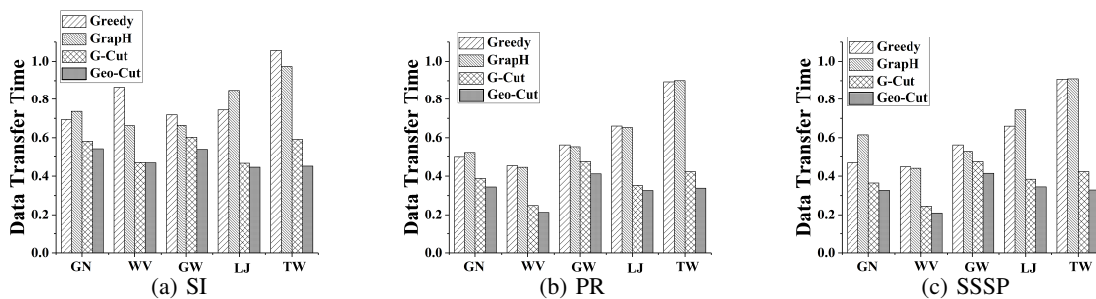


Fig. 5: Normalized inter-DC data transfer time of Amazon EC2.

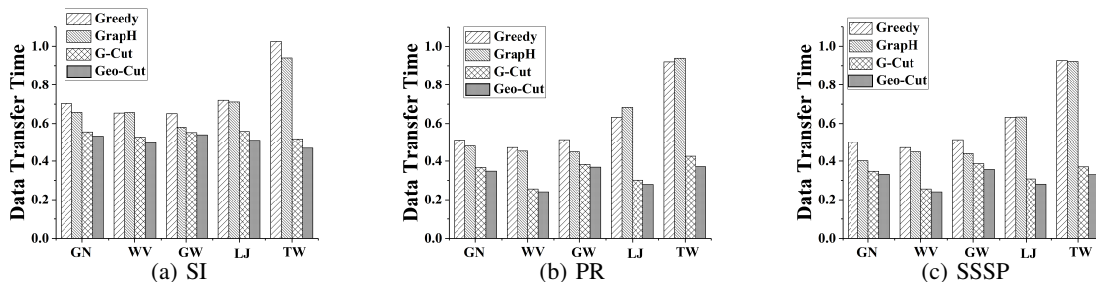


Fig. 6: Normalized inter-DC data transfer time of Windows Azure.

on reducing the inter-DC data transfer time on Amazon EC2 than that on Window Azure. Specifically, on Amazon EC2, Placement reduces the inter-DC data transfer time by 27%-35% compared to Streaming, while on Azure, the improvement obtained by partition mapping is not obvious. This is mainly because the network bandwidth heterogeneity of Amazon EC2 DCs is much higher than that of Windows Azure DCs. Specifically, the highest download/upload bandwidth is 5x/2x as high as the lowest download/upload bandwidth on Amazon EC2 and is only 1.79x/1.42x on Windows Azure. Higher bandwidth heterogeneity offers better opportunity for partition mapping to further reduce the inter-DC data transfer time. Second, comparing the results obtained by Geo-Cut and Placement, the edge migration technique is also more effective on reducing the inter-DC data transfer time on Amazon EC2 than that on Windows Azure. Specifically, on Amazon EC2, Geo-Cut reduces the inter-DC data transfer time by 13%-16% compared to Placement, while on Windows Azure, the time reduction is only 2%-6%. This is mainly due to the fact that the difference between network bandwidth and price heterogeneities on Amazon EC2 is much larger than that on Windows Azure. On Amazon EC2, the highest network price is 8x as high as the lowest network price and is only 1.38x on Windows Azure. As Streaming is only aware of the price heterogeneity when assigning edges to DCs, it is more likely to cause performance bottlenecks on Amazon EC2 DCs than on Windows Azure, where the edge migration technique can be made use of. These observations demonstrate that the techniques designed in Geo-Cut work well for geo-distributed environments with significant network bandwidth and price heterogeneities.

### 6.2.2 Inter-DC Data Transfer Cost

Figures 7 and 8 show the monetary costs spent on inter-DC data communication optimized by the compared algorithms. All results are normalized to the budget.

TABLE 9: Vertex replication rates optimized by compared algorithms for PageRank on Windows Azure.

	Baseline	Greedy	GraphH	G-Cut	Geo-Cut
Gnutella	2.60	1.91	1.94	1.81	1.85
WikiVote	2.68	2.02	2.06	1.97	2.00
GoogleWeb	1.44	1.26	1.26	1.12	1.15
LiveJournal	3.28	2.25	2.26	1.72	1.78
Twitter	2.36	1.49	1.62	1.29	1.34

Overall, Geo-Cut obtains the lowest inter-DC data transfer cost on both clouds and is able to satisfy the budget constraint under all settings. Specifically, Geo-Cut reduces the data communication cost by 36%-70%, 14%-53%, 1%-50% and 2%-18% compared to Baseline, Greedy, GraphH and G-Cut on Amazon EC2 and 45%-75%, 18%-55%, 6%-48% and 4%-15% compared to Baseline, Greedy, GraphH and G-Cut on Windows Azure, respectively. The reason that Geo-Cut obtains lower data communication cost lies in two aspects.

First, Geo-Cut leads to less WAN usage during graph processing compared to other partitioning methods including Baseline, Greedy and GraphH, as shown in Figures 10 and 11. SSSP obtains similar results as those of PageRank and thus is not shown in the figures. Our streaming heuristic in the streaming graph partitioning technique prefers to assign an edge to the partition which does not require vertex replication. As a result, Geo-Cut can achieve a low vertex replication rate. For example, we show the vertex replication rates optimized by the compared partitioning algorithms for PageRank on Windows Azure in Table 9. As shown in the table, vertex replication rate obtained by Geo-Cut is lower than those obtained by Baseline, Greedy and GraphH, and is comparable to that of G-Cut. This is because G-Cut is also designed for geo-distributed clouds with WAN usage as one optimization factor. Lower vertex replication rate leads to less inter-DC communications between vertex replicas and potentially lower inter-DC networking cost.

Second, in addition to reducing WAN usage, Geo-Cut can smartly distribute more data communication to DCs with lower

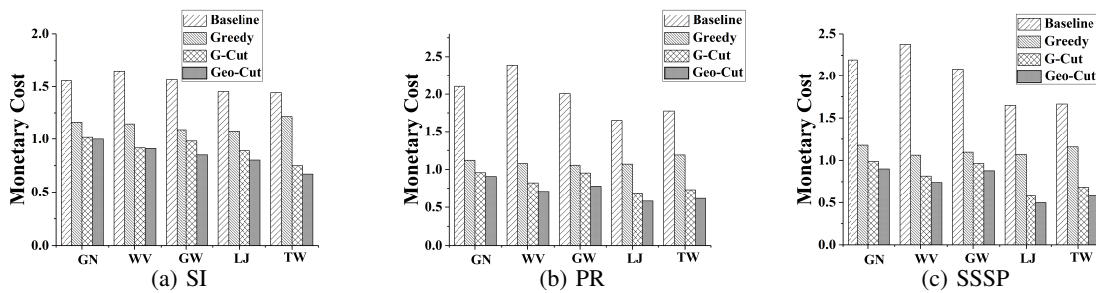


Fig. 7: Normalized inter-DC data transfer cost of Amazon EC2.

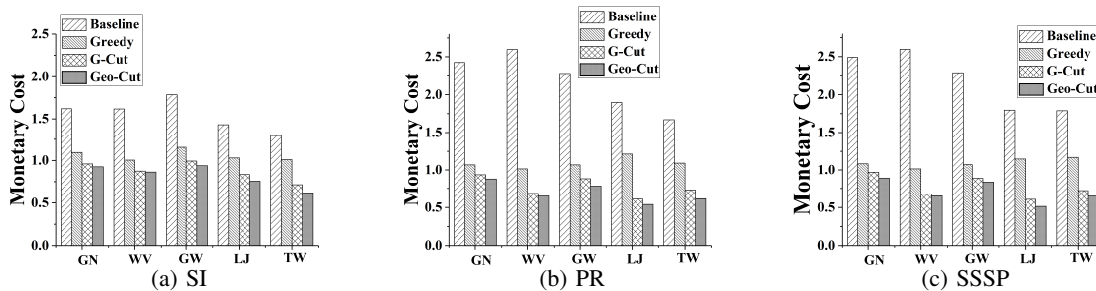


Fig. 8: Normalized inter-DC data transfer cost of Windows Azure.

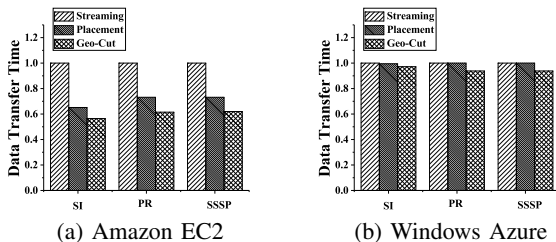


Fig. 9: Normalized data transfer time optimized by traffic-aware graph partitioning only (Streaming), both Streaming and partition mapping (Placement) and Geo-Cut using WikiVote.

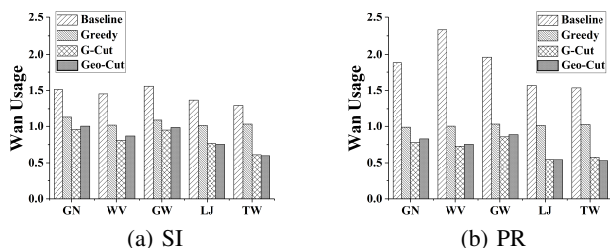


Fig. 10: Normalized WAN usage obtained on Amazon EC2.

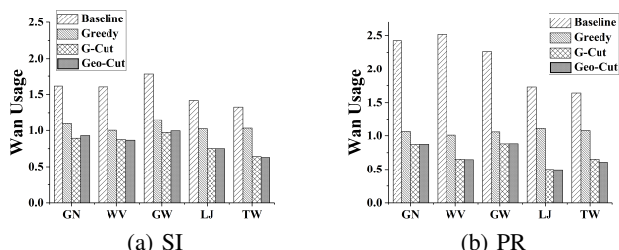


Fig. 11: Normalized WAN usage obtained on Windows Azure.

TABLE 10: Uploading data distribution on Amazon EC2.

	USE	USW-O	USW-NC	EU	SIN	TKY	SYD	SA
price(\$)	0.02	0.02	0.02	0.02	0.09	0.09	0.14	0.16
Greedy(%)	12	13	12	13	12	13	13	12
Graph(%)	12	15	12	14	11	13	12	11
G-Cut(%)	11	11	13	13	15	12	13	12
Geo-Cut(%)	11	15	16	15	13	9	10	11

inter-DC network prices. For example, Table 10 shows the uploading data distribution in different DCs of Amazon EC2 optimized by the compared partitioning methods using WikiVote graph and SI algorithm. Overall, Geo-Cut distributes 57% of the total uploading data to the cheapest four DCs while Greedy is more load-balanced with 50% of the total data distributed to the cheapest four DCs. G-Cut distributes 48% of the total uploading data to the cheapest four DCs due to its ignorance of the network price heterogeneity. Graph is also aware of the price heterogeneity and distributes 53% of the total data to the cheapest four DCs. However, due to its larger WAN usage during graph processing, Graph still results in higher data communication cost compared to Geo-Cut.

### 6.2.3 Trade-off Between Performance and Cost

It is interesting to also study the trade-off between the performance and cost of graph processing in geo-distributed DCs. We present the inter-DC data transfer time and cost results obtained by the compared partitioning methods using Twitter graph and Subgraph Isomorphism algorithm in a skyline evaluation manner as shown in Figure 12. All time results are normalized to that of Baseline and the cost results are normalized to the budget. We have the following observations.

First, there is no obvious relationship between the performance and cost optimization goals. For example, on Amazon EC2, the partitioning results of Graph and Greedy lead to almost the same inter-DC data transfer time while the data transfer cost obtained by Graph is much lower. On the other hand, the results on Windows Azure show that Graph and Greedy obtain almost the same inter-DC data transfer cost while the data transfer time of Graph is slightly lower than

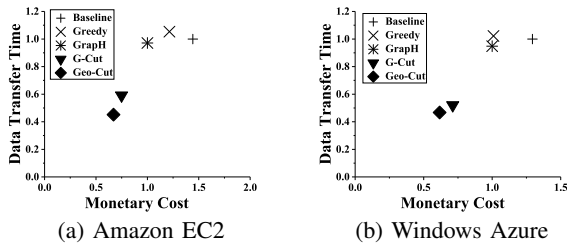


Fig. 12: Trade-off between inter-DC data transfer time and cost using Twitter graph and SI algorithm.

that of Greedy. This shows that the complicated bandwidth and price heterogeneities in geo-distributed DCs make it hard to optimize the performance and cost goals at the same time.

Second, on both clouds, Geo-Cut is able to obtain better result when comparing both inter-DC data transfer time and monetary cost to the other methods. This is consistent with our observations in the previous two subsections where Geo-Cut is able to obtain both lower inter-DC data transfer time and lower cost than the other methods at the same time. This shows that Geo-Cut is able to obtain performance- and cost-efficient graph partitioning result in geo-distributed DCs.

#### 6.2.4 Graph Partitioning Overhead

We evaluate the partitioning overhead of Geo-Cut using PageRank algorithm. As shown in Figure 13, we breakdown the overhead into three parts, namely the overhead spent on streaming partitioning, the overhead on partition mapping and the overhead on edge migration. We have several observations.

First, the overhead of Geo-Cut is much higher than that of Greedy while is comparable to that of GrapH. For example, the partitioning overhead for the Livejournal graph is 105s with Greedy, 1720s with GrapH and 1020s with Geo-Cut. This is mainly because the heterogeneity-aware partitioning techniques of Geo-Cut are more complicated than Greedy.

Second, streaming partitioning takes the largest portion of the entire overhead of Geo-Cut, and the ratio increases as the graph size increases. For example, for GoogleWeb, the streaming partitioning overhead is only 57% of the entire overhead while for Livejournal, the ratio is over 90%. Overall, to partition one million edges, it takes Geo-Cut less than three seconds for small size graphs such as GoogleWeb and around 100 seconds for large size graphs such as Twitter.

Third, partition mapping and edge migration techniques have comparable overhead. For edge migration, the  $L_Q$  parameter, which decides the number of edges to be migrated, has significant impact on the trade-off between migration effectiveness and efficiency. We perform a sensitivity study over this parameter by varying  $L_Q$  from 0%, 5%, 10%, 15% to 20% of the number of vertices in  $Q$ . Figure 14 shows the resulted edge migration overhead and normalized inter-DC data transfer time on Amazon EC2 using Livejournal graph. As  $L_Q$  increases, the effectiveness of edge migration increases (i.e., the inter-DC data transfer time decreases) while the efficiency decreases (i.e., the migration overhead increases almost linearly). When  $L_Q$  is larger than 10%, the inter-DC data transfer time becomes stable. By default, we set  $L_Q$  to 5% to get a good balance between the effectiveness and efficiency of edge migration.

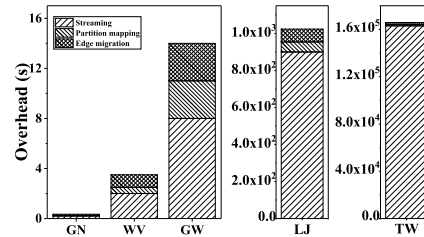


Fig. 13: Graph partitioning overhead breakdown of Geo-Cut.

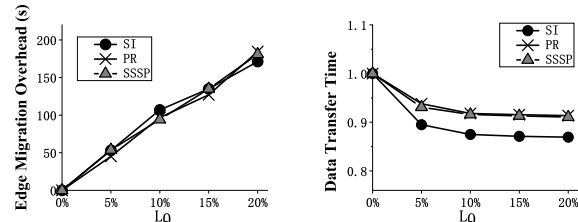


Fig. 14: Sensitivity study of  $L_Q$  on Amazon EC2 using LJ.

### 6.3 Simulation Results

We present the simulation results using real cloud traces to evaluate the sensitivity of Geo-Cut to different parameters, including network heterogeneities in bandwidth and prices, budget constraint and dynamic graph changes.

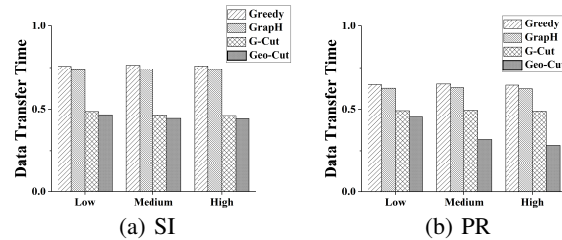


Fig. 15: Normalized inter-DC data transfer time under different bandwidth heterogeneities using Livejournal.

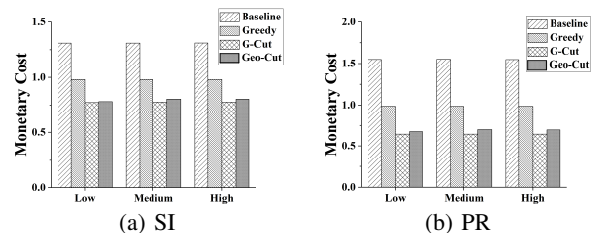


Fig. 16: Normalized inter-DC data transfer cost under different bandwidth heterogeneities using Livejournal.

#### 6.3.1 Network Heterogeneity

**Bandwidth heterogeneity.** Figure 15 and Figure 16 show the normalized inter-DC data transfer time and cost optimized by the compared algorithms under different network bandwidth heterogeneities, respectively. The time results are normalized to those of Baseline and the cost results are normalized to those of GrapH. SSSP obtains similar results as those of PageRank and thus is not shown in the figures.

Geo-Cut obtains the lowest inter-DC data transfer time among the compared methods under all settings. Specifically, Geo-Cut reduces the data transfer time over the compared methods by 4%-55%, 5%-68% and 4%-72% for the Low, Medium and High bandwidth heterogeneity environments,

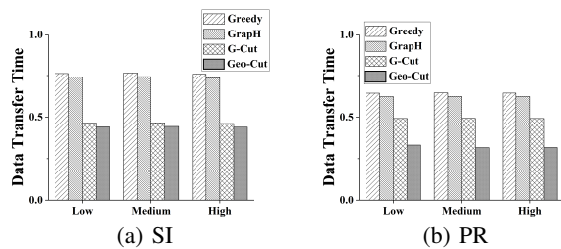


Fig. 17: Normalized inter-DC data transfer time under different price heterogeneities using Livejournal.

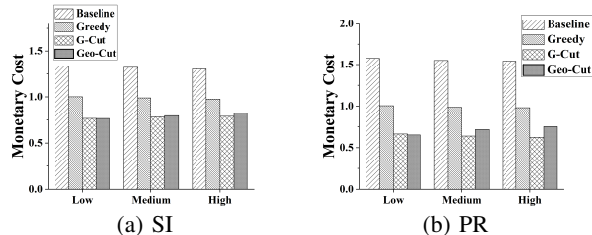


Fig. 18: Normalized inter-DC data transfer cost under different price heterogeneities using Livejournal.

respectively. Benefiting from its bandwidth heterogeneity awareness, Geo-Cut obtains higher time reduction over Baseline, Greedy and GraphH in environments with more heterogeneous network bandwidths. This trend does not apply when comparing Geo-Cut with G-Cut, as G-Cut is also designed to be bandwidth heterogeneity aware.

Similarly, Geo-Cut obtains the lowest inter-DC data transfer cost among all compared methods (except G-Cut) under all settings. The monetary cost obtained by Geo-Cut increases slightly with the increase of network bandwidth heterogeneity. This is because, with the increase of network bandwidth heterogeneity, the partition mapping and edge migration techniques have better opportunity of further reducing the inter-DC data transfer time over the results of Streaming, while sacrificing the monetary cost optimized by Streaming. For example, when the heterogeneity changes from Low to High, the two techniques obtain 44%-54% more time reduction over Streaming while the additional monetary cost compared to the cost optimized by Streaming increases by 8%-13%. This shows that Geo-Cut can effectively address the network bandwidth heterogeneity problem to improve the performance of geodistributed graph processing.

To verify these observations, we also implement the bandwidth heterogeneity experiments with PageRank and Twitter, which is the largest graph in our paper. Specifically, Geo-Cut reduces the the data transfer time over the compared methods by 3%-73%, 2%-77% and 3%-81% for the Low, Medium and High bandwidth heterogeneity environments, respectively. The normalized monetary cost of Geo-Cut slightly increases from 80% to 85% when the bandwidth heterogeneity changes from Low to High. These results are consistent with those obtained using Livejournal. Because of the high experimental overhead of Twitter, we use Livejournal only for the rest of the simulations.

**Price heterogeneity.** Figure 17 and Figure 18 show the normalized inter-DC data transfer time and cost optimized by the compared algorithms under different network price heterogeneities, respectively. The time results are normalized to

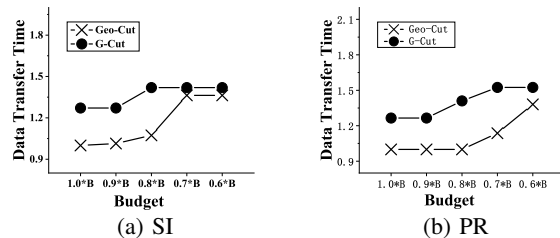


Fig. 19: Normalized inter-DC data transfer time under different budget constraints.

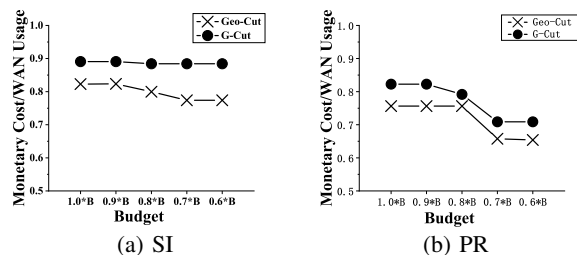


Fig. 20: Monetary cost/WAN usage under different budget constraints.

those of Baseline and the cost results are normalized to those of GraphH. SSSP obtains similar results as those of PageRank and thus is not shown in the figures.

Similar to the bandwidth heterogeneity study, Geo-Cut obtains the lowest inter-DC data transfer time among the compared methods under all settings. Differently, the time reduction obtained by Geo-Cut compared to the other methods does not change much with the increase of price heterogeneity. For example, Geo-Cut reduces the inter-DC data transfer time by 5%-67%, 5%-68% and 5%-68% for Low, Medium and High network price heterogeneities, respectively. This is mainly because the three-stage design of Geo-Cut can guarantee good inter-DC data transfer time for different price heterogeneities. The price heterogeneity only affects the optimization result of the streaming partitioning stage, which tends to assign edges more balanced to different DCs when the price heterogeneity is low and assign more edges to less pricy DCs when the price heterogeneity is high. In the former case, the edge migration method can be used to adapt to the network bandwidth heterogeneity while in the latter case, the partition mapping technique can be used to reduce the inter-DC data transfer time. The monetary cost obtained by Geo-Cut increases slightly with the increase of network bandwidth heterogeneity. This is because, with the increase of network price heterogeneity, the partition mapping and edge migration techniques have to sacrifice more monetary cost optimized by Streaming, in order to further reducing the inter-DC data transfer time over the results of Streaming.

### 6.3.2 Budget Constraint

Figure 19 shows the inter-DC data transfer time obtained by Geo-Cut and G-Cut under different budget constraints, where  $B$  represents the default budget. All results are normalized to those of Geo-Cut under  $B$ . Figure 20 shows the monetary cost obtained by Geo-Cut and the WAN usage obtained by G-Cut under different budget constraints. All results are normalized to their own budget. We have the following observations.

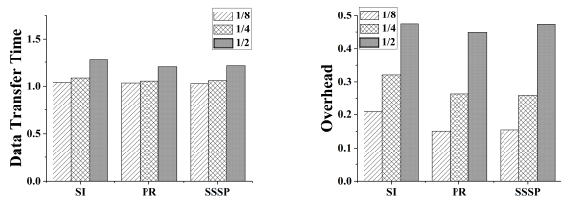


Fig. 21: Dynamic edge insertion study using Livejournal.

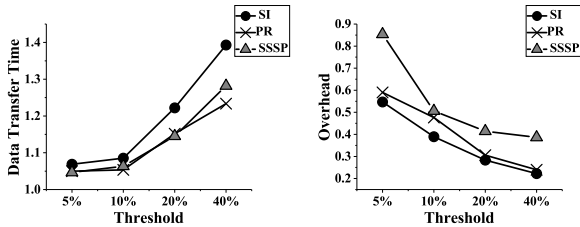


Fig. 22: Sensitivity study on the threshold to trigger partition refinement.

First, when the budget is loose (e.g., from  $1.0 * B$  to  $0.9 * B$ ), the inter-DC data transfer time obtained by both algorithms increase slowly with the decrease of budget. After the budget decreases to a certain value (e.g.,  $0.8 * B$  for Geo-Cut and  $0.9 * B$  for G-Cut), the inter-DC data transfer time increases suddenly. For example, as shown in Figure 19a, when the budget for Geo-Cut decreases from  $0.8 * B$  to  $0.7 * B$ , the time increases by 21%, while the time increases only by 5% when the budget changes from  $0.9 * B$  to  $0.8 * B$ . This is because, after the budget gets tighter than the certain value, both algorithms have to sacrifice some time optimization to satisfy the budget constraint. In Figure 20a, we can see that when the budget is  $0.7 * B$ , the cost obtained by Geo-Cut has already exceeded the budget, which means the budget is too tight to be satisfied. Second, comparing Geo-Cut with G-cut, Geo-Cut obtains better inter-DC data transfer time than G-Cut under all budget settings. Further, Geo-Cut can satisfy tighter budget than G-Cut. For example, as shown in Figure 20b, the tightest budget that Geo-Cut can satisfy is  $0.7 * B$  while the tightest budget of G-Cut is only  $0.8 * B$ . This is mainly because Geo-Cut is aware of both network bandwidth and price heterogeneities and thus can better optimize the inter-DC data transfer time and cost at the same time.

### 6.3.3 Dynamic Graph Refinement

**Threshold to trigger partition refinement.** Figure 22 shows the inter-DC data transfer time and overhead obtained by our dynamic refinement method when inserting 1/4 of the edges to part of the Livejournal graph, using different thresholds to trigger partition refinement. All results are normalized to those of repartitioning the complete graph. We have the following observations.

First, when the threshold is set to 5%, both the obtained data transfer time and the overhead of our dynamic partitioning method are very close to those of repartitioning the complete graph. This is because when the threshold is low, the partition refinement happens more often and hence leads to better data transfer time and high optimization overhead. Second, when the threshold increases from 5% to 40%, the obtained data transfer time increases while the overhead decreases. For ex-

ample, the normalized data transfer time of SSSP increases from 1.05 to 1.28 while the normalized overhead decreases from 0.85 to 0.39. Overall, we set the threshold to 10% of the overall data traffic size of a partition, which can obtain a good data transfer time with acceptable overhead.

**Edge insertion and deletion.** Figure 21 shows the inter-DC data transfer time and overhead results obtained by our dynamic refinement method when inserting edges to part of the Livejournal graph. All results are normalized to those of repartitioning the complete graph.

We have the following observations. First, with small number of inserted edges, the dynamic partition refinement method obtains similar inter-DC transfer time compared to repartitioning the entire graph. Specifically, the inter-DC data transfer time obtained by the partition refinement method is 3%-4% higher than that of repartitioning the entire graph when inserting 1/8 of total edges. On the other hand, the partition refinement method consumes much lower overhead compared to repartitioning, where the overhead of partition refinement is 78%-86% lower than that of repartitioning when inserting 1/8 of total edges. This observation demonstrates that our dynamic refinement method is effective when the change of graph is not huge. Second, when the number of inserted edges increases, as can be expected, the inter-DC data transfer time obtained by the partition refinement method becomes higher than that of repartitioning, although the optimization overhead of partition refinement is still lower than that of repartitioning. For example, the inter-DC data transfer time obtained by refinement is 5%-8% higher than that of repartitioning when inserting 1/4 of the edges and is 20%-27% higher when inserting 1/2 of the edges. The overhead of partition refinement is 68%-74% and 56%-61% lower than that of repartitioning when inserting 1/4 and 1/2 of the edges, respectively. This observation suggests that, when the change of graph is significant, it is better to repartition the graph entirely than using the dynamic partition refinement method. Experiments on edge deletion have shown similar results to those of edge insertion and thus are not presented in the paper.

**Changing number of partitions.** We compare the inter-DC transfer time and overhead obtained by dynamic partitioning to that of the repartitioning method. The inter-DC data transfer time optimized by dynamic partition refinement is quite close to that of repartitioning, with 4%-9% increase, while the optimization overhead of partition refinement is 67%-71% lower than that of the repartitioning method. As can be expected, the results of reducing 1/4 of the partitions are very similar to those of inserting 1/4 of the edges.

The above results show that our partition refinement technique can efficiently obtain good partitioning results for different types of dynamic graph changes.

## 6.4 Summary of Findings

On observing the above evaluation results obtained from real clouds and simulations, we have the following conclusions.

First, the multi-level heterogeneities in network bandwidth and communication prices in geo-distributed DCs can greatly impact the cost and performance efficiency of graph processing jobs. Compared to existing graph partitioning meth-



ods which are not aware of the complicated network heterogeneities, Geo-Cut can reduce the inter-DC data transfer time by up to 79% and reduce the monetary cost by 70%.

Second, Geo-Cut is highly efficient thanks to its two-stage optimizations and streaming-based edge assignment design. It can partition one million edges for large size graphs such as Twitter in less than 100 seconds. As a result, Geo-Cut has also shown good results when partitioning dynamic graphs.

## 7 CONCLUSION

Recently, many graph applications are deployed in geo-distributed DCs to provide services with low latency. This raises new challenges to existing graph partitioning methods, due to the multi-level heterogeneities in network bandwidth and communication prices in geo-distributed DCs. In this paper, we propose a geo-aware graph partitioning method named *Geo-Cut*, which aims at minimizing the inter-DC data transfer time of graph processing while satisfying the budget constraint on inter-DC networking cost. Geo-Cut adopts two optimization stages to reduce the optimization overhead including an efficient one-pass streaming graph partitioning and graph partitioning refinement by considering the network bandwidth and price heterogeneities. Geo-Cut can be also applied to partition dynamic graphs thanks to its lightweight runtime overhead. Our evaluation with real-world graphs in both real geo-distributed DCs and simulations show that Geo-Cut can reduce the inter-DC data transfer time by up to 79% (42% as the median value) and reduce the monetary cost by up to 75% (26% as the median value) compared to state-of-the-art graph partitioning methods with a low overhead. As future work, we plan to extend our techniques to other graph processing models and experiment on heterogeneous computing environments with GPUs [37].

## ACKNOWLEDGMENT

The corresponding author is Shadi Ibrahim (shadi.ibrahim@inria.fr). This work is in part supported by the National Natural Science Foundation of China (No. 61802260), the Guangdong Natural Science Foundation (No. 2018A030310440 and 2019A1515012053), the Shenzhen Science and Technology Foundation (No. JCYJ20180305125737520), the Natural Science Foundation of SZU (No. 827-000370, 827-000175 and 860-000002110319), the Guangdong Province Key Laboratory of Popular High Performance Computers (No. 2017B030314073), the ANR KerStream project (ANR-16-CE25-0014-01), and the Stack/Apollo connect talent project. Bingsheng He's research is partly supported by a MoE AcRF Tier 1 grant (T1 251RES1610) in Singapore and National Natural Science Foundation of China (No. 61929103).

## REFERENCES

- [1] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *2015 International Conference on Very Large Data Bases*, vol. 8, no. 12, pp. 1804–1815, 2015.
- [2] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *2013 ACM International Conference on Web Search and Data Mining*, 2013, pp. 507–516.
- [3] L. Zhu, A. Galstyan, J. Cheng, and K. Lerman, "Tripartite graph clustering for dynamic sentiment analysis on social media," in *2014 ACM Conference on Management of Data*, 2014, pp. 1531–1542.
- [4] E. Minkov, W. W. Cohen, and A. Y. Ng, "Contextual search and name disambiguation in email using graphs," in *2006 International Conference on Research on Development in Information Retrieval*, 2006, pp. 27–34.
- [5] 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 *2010 ACM Conference on Management of Data*, 2010, pp. 135–146.
- [6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *2012 International Conference on Very Large Data Bases*, vol. 5, no. 8, pp. 716–727, 2012.
- [7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *2012 USENIX Symposium on Operating Systems Design and Implementations*, 2012, pp. 17–30.
- [8] Scaling the Facebook data warehouse to 300 PB, <https://goo.gl/Eyv6o3>.
- [9] "The court of justice declares that the commissions us safe harbour decision is invalid," <https://goo.gl/vLg6aw>, 2015.
- [10] C. Mayer, M. A. Tariq, C. Li, and K. Rothermel, "GrapH: Heterogeneity-aware graph computation with adaptive partitioning," in *2016 International Conference on Distributed Computing Systems*, 2016, pp. 118–128.
- [11] A. C. Zhou, Y. Xiao, Y. Gong, B. He, J. Zhai, and R. Mao, "Privacy regulation aware process mapping in geo-distributed cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1872–1888, Aug 2019.
- [12] A. C. Zhou, Y. Gong, B. He, and J. Zhai, "Efficient process mapping in geo-distributed cloud data centers," in *2017 International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2017, pp. 16:1–16:12.
- [13] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *2015 ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2015, pp. 421–434.
- [14] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *2010 Proceedings of the International Conference on Very Large Data Bases Endowment*, vol. 3, no. 1-2, pp. 460–471, 2010.
- [15] AWS Direct Connect Pricing, <https://aws.amazon.com/directconnect/pricing/>, accessed Nov 2016.
- [16] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *2016 ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2016, pp. 73–86.
- [17] 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 *2014 USENIX Symposium on Operating Systems Design and Implementations*, 2014, pp. 599–613.
- [18] M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph partitioning for distributed graph processing," *Data Science and Engineering*, vol. 2, no. 1, pp. 94–105, 2017.
- [19] T. Chen and B. Li, "A distributed graph partitioning algorithm for processing large graphs," in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, March 2016, pp. 53–59.
- [20] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "A distributed algorithm for large-scale graph partitioning," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 10, no. 2, pp. 12:1–12:24, Jun. 2015.
- [21] METIS, <http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>.
- [22] Q. Hua, Y. Li, D. Yu, and H. Jin, "Quasi-streaming graph partitioning: A game theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2019.
- [23] N. Xu, B. Cui, L. Chen, Z. Huang, and Y. Shao, "Heterogeneous environment aware streaming graph partitioning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 6, pp. 1560–1572, 2015.
- [24] C. Mayer, R. Mayer, M. A. Tariq, H. Geppert, L. Laich, L. Rieger, and K. Rothermel, "Adwise: Adaptive window-based streaming edge partitioning for high-speed graph processing," in *2018 International Conference on Distributed Computing Systems*, 2018, pp. 685–695.
- [25] A. C. Zhou, S. Ibrahim, and B. He, "On achieving efficient data transfer for graph processing in geo-distributed datacenters," in *2017 International Conference on Distributed Computing Systems*, 2017, pp. 1397–1407.

- [26] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: Experience with a globally-deployed software defined wan,” in *2013 ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2013, pp. 3–14.
- [27] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” in *1998 International World Wide Web Conferences*, 1998, pp. 107–117.
- [28] D. P. Bertsekas, F. Guerriero, and R. Musmanno, “Parallel asynchronous label-correcting methods for shortest paths,” *Journal of Optimization Theory and Applications*, vol. 88, no. 2, pp. 297–320, 1996.
- [29] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, “Capturing topology in graph pattern matching,” *2011 International Conference on Very Large Data Bases*, vol. 5, no. 4, pp. 310–321, 2011.
- [30] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, “Hdrf: Stream-based partitioning for power-law graphs,” in *2015 ACM International Conference on Information and Knowledge Management*, 2015, pp. 243–252.
- [31] I. Stanton and G. Kliot, “Streaming graph partitioning for large distributed graphs,” in *2012 ACM Knowledge Discovery and Data Mining*, 2012, pp. 1222–1230.
- [32] M. Mitzenmacher, “The power of two choices in randomized load balancing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.
- [33] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, “The power of choice in data-aware cluster scheduling,” in *2014 USENIX Symposium on Operating Systems Design and Implementations*, 2014, pp. 301–316.
- [34] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, “Hopper: Decentralized speculation-aware cluster scheduling at scale,” in *2015 ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2015, pp. 379–392.
- [35] J. Huang and D. J. Abadi, “Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs,” *2016 International Conference on Very Large Data Bases*, vol. 9, no. 7, pp. 540–551, 2016.
- [36] Stanford Large Network Dataset Collection, <https://snap.stanford.edu/data/>, accessed on Oct 2016.
- [37] J. Zhong and B. He, “Medusa: Simplified graph processing on gpus,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1543–1552, 2014.



**Amelie Chi Zhou** is currently an Assistant Professor in Shenzhen University, China. Prior to that, she was a Postdoc Fellow in Inria-Bretagne research center, France. She received her PhD degree in 2016 from School of Computer Engineering, Nanyang Technological University, Singapore. Her research interests lie in cloud computing, big data processing, distributed systems and resource management.



**Bingkun Shen** received his bachelor's degree from Nanjing Tech University in 2014. He is currently a postgraduate student in College of Computer Science and Software Engineering of Shenzhen University and his research mainly focuses on cloud computing.



**Yao Xiao** received his bachelor degree in School of Automation from Huazhong University of Science and Technology (2008-2012). He currently is a postgraduate student in College of Computer Science and Software Engineering of Shenzhen University. His research interest includes cloud computing, big data processing and resource management.



**Shadi Ibrahim** is a permanent Inria Research Scientist. He obtained his Ph.D. in Computer Science from Huazhong University of Science and Technology in 2011. His research interests are in cloud computing, big data management, virtualization technology, and file and storage systems. He has published several research papers in recognized Big Data and cloud computing research conferences and journals such as TPDS, FGCS, SC, IPDPS and ICPP.



**Bingsheng He** is currently an Associate Professor at Department of Computer Science, National University of Singapore. Before joining NUS in 2016, he held a research position in the System Research group of MSRA (2008-2010) and a faculty position in Nanyang Technological University, Singapore. He got the Ph.D. degree in Hong Kong University of Science & Technology in 2008. His research interests include big data management, parallel and distributed systems and cloud computing.