A Distributed Graph-Parallel Computing System with Lightweight Communication Overhead

Yue Zhao, Kenji Yoshigoe, *Senior Member, IEEE*, Jiang Bian, *Member, IEEE*, Mengjun Xie, *Member, IEEE*, Zhe Xue, and Yong Feng

Abstract—In order to process complex and large-scale graph data numerous distributed graph-parallel computing platforms have been proposed. However, excessive communications among computing nodes in these systems not only aggravate the network I/O workload of the underlying computing hardware systems but may also cause a decrease in runtime performance and scalability. In this paper, we propose and implement a system called Ligraph, which computes large-scale graph data in distributed mode with lightweight communication overhead. Ligraph is similar to PowerGraph system with three new features: (1) a Gather partial sum difference based computing model; (2) a corresponding lightweight Gather communication mechanism; (3) for PageRank-like algorithms Ligraph additionally employs a lightweight synchronizing communication mechanism and an edge direction-aware graph partition strategy proposed by our former work LightGraph, which is specially designed for PageRank-like algorithms. We have conducted extensive experiments using real-world data sets, and our results verified the effectiveness of Ligraph on reducing the communication overhead and improving the runtime performance and the scalability compared with PowerGraph and LightGraph. For example, compared with PowerGraph under Random partition scenario Ligraph can not only reduce up to 35.2 percent of the communication overhead but also cut up to 21.8 percent of the runtime for PageRank algorithm while processing Twitter data set. Our experiment results also demonstrate that compared with several other representative existing systems Ligraph also outperforms them in graph computing rate.

Index Terms—Distributed graph-parallel computing, big data, communication overhead

1 Introduction

Big graph analysis and computing is an emerging and significant topic in both industry and academic research. The reason is that complex networks such as social, biological and computer networks can be mathematically modeled as graphs. And these real-world networks are often very large in size, consisting of millions or even billions of vertices, and a much larger number of edges. Efficient and fast processing of these large real-world networks is a basic requirement of engineers and scientists. Thus, more and more attention and effort have been attracted to the work of designing effective and scalable computing systems to analyze and process the huge real-world graphs.

Numerous graph-parallel computing abstractions have been proposed and applied in real-world applications. For example the single machine graph-parallel computing systems [1], [2], [3], [4], [5], [6], [7], [8], which process large-scale graph under the shared-memory multiprocessor computing environment, and the distributed graph-parallel computing systems [9], [10], [11], [12], [13], [14], [15], [16], [17], which process large-scale graph in parallel using multiple machines. Compared to single machine graph-processing systems, distributed graph-parallel computing systems have higher scalability and greater computational and storage resources for handling larger networks. However, in distributed graph-parallel computing systems the communications among subtasks allocated on different machines constitute significant obstacle to achieve good parallel program performance. The communication overhead burdens the I/O system of the underlying cloud/cluster platform and potentially impacts the graph computing efficiency and the scalability of the computing system. Fig. 1 illustrates the scalability of several representative existing distributed graph-parallel computing systems while executing PageRank algorithm on Twitter data set [18] (the underlying system environment see Section 6.1). As the figure shows all the evaluated systems exhibit a poor speedup with the number of machines used increasing.

PowerGraph is one of the most advanced and popular representative of the distributed graph-parallel computing systems. PowerGraph adopts the mechanism of vertex-program and exposes substantially greater parallelism [16], which can quickly partition a graph, especially a power-law graph, and improve the graph computing rate significantly. However, like other distributed graph-parallel computing systems, the effectiveness of PowerGraph also suffers its heavy communications overhead. In this paper, we propose and implement a new distributed graph-parallel computing system with lightweight communication overhead, Ligraph. Like PowerGraph, Ligraph is applicable for common large-scale graph-structured computation. Ligraph is similar to

- *Y. Zhao is with the National Heart Lung and Blood Institute (NHLBI), National Institutes of Health (NIH), Bethesda, MD. E-mail: yue.zhao@nih.gov.*
- *K. Yoshigoe and M. Xie are with the Department of Computer Science, University of Arkansas at Little Rock, Little Rock, AR. E-mail: {kxyoshigoe, mxxie}@ualr.edu.*
- *J. Bian is with the Department of Health Outcomes and Policy, University of Florida, Gainesville, FL. E-mail: bianjiang@ufl.edu.*
- *Z. Xue is with the School of Preclinical Medicine, Beijing University of Chinese Medicine, Beijing, China. E-mail: xuezhesanctity@163.com.*
- *Y. Feng is with the School of Information, Liaoning University, Shenyang, Liaoning, China. E-mail: fengyong@lnu.edu.cn.*

Manuscript received 8 Sept. 2015; revised 29 Jan. 2016; accepted 12 Feb. 2016. Date of publication 28 Feb. 2016; date of current version 28 Oct. 2016. Recommended for acceptance by M. Sheng, A.V. Vasilakos, Q. Yu, and L. Yao. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TBDATA.2016.2532907

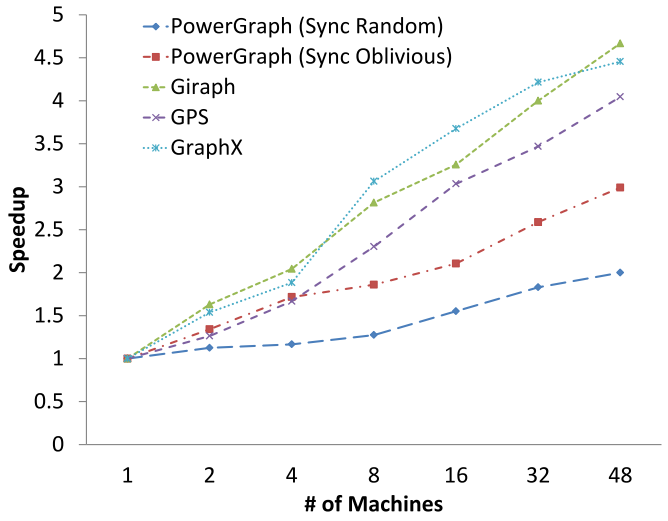


Fig. 1. The PageRank runtime speedup of existing systems while computing the Twitter data set [18].

PowerGraph with the following new features, which are our key contributions of this work:

a. A Gather partial sum difference based computing model;
b. A corresponding lightweight Gather communication mechanism;
c. For PageRank-like algorithms Ligraph additionally employs a lightweight synchronizing communication mechanism and an edge direction-aware graph partition strategy (EDAP), which optimally isolates the outgoing edges from the incoming edges of a vertex.

The rest of the paper is organized as follows. Section 2 introduces the background of this work. The Gather partial sum difference based computing model is detailed in Section 3. Section 4 introduces the lightweight Gather communication mechanism. We conducted the volume of communications analysis in Section 5. The design and results of the experiment are detailed in Section 6. Section 7 introduces the related work. Section 8 concludes this paper and states the direction for future work.

2 Background

In a graph-parallel abstraction, the data to process is presented as a sparse graph in memory, $G = \{V, E\}$, and the computation is conducted by executing a vertex-program Q in parallel on each vertex, $v \in V$. With the computing resource turning to be more and more abundant and available people resort to distributed platforms to conduct graph-parallel computing. In the following part of this section we mainly give a detailed introduction of two of them: PowerGraph and LightGraph, which are most relevant to this work.

Pregel [10] is a bulk synchronous graph computing abstraction. It uses message passing mode to transfer information between super-steps. Pregel proposes a commutative associative message combiner to reduce the number of messages. GraphLab [2], [15] is a framework, which supports asynchronous parallel graph computations in machine learning. It differs from Pregel in that it allows the vertex programs to run asynchronously based on a scheduler. PowerGraph [16] is drawn from the distributed GraphLab framework [15]. It introduces several significant improvements to the distributed GraphLab framework. First, PowerGraph proposes a GAS programming model, in which a vertex program consists of three phases: Gather, Apply and Scatter. By partitioning the original vertex program into sub-phases and lifting these sub-phases into the abstraction PowerGraph distributes the execution of a single vertex-program over the entire system [16]. Second, PowerGraph inherits and incorporates many significant advantages of both Pregel [10] and GraphLab [2], [15]. For example, from GraphLab PowerGraph inherits the shared-memory and data-graph view of computation, which frees users from architecting a communication protocol to share information. Like distributed Graphlab, PowerGraph supports both bulk-synchronous and asynchronous computation model. And in order to reduce the communication overhead in Gather phase PowerGraph borrows the commutative associative message combiner from Pregel. At last, PowerGraph employs the vertex-cut approach for graph partition. Vertex-cut can quickly comminute a large power-law graph by cutting a small fraction of very high-degree vertices. Thus, it addresses the problem of partitioning the power-law graphs [16]. The most important partition strategies used in PowerGraph are Random and Oblivious [16]. Random strategy employs a hash function to randomly distribute edges to computing nodes. It is fully data-parallel during the partitioning process and can achieve a near-perfect balance in workload distribution on large graphs. However, the blind vertex cutting always creates a large amount of vertex replicas and results in heavy communication overhead for the graph computing. On the other hand, the Oblivious strategy uses a sequential greedy heuristic method to direct the placement of the subsequent edges. The goal is to minimize the conditional expected replication factor. As defined in [16], the replication factor is the ratio of the number of overall vertices in the distributed graph over that in the original input graph. In a p-way vertex-cut placement scenario, assuming each vertex (v) of the original input graph spans over $A(v)$ machines, the replication factor can be formally defined as:

$$ReplicationFactor = \frac{1}{|V|}\sum_{v \in V}|A(v)|. \tag{1}$$

Therefore, the objective of the Oblivious strategy is to place the $(i+1)$th edge after having placed the previous i edges satisfying:

$$arg\min_{j}\mathbb{E}\left[\sum_{v \in V}|A(v)|\middle|A_{e_1}...A_{e_i}, A_{(e_{i+1})} = j\right], \tag{2}$$

where A_{e_i} is the location of the ith edge, j is the ID of a machine in the distributed system. Oblivious runs the greedy heuristic independently on each machine and it reduces the number of overall vertex replicas in the distributed graph. This enhances the graph computing efficiency.

LightGraph [19] is a lightweight distributed graph-parallel synchronizing communication mechanism. Its prototype is currently implemented based on PowerGraph. That work first summarizes and defines PageRank-like algorithm, in which the direction of data access happening in an edge is always consistent with the direction of that edge. According to this feature LightGraph identifies and eliminates the redundant synchronizing communication for

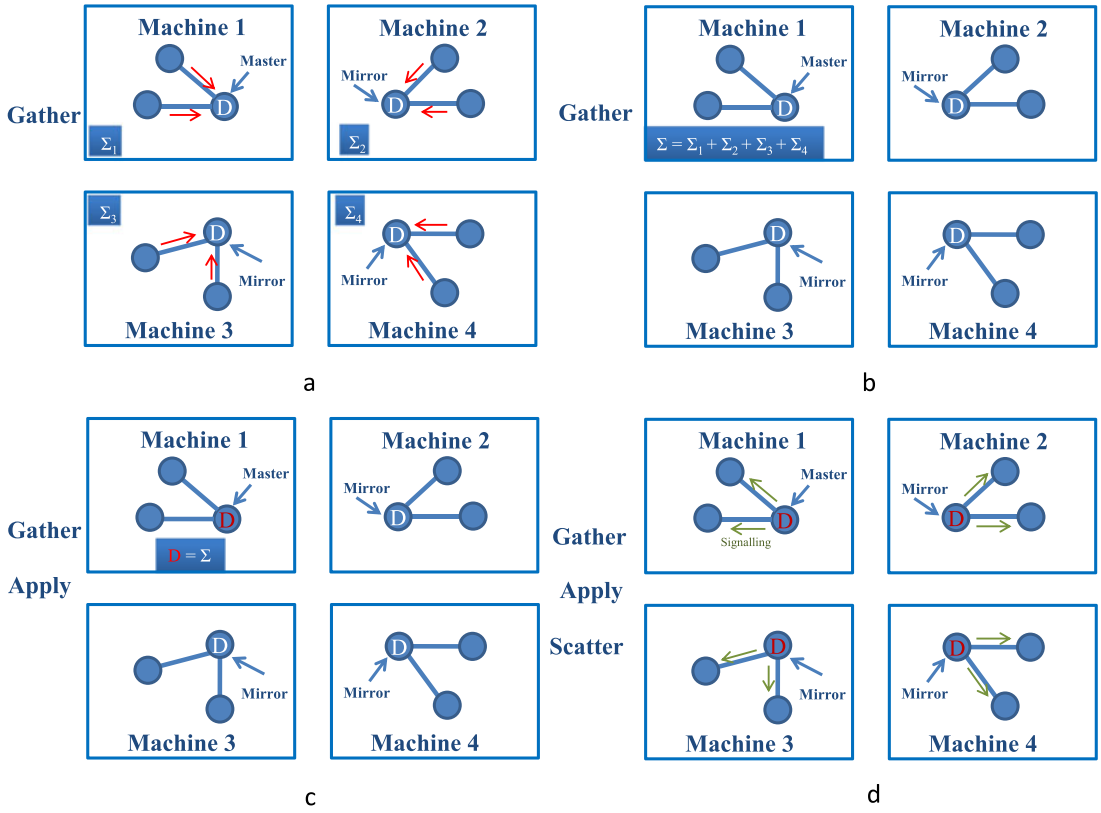


Fig. 2. The GAS computing model of PowerGraph.

PageRank-like algorithm. In particular, the mirror without outgoing edge does not need to be synchronized by its master. In order to minimize the required synchronizing communications LightGraph also proposes an edge direction-aware graph partitioning strategy, which takes the direction of edge as a heuristic parameter in the initial graph partition phase. This new graph partitioning strategy optimally isolates the outgoing edges from the incoming edges of a vertex. However, LightGraph can only reduce the synchronizing communication overhead happening in the graph computing, which limits its effectiveness. And also, LightGraph only works for PageRank-like algorithms. This seriously limits its scope of application.

3 The Partial Sum Difference Based Computing Model

Like other distributed graph-parallel computing systems, to process a large-scale graph, Ligraph first partitions a graph into smaller sub-graphs and then distributes the sub-graphs among the computing nodes. Ligraph adopts vertex-cut strategy to conduct the partition. Replicas have to be created for the vertices across the cutting-line. From the original vertex and its replicas, one is randomly selected and nominated as master. And other replicas are noted as mirrors. Computation states and data traverse the sub-graphs placed on different machines via the communications between the master and the mirrors. Similar to PowerGraph, in Ligraph communications mainly happen in the Gather result communication phase and the synchronization phase. However, different from PowerGraph Ligraph adopts a new Gather partial sum difference based computing model.

In order to detail this model, we first introduce the GAS computing model of PowerGraph. In GAS computing model the execution of a vertex program consists of three phases: Gather, Apply and Scatter (Fig. 2). In particular, Gather function runs locally on each replica of a vertex (including all mirrors and master). Once it is finished the Gather partial sum is sent from each mirror to master. The master runs the Apply function and then synchronizes all mirrors with the updated vertex data. At last, the Scatter phase is run in parallel on all replicas of this vertex.

Ligraph adopts a new Gather partial sum difference based computing model (Fig. 3). In particular, instead of calculating Gather partial sum the Gather function in Ligraph calculates the Gather partial sum difference between the former iteration and current iteration of each replica. In order to achieve this Ligraph keeps the Gather partial sum of former iteration and calculates the Gather partial sum of current iteration. Then, Ligraph calculates the difference of these Gather partial sums. On the other hand, the Apply function in this new computing model calculates the new value of vertex data by summing the original value with the Gather partial sum differences collected from all replicas. The motivation of proposing this Gather partial sum difference based computing model is to reduce the Gather communication overhead, which is detailed in the following section.

Ligraph collects the Gather partial sum differences from each replica of a vertex and sums them with the original vertex value on master side. During this whole process there is no data information missed, which guarantees that the final result of vertex data is equal to that calculated under the GAS computing model of PowerGraph. Thus, the Gather partial sum difference

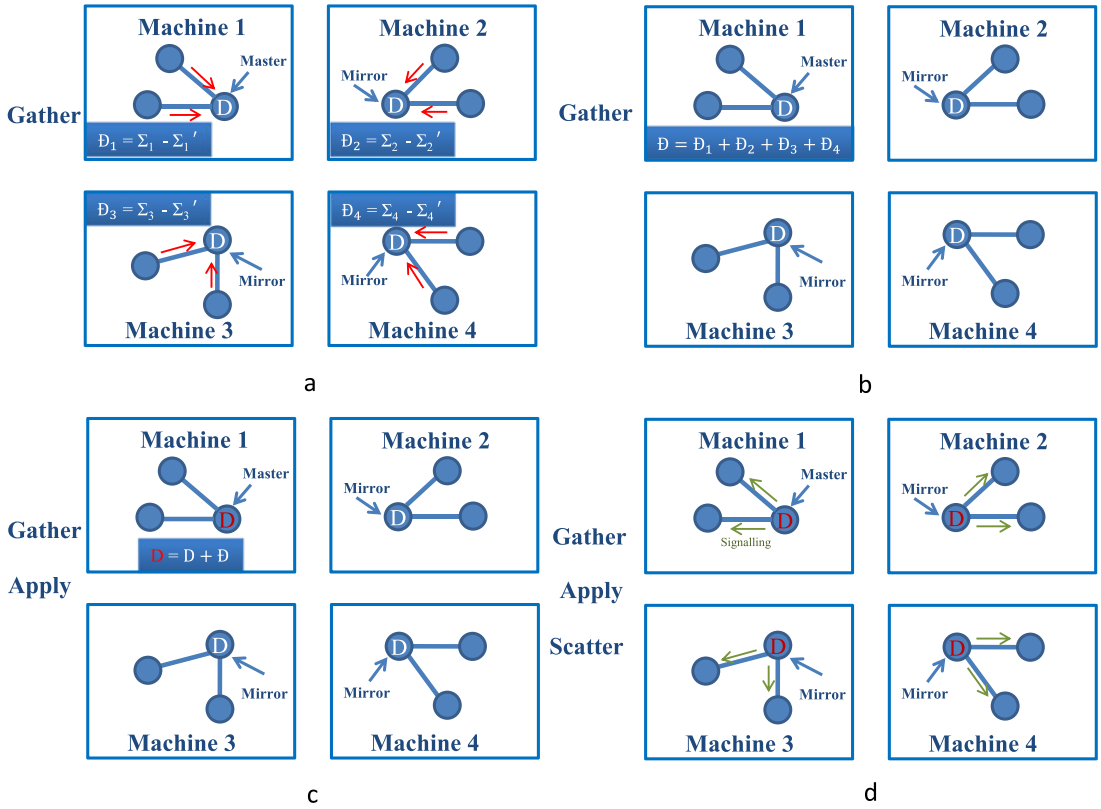


Fig. 3. The partial sum difference based computing model of Ligraph.

computing model does not impact the functionality and accuracy of the vertex-program.

Gather partial sum difference based computing model additionally introduces the difference computing operation between partial sums on each mirror. However, compared with the workload of the complex vertex-program in most machine learning algorithms this overhead is negligible.

4 The Lightweight Communication Mechanism

In PowerGraph each mirror needs to send its Gather partial sum to its master, and the master needs to synchronize all its mirrors by sending them the synchronizing message. Thus, the overall communication volume is proportional to the number of mirrors. For complex and large-scale graph, the number of mirrors may quickly expand with more computing machines used. Then the high communication overhead may limit not only the performance but also the scalability of the system.

By adopting the Gather partial sum difference based computing model, Ligraph is able to reduce the Gather communication overhead dramatically. Instead of sending Gather partial sum to master, In Ligraph, each mirror sends Gather partial sum difference to master. Thus, if the difference is equal to 0 the corresponding Gather communication is able to be avoided. So, Ligraph first checks the Gather partial sum difference value. Only when the difference is not equal to 0 the Gather communication is launched.

For PageRank-like algorithms [19] Ligraph additionally employs the lightweight synchronizing communication mechanism and edge direction-aware partition (EDAP) strategy proposed in our former work [19]. In brief, Ligraph cuts the synchronizing communications for the mirrors without outgoing edges, because the data on these mirrors will never be accessed by other vertex programs in the future computing. And in order to increase the proportion of mirrors without outgoing edges among overall mirrors Ligraph adopts the EDAP strategy, which takes the direction of edge as a heuristic parameter and optimally isolates the outgoing edges from the incoming edges of a vertex, to distribute the original graph to computing machines. In particular, Ligraph employs the EDAP_Random and EDAP_Oblivious partition methods, which are detailed in work [19]. Fig. 4 compares the communication patterns of PowerGraph and Lighraph. As the figure demonstrates compared with PowerGraph, Ligraph can achieve a lightweight communication among the working machines.

Figs. 5 and 6 illustrate the possible communication senarios for a sample graph. In this example, under EDAP placement edge $(A \rightarrow J)$ and edge $(A \rightarrow I)$, which both are outgoing edge of vertex A, are placed in the same machine (M1), and edge $(H \rightarrow A)$ and edge $(G \rightarrow A)$, which both are incoming edge of vertex A, are placed in the same machine (M3). Thus, in the new placement the mirror of vertex A on machine 3 turns to be a mirror without outgoing edge like the mirror on machine 4. Therefore the synchronizing communication for this mirror can be cut. Also, assuming in current iteration the Gather partial sum differences on mirrors on machine 0, 1, and 3 are equal to 0, then these mirrors do not need to launch Gather communications. EDAP has effectiveness on reducing the synchronizing communication overhead and does not have direct effectiveness on reducing the Gather partial sum difference communication

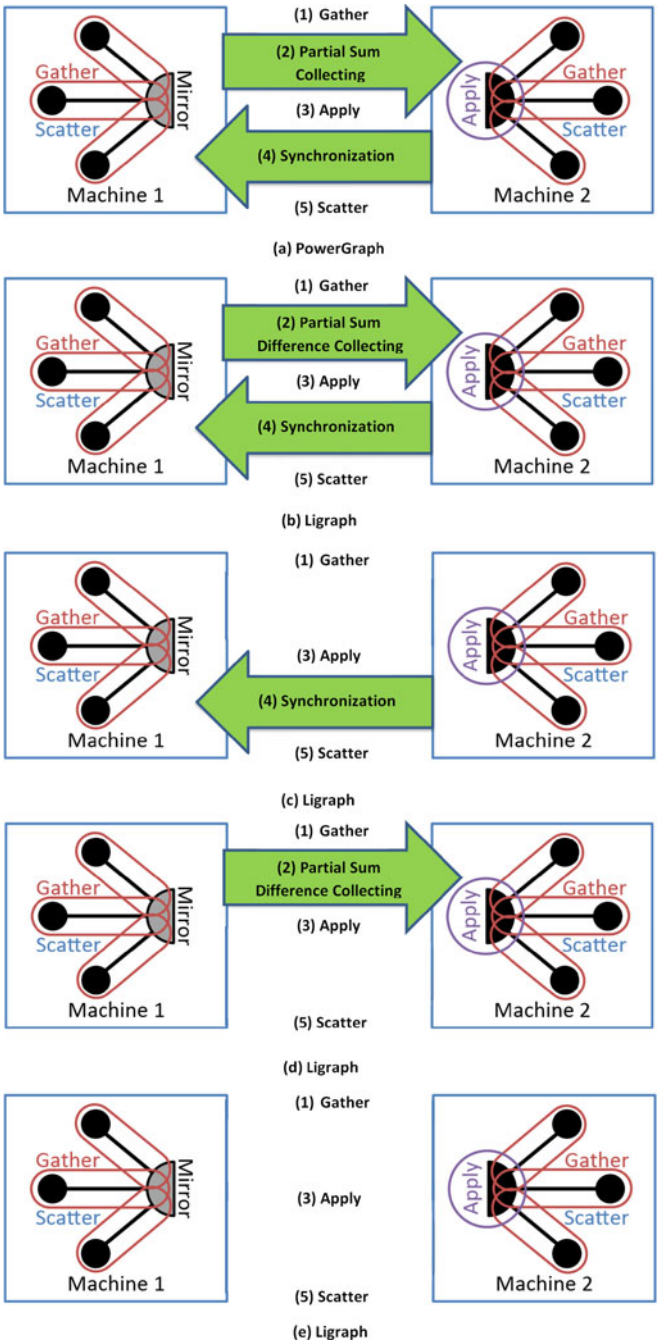


Fig. 4. The communication pattern comparing between PowerGraph and Ligraph; (a) PowerGraph; (b) Ligraph with Gather partial sum difference being not equal to 0; (c) Ligraph with Gather partial sum difference being equal to 0; (d) Ligraph with master communicating with a mirror, which has no outgoing edges, in PageRank-like algorithm and the Gather partial sum difference being not equal to 0; (e) Ligraph with master communicating with a mirror having no outgoing edges in PageRank-like algorithm and the Gather partial sum difference being equal to 0.

overhead. This is also demonstrated by our experiment results (See Figs. 7 and 8 and the related analysis).

5 Volume of Communications Analysis

In this section we conduct the volume of communications analysis. We look inside the distribution structure of a graph and explore its relationship with the volume of communications. While processing a graph $G = \{V, E\}$, in both PowerGraph and Ligraph the majority of overall

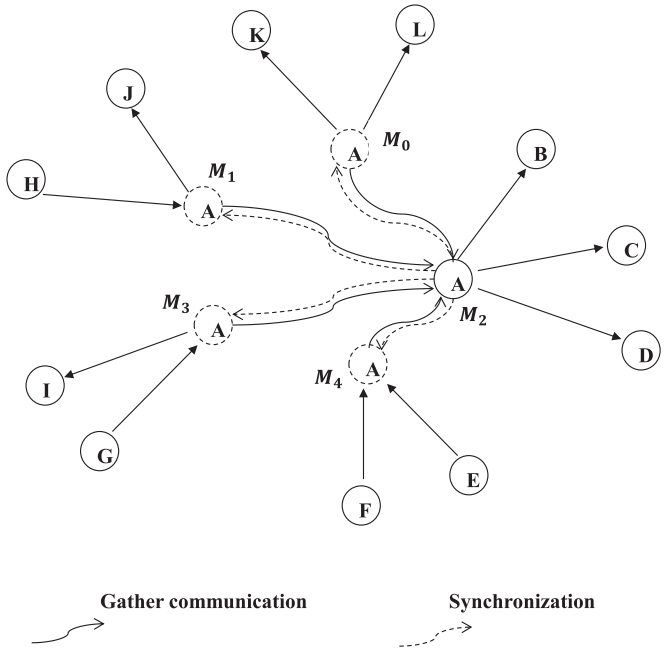


Fig. 5. An illustration of communications of PagerRank in PowerGraph.

communications happen in the Gather partial sum transmitting phase and the synchronization phase. Thus, our analysis mainly focuses on this part of communications. Because, PageRank-like algorithms can gain best benefit from Ligraph, in order to demonstrate the optimal effectiveness of Ligraph the extreme analysis in this section is conducted for Pagerank-like algorithms. Table 1 explains the related notations.

5.1 General Analysis

Given a vertex, v, according to the graph partition process in PowerGraph and Ligraph all mirrors of v have edges. Thus v's mirrors can be classified into the following three classes:

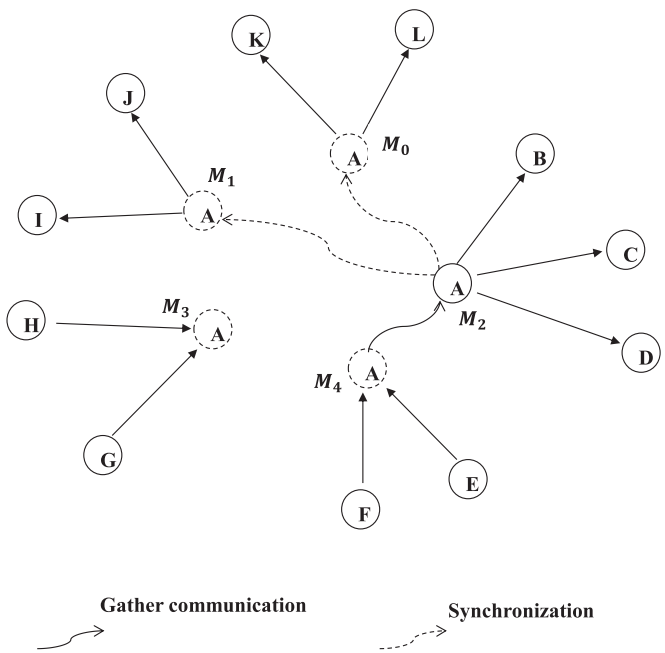


Fig. 6. An example of communications of PagerRank under EDAP-based graph placement in Ligraph.

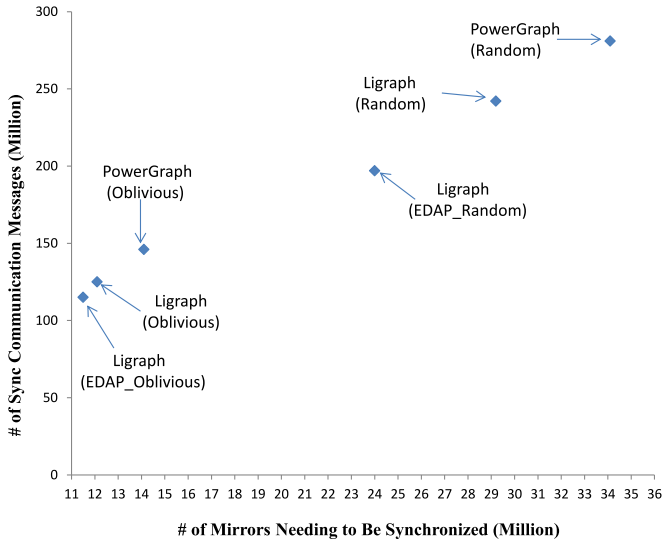


Fig. 7. The number of synchronizing communication messages versus the number of mirrors needing to be synchronized.

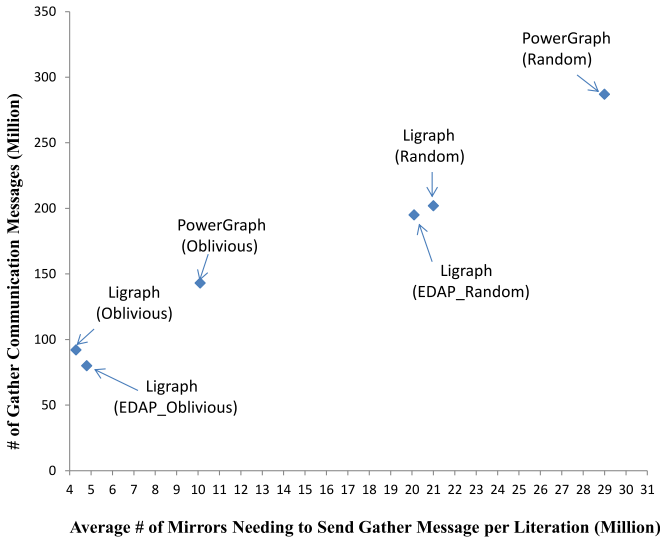


Fig. 8. The number of Gather communication messages versus the average number of mirrors needing to send Gather message per iteration.

a. mirrors with both incoming and outgoing edge;
b. mirrors with outgoing edge and without incoming edge;
c. mirrors with incoming edge and without outgoing edge;

We also introduce a flag, $\emptyset(v,i)$, which subjects to

$$\emptyset(v,i) = \begin{cases} 0 & v \text{ is not active in the } i\text{th ireration} \\ 1 & v \text{ is active in the } i\text{th ireration.} \end{cases}$$

In PowerGraph. All mirrors need to send its Gather partial sum to its master. And on the master side, after the Apply phase is done, data on master is updated. Then the master will synchronize all its mirrors with the new data.

All v's mirrors need to be synchronized by v's master. Thus, the number of synchronizing messages happening on vertex v is:

$$|v.sync| = \sum_{i=0}^{n-1}\left[\emptyset(v,i)\sum_{j=0}^{2}|v\_m_j|\right]. \tag{3}$$

All v's mirrors need to calculate their partial sums and deliver these partial sums to v's master. Thus, the number of Gather communication messages happening on vertex v is:

$$|v.gather| = \sum_{i=0}^{n-1}\left[\emptyset(v,i)\sum_{j=0}^{2}|v\_m_j|\right]. \tag{4}$$

Thus, the total number of communication messages happening on vertex v is:

$$\begin{aligned} &|v.total| \\ &= |v.sync| + |v.gather| \\ &= \sum_{i=0}^{n-1}\left[\emptyset(v,i)\sum_{j=0}^{2}|v\_m_j|\right] + \sum_{i=0}^{n-1}\left[\emptyset(v,i)\sum_{j=0}^{2}|v\_m_j|\right] \\ &= 2\sum_{i=0}^{n-1}\left[\emptyset(v,i)\sum_{j=0}^{2}|v\_m_j|\right]. \end{aligned} \tag{5}$$

There is no duplicated communication between any two different vertices. Consequently, the total number of communication messages happening in the whole graph is:

$$\begin{aligned} |V.total| &= \sum_{i=0}^{|V|-1}|v_i.total| \\ &= 2\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\left[\emptyset(v_i,j)\sum_{k=0}^{2}|v_i\_m_k|\right]. \end{aligned} \tag{6}$$

In Ligraph. In Ligraph, v's mirror with no outgoing edge does not need to be synchronized by v's master.

TABLE 1
Notations

Symbol	Description
n	The number of iterations in a graph computing job
$\emptyset(v,i)$	The flag indicating whether v is active or not in the ith iteration
$r(v,i)$	The ratio of v's mirrors with Gather partial sum difference is not equal to 0 in the ith iteration
$\|v\_m_0\|$	The number of v's mirrors with both incoming and outgoing edge
$\|v\_m_1\|$	The number of v's mirrors with no incoming edge
$\|v\_m_2\|$	The number of v's mirrors with no outgoing edge
$\|v.sync\|$	The number of synchronizing messages happening on vertex v
$\|v.gather\|$	The number of Gather communication messages happening on vertex v
$\|v.total\|$	The total number of communication messages happening on vertex v
$\|V.total\|$	The number of communication messages happening on all vertices computing
$P\_reduced\_comm$	Percentage of reduced communication messages

Thus, the number of synchronizing messages happening on vertex v is:

$$|v.sync| = \sum_{i=0}^{n-1}[\o(v,i)(|v\_m_0| + |v\_m_1|)]. \tag{7}$$

In Ligraph, v's mirror with Gather partial sum being equal to 0 does not need to launch Gather communication to v's master. Thus, the number of Gather communication messages happening on vertex v is:

$$|v.gather| = \sum_{i=0}^{n-1}[\o(v,i)r(v,i)\sum_{j=0}^{2}|v\_m_j|]. \tag{8}$$

The number of total communication messages happening on vertex v is:

$$\begin{aligned}
&|v.total| \\
&= |v.sync| + |v.gather| \\
&= \sum_{i=0}^{n-1}[\o(v,i)(|v_m_0| + |v_m_1|)] + \sum_{i=0}^{n-1}\left[\o(v,i)r(v,i)\sum_{j=0}^{2}|v_m_j|\right] \\
&= \sum_{i=0}^{n-1}\left[\o(v,i)\left(|v_m_0| + |v_m_1| + r(v,i)\sum_{j=0}^{2}|v_m_j|\right)\right] \\
&= \sum_{i=0}^{n-1}\left[\o(v,i)((1+r(v,i))\sum_{j=0}^{2}|v_m_j| - |v_m_2|)\right].
\end{aligned} \tag{9}$$

Consequently, the number of total communication messages happening in the whole graph is:

$$\begin{aligned}
|V.total| &= \sum_{i=0}^{|V|-1}|v_i.total| \\
&= \sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\left[\o(v_i,j)((1+r(v_i,j))\sum_{k=0}^{2}|v_i_m_k| - |v_i_m_2|)\right].
\end{aligned} \tag{10}$$

Thus the number of reduced communication messages achieved by Ligraph over PowerGraph is:

$$\begin{aligned}
&|V.total|_{Reduced} \\
&= 2\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\left[\o(v_i,j)\sum_{k=0}^{2}|v_i_m_k|\right] \\
&- \sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\left[\o(v_i,j)((1+r(v_i,j))\sum_{k=0}^{2}|v_i_m_k| - |v_i_m_2|)\right] \\
&= \sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\left[\o(v_i,j)((1-r(v_i,j))\sum_{k=0}^{2}|v_i_m_k| + |v_i_m_2|)\right].
\end{aligned} \tag{11}$$

Thus,

$$\begin{aligned}
&P_reduced_comm/100 \\
&= \frac{\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}\Theta}{2\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[\o(v_i,j)\sum_{k=0}^{2}|v_i_m_k|]},
\end{aligned} \tag{12}$$

in which

$$\Theta = \o(v_i,j)((1-r(v_i,j))\sum_{k=0}^{2}|v_i\_m_k| + |v_i\_m_2|). \tag{13}$$

5.2 Extreme Analysis

In Ligraph, it is possible that all outgoing edges of a vertex v are aggregated in a small number of v's mirrors. Thus, in the extreme case:

$$\lim[\sum_{i=0}^{|V|-1}[|v_i\_m_2|] = \sum_{i=0}^{|V|-1}\sum_{k=0}^{2}[|v_i\_m_k|]. \tag{14}$$

And in the extreme case, in each iteration, all mirrors' Gather partial sum difference are equal to 0. Thus,

$$\lim\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[r(v_i,j)] = 0. \tag{15}$$

Thus, in the extreme case:

$$\begin{aligned}
&\lim[P_reduced_comm/100] \\
&= \lim\left[\frac{\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[\o(v_i,j)(\sum_{k=0}^{2}|v_i_m_k| + |v_i_m_2|)]}{2\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[\o(v_i,j)\sum_{k=0}^{2}|v_i_m_k|]}\right] \\
&= \lim\left[\frac{\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[\o(v_i,j)2\sum_{k=0}^{2}|v_i_m_k|]}{2\sum_{i=0}^{|V|-1}\sum_{j=0}^{n-1}[\o(v_i,j)\sum_{k=0}^{2}|v_i_m_k|]}\right] \\
&= 1.
\end{aligned} \tag{16}$$

Thus, in the extreme case, Ligraph can eliminate all communication overhead happening in PowerGraph.

Table 2 compares the key characteristics of Ligraph with three state-of-the-art graph parallel platforms.

6 Experimental Evaluation

In this section, we demonstrate the comparative effectiveness of various aspects of Ligraph over PowerGraph and LightGraph through experiments.

6.1 Experiment Environment

Our experiments were conducted on a 65-node (528 processors) Linux-based cluster. The cluster consists of one front-end node that runs the TORQUE resource manager and the Moab scheduler and 64 computing (worker) nodes. Each computing node has 16 GB of RAM and 2 quad-core Intel Xeon 2.66 GHz CPUs. The /home directory is shared among all nodes through NFS. Due to some hardware resource limitation, not all the 64 computing nodes can be used. Thus, we used up to 48 nodes in our experiment.

6.2 Benchmarking Algorithms and Data Set

We selected PageRank, SSSP (directed shortest path) and Triangle counting detailed in Table 3 as benchmarking algorithms. And we computed three data sets listed in Table 4. These data sets are all large-scale graphs. The selection standard is to select graphs extracted from real-world use with diverse characteristics and different scales in size.

6.3 Experiment Design and Results

We ran the benchmarking algorithms on the selected data sets and compared metrics measured in Ligraph with those

TABLE 2
Comparing Key Characteristics of Ligraph with Existing Graph-Parallel Platforms

Metrics	Pregel [10]	GraphLab [15]	PowerGraph [16]	Ligraph
Comm. overhead	$\propto$ # of edge-cuts	$\propto$ # of edge-cuts	$\propto$ # of mirrors	$\propto$ # of partial mirrors
Graph placement	Edge-cut	Edge-cut	Random and Greedy vertex-cut	Random, Greedy and EDAP vertex-cut
Computing model	Sync.	Sync. & Async.	Sync. & Async.	Sync. & Async.
Dynamic comm.	no	no	yes	yes
Load balance	no	no	yes	yes

TABLE 3
Summary of Benchmarking Algorithms

Algorithm	Characteristics	Application Illustration	Category
PageRank	Iterative, high communication	Importance ranking	PageRank-like
SSSP (shortest path)	Iterative, medium communication	Decision making	PageRank-like
Triangle Counting	Single step, medium communication	Clustering coefficient	General

TABLE 4
Summary of Data Sets

Graph	Description	# of vertices	# of edges	Graph density ($\times 10^{-5}$)	Average degree	Memory size (GB)
soc-LiveJournal [18]	Friednship social network	4,847,571	68,993,773	0.59	14	10.3
Twitter [18]	Social news website	11,316,811	85,331,846	0.13	8	23.2
BFS1 [20]	Facebook social networks	61,876,615	336,776,269	0.02	5	51.6

in PowerGraph and LightGraph, respectively. Experiments are conducted under both synchronous and asynchronous computation modes and all presented results come from the average of at least three runs.

First, we ran PageRank on sov-LiveJournal data set. And we measured the numbers of each kind of mirrors and the total numbers of various communication messages happening in the whole graph computing process. In particular, we measured the Gather partial sum communications and the synchronizing communications, which dominates the communications in the whole job in PowerGraph and Ligraph. We conducted 100 times running for each case and calculated their average value. Figs. 7 and 8 plot the results under synchronous mode using 16 machines. In Fig. 7 the numbers of mirrors needing to be synchronized under Ligraph are actually the numbers of mirrors with outgoing edge. As Fig. 7 shows 86.8 and 89.6 percent of overall mirrors have outgoing edge under Random and Oblivious placement, respectively. And Ligraph only synchronizes this proportion of mirrors. Instead PowerGraph needs to synchronize all mirrors. Consequently, the numbers of synchronizing communication messages are reduced by 14.7 and 10.8 percent by Ligraph (Random) and Ligraph (Oblivious) over PowerGraph (Random) and PowerGraph (Oblivious), respectively. And under EDAP_Random and EDAP_Oblivious the percentages of mirrors with outgoing edge are reduced to 71.5 and 82.8 percent, respectively. Consequently, the numbers of synchronizing communication messages are reduced by 26.4 and 16.5 percent by Ligraph (EDAP_Random) and Ligraph (EDAP_Oblivious) over PowerGraph (Random) and PowerGraph (Oblivious), respectively. Fig. 8 demonstrates the result of the number of Gather communication messages happening in the whole job and the average number of mirrors needing to send Gather message per iteration. As expected, compared with PowerGraph Ligraph reduced the number of mirrors needing to send Gather message and consequently the number of Gather communication messages is reduced. The result also demonstrates that there is no strong correlation between the graph placement strategy and the effectiveness of Ligraph in reducing the volume of Gather communication overhead. This is because redistributing edges among machines according to the edge direction does not have direct influence on the Gather partial sum difference of a mirror.

Fig. 9 shows the volume of communications happening in PageRank running in PowerGraph and Ligraph, respectively. The data set computed is the Twitter data set. Fig. 10 shows the corresponding results on LiveJournal data set. As expected, Ligraph and its EDAP strategy can significantly reduce the communication overhead for PageRank. For example, for LiveJournal dataset Ligraph (EDAP_Random) can consistently reduce at least 20.8 percent communications when the number of machines is larger than 4 under synchronous mode over PowerGraph (Random). Under asynchronous mode, over PowerGraph (Random) the maximal communication reduction achieved by Ligraph (EDAP_Random) is 35.2 percent while processing Twitter data set. Moreover, as the number of machines increases, the volume of communications reduced by Ligraph also increases.

Figs. 11 and 12 show the PageRank runtime on Twitter and LiveJournal data set, respectively. By reducing the volume of communications, Ligraph shortens the runtime of PageRank under both synchronous and asynchronous modes. Moreover, by using EDAP strategy Ligraph further accelerates the execution of PageRank. For example, for Twitter data set, over asynchronous PowerGraph (Random) the maximal runtime reduction achieved by asynchronous

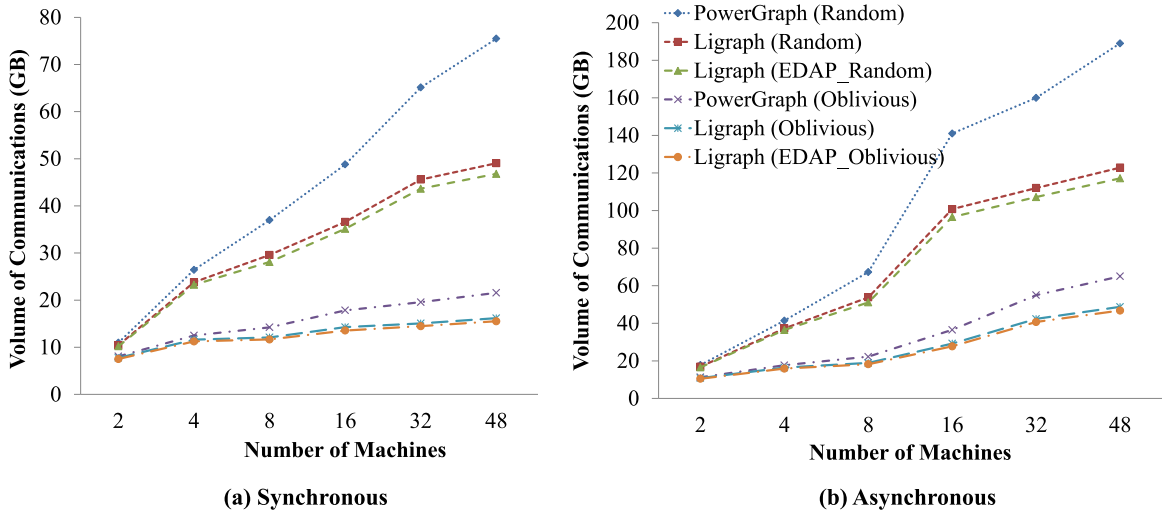


Fig. 9. Comparing the volume of communications for Twitter between Ligraph and PowerGraph.

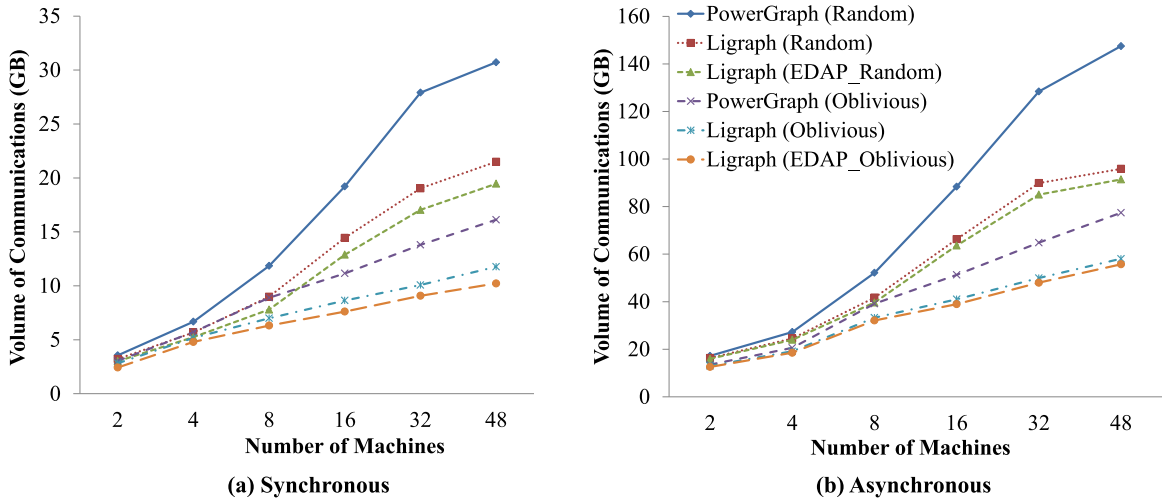


Fig. 10. Comparing the volume of communications for LiveJournal between Ligraph and PowerGraph.

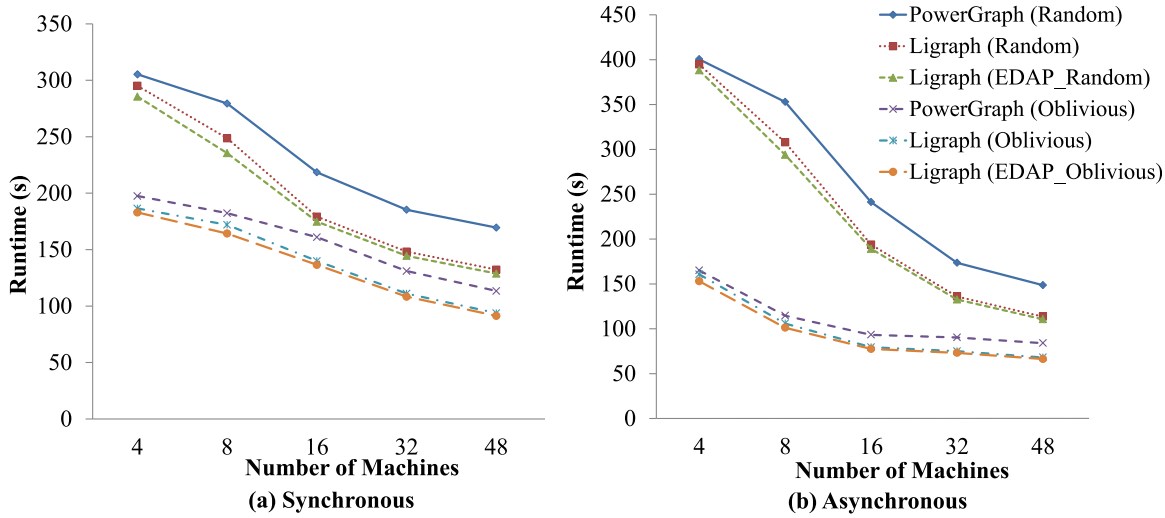


Fig. 11. Comparing the PageRank runtime for Twitter between Ligraph and PowerGraph.

Ligraph (EDAP_Random) is 21.8 percent. Furthermore, Ligraph shows consistent performance gains as the number of machines used increases. For instance, among all cases presented Ligraph (EDAP_Oblivious) can shorten at least 14.6 percent runtime over PowerGraph (Oblivious) on LiveJournal dataset under synchronous mode.

Our experiment results of SSSP also verify the effectiveness of Ligraph and its EDAP partition strategy. We took the

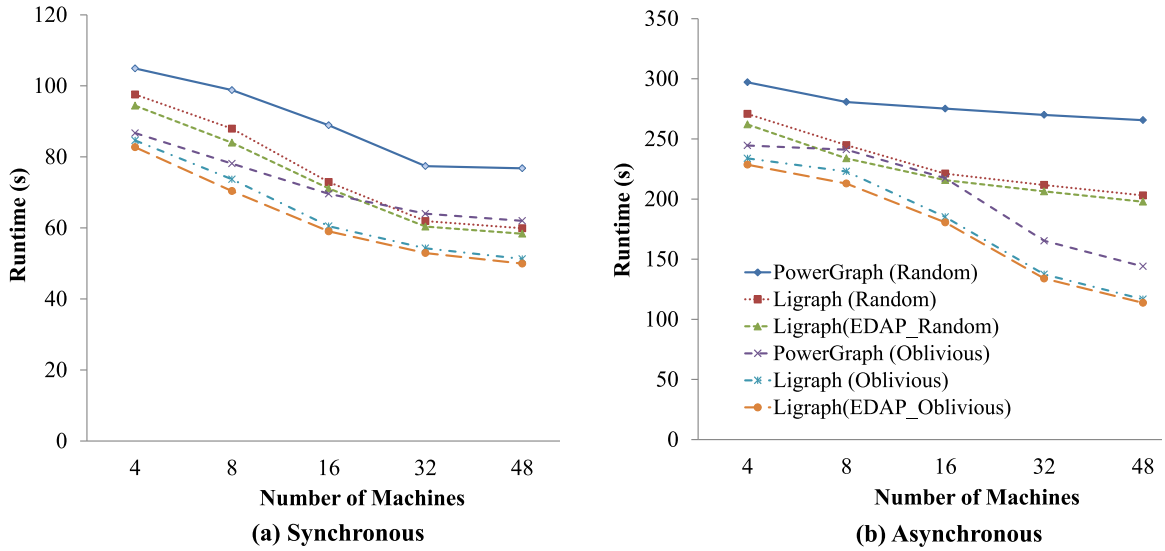


Fig. 12. Comparing the PageRank runtime for LiveJournal between Ligraph and PowerGraph.

results processing BFS1 data set under synchronous mode as example and presented them in Fig. 13. As the figure shows, up to 30.2 and 16.1 percent improvement in volume of communication and runtime are achieved by Ligraph with EDAP strategy, respectively. And, better effectiveness is achieved with the number of machines used increasing.

Although, for the benchmarking algorithms, the communication overhead has been drastically reduced, the overall runtime does not decrease significantly in the same scale. The reason is that a bulk of communication overhead in executing these algorithms can be already hidden in the GAS three-phase programming model of PowerGraph. Moreover, for a CPU-bound algorithm such as PageRank, the effect of eliminating communication overhead on runtime performance will not be that evident.

Fig. 14 shows the volume of communications of PageRank in LightGraph and Ligraph processing the Twitter data set under asynchronous and synchronous computation mode, respectively. Fig. 15 shows the corresponding results on LiveJournal data set. As the figures demonstrate, compared with LightGraph the volumes of communications in PageRank execution under both modes in Ligraph are reduced. For example, for LiveJournal dataset Ligraph (EDAP_Oblivious) can consistently reduce at least 15.2 percent communications when the number of machines is larger than 2 under synchronous mode over LightGraph (EDAP_Oblivious); Over asynchronous LightGraph (EDAP_Random) the maximal network I/O reduction achieved by asynchronous Ligraph (EDAP_Random) is 21.8 percent while processing the Twitter data set. Moreover, as the number of machines increases, the volume of communications reduced by Ligraph also increases.

Figs. 16 and 17 compare the PageRank runtime in LightGraph and Ligraph on Twitter and LiveJournal data set, respectively. As shown in the figures, compared with LightGraph Ligraph shortens the runtime of PageRank under both synchronous and asynchronous modes. For example, for Twitter data set, over synchronous LightGraph (EDAP_Random) the maximal runtime reduction achieved by synchronous Ligraph (EDAP_Random) is 14.3 percent.

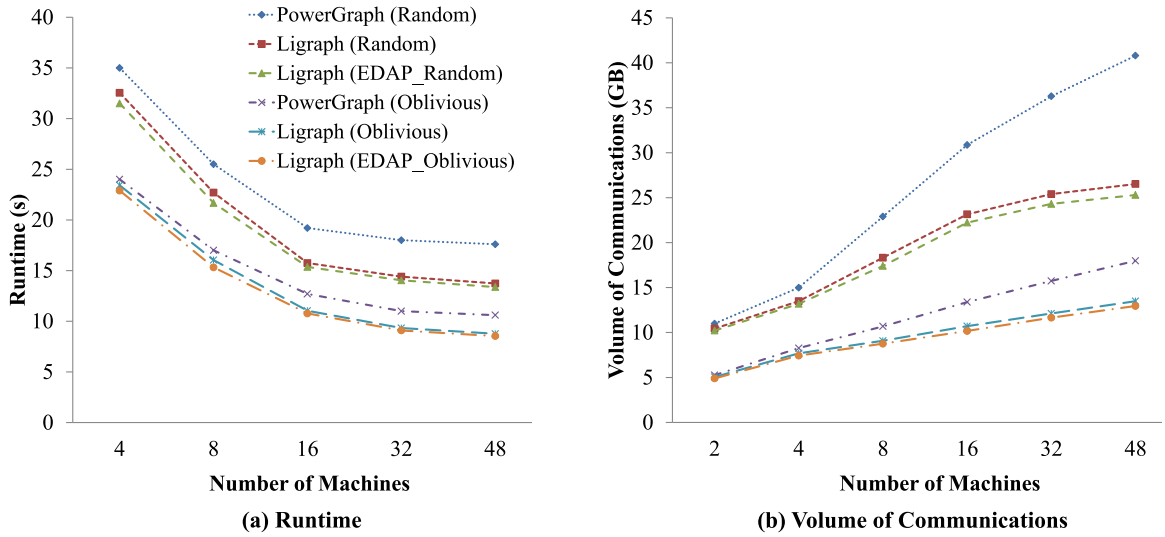


Fig. 13. Comparing the runtime and volume of communications for SSSP running on BFS1 dataset under synchronous computation mode between Ligraph and PowerGraph.

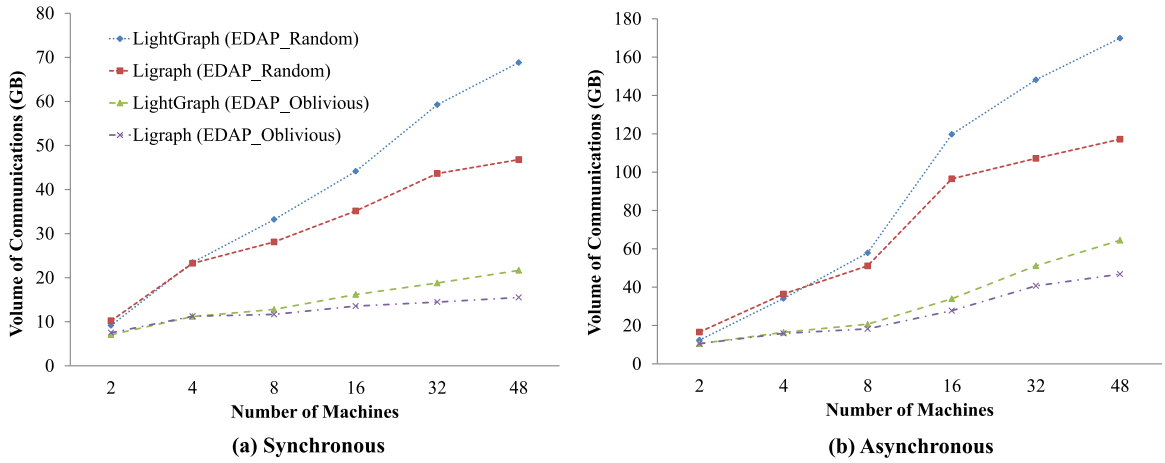


Fig. 14. Comparing the volume of communications for Twitter between Ligraph and LightGraph.

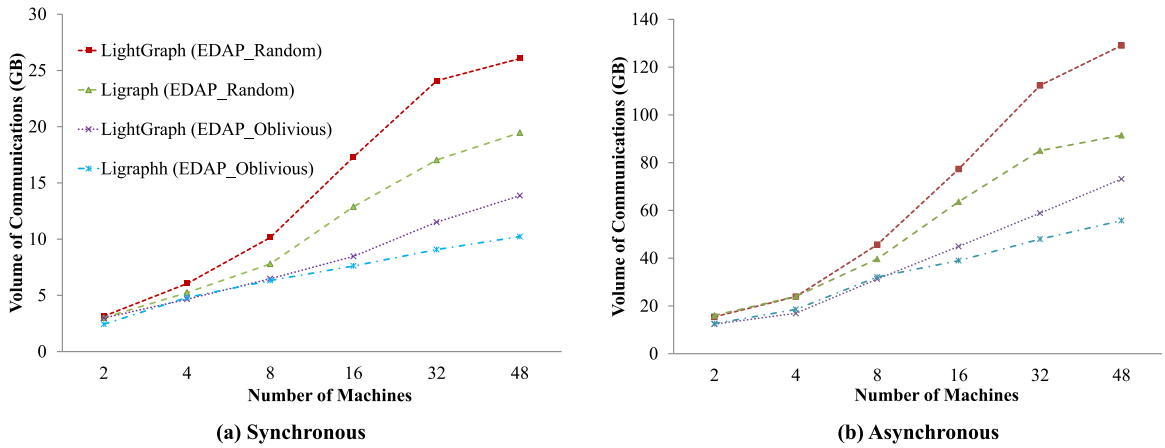


Fig. 15. Comparing the volume of communications for LiveJournal between Ligraph and LightGraph.

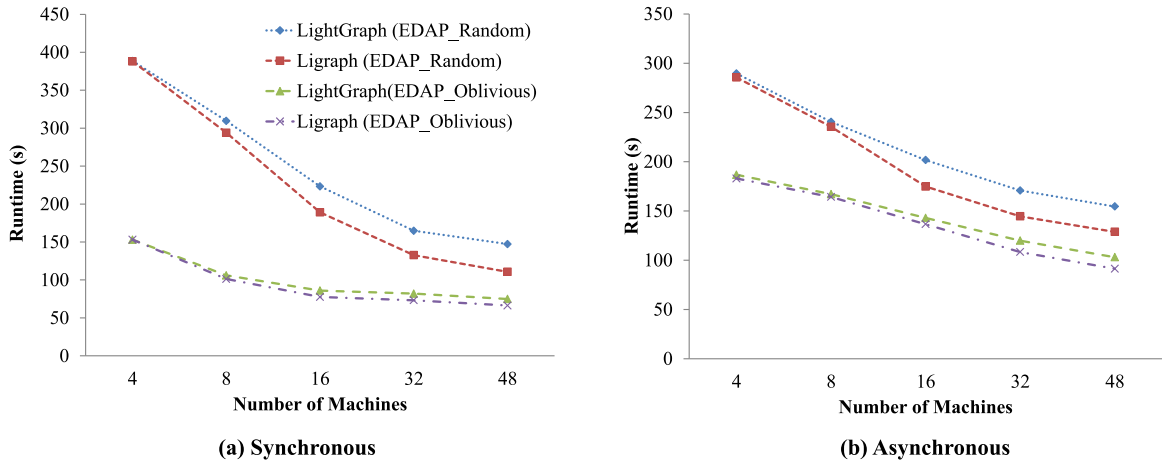


Fig. 16. Comparing the PageRank runtime for Twitter between Ligraph and LightGraph.

Furthermore, Ligraph shows better performance gains as the number of machines used increases.

In all presented experiments we outputted the final computing results of the algorithms and compared the results among Ligraph, PowerGraph, and LightGraph. The results are all consistent.

Fig. 18 compares the graph computing speedup of PageRank between PowerGraph and Ligraph. The speedup of PowerGraph is low. Especially, the speedup under Random placement is the worst, which is less than 2.0 using 48 machines. The reason of this poor speedup is that Random placement creates a large number of replicas of vertices, which increases both the computing workload and the communication overhead among computing nodes. Oblivious placement reduces the number of vertex replicas by greedily placing edges on machines, which already have the vertices in that edge. Thus, the speedup is improved. Upon PowerGraph, Ligraph lightens the communication

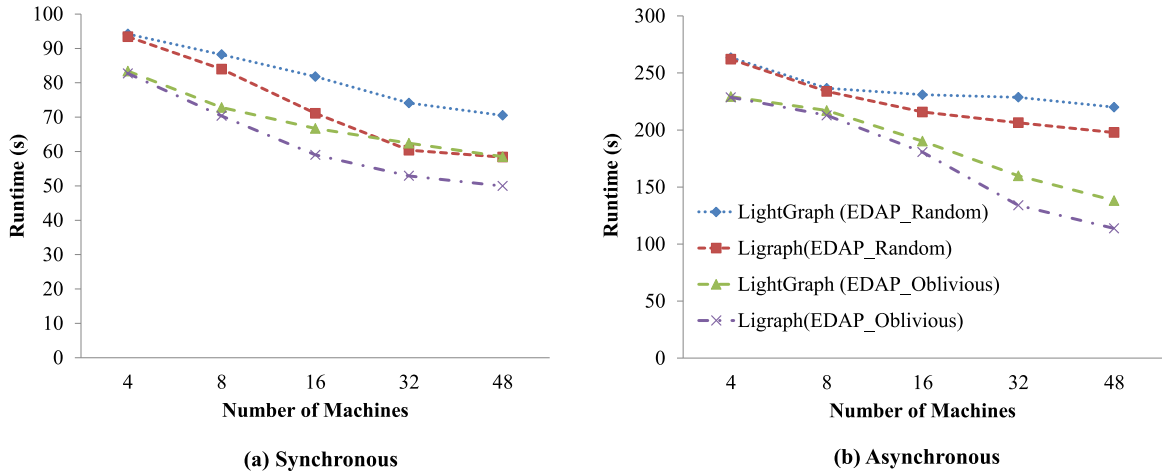


Fig. 17. Comparing the PageRank runtime for LiveJournal between Ligraph and LightGraph.

overhead happening in the graph computing. Consequently, the speedup is increased. Better speedup achieved by Ligraph and its EDAP partition strategy also demonstrates the effectiveness of them in large scale distributed graph computing.

In Figs. 19 and 20 we provide runtime comparisons of Ligraph with several representative existing systems for PageRank and Triangle counting, respectively. In our experiments Giraph, GPS, and GraphX all adopt their system default graph partition strategy: random edge-cut partitioning. And because all these there systems do not support asynchronous graph computing they all conduct the graph computing under synchronous computing mode. As the results demonstrate Ligraph outperforms other systems in runtime for both PageRank and Triangle counting. For example, while computing Livejournal data set the runtimes of Giraph and GPS are 122.3 and 110.6 s, respectively. On the other hand, the runtime of Ligraph (Sync, EDAP_Oblivious) is only 49.9 s. And for Twitter data set the runtimes of GraphX and PowerGraph (sync random) are 137.8 and 42.1 s, respectively, while the runtime of Ligraph (Sync, Random) is only 35.6 s.

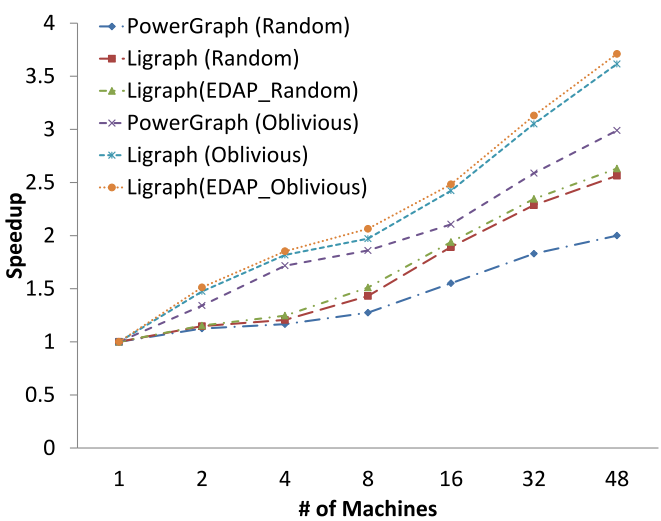


Fig. 18. Comparing the PageRank runtime speedup between PowerGraph and Ligraph under synchronous mode while computing the Twitter data set.

7 Related Work

A number of distributed graph-parallel abstractions have emerged in literatures. Pregel [10] explores graph-parallelization through the use of a bulk synchronous distributed message-passing system. Several other systems are successor of Pregel including GPS [11], Giraph [13], GoldenOrb [21], Mizan [22], and Phoebus [23]. Gregor and Lumsdaine proposed the parallel BGL [9]: a generic C++ library for distributed graph computation and applies the paradigm of generic programming to the domain of graph computations. Kineograph [12] takes a stream of incoming data to construct a continuously changing graph, which captures the relationships that exist in the data feed. Stutz et al. proposed the Signal-Collect [14] framework to concisely specify and execute a number of computations that are typical for Semantic Web. Gunrock [24], Medusa [25], CuSha [26], and MapGraph [27] are designed for the large-scale graph analytics on the GPU. Among the numerous graph-parallel computing systems PowerGraph [16], PowerLyra [17], GRACE [28] and Trinity [29] support both synchronous and asynchronous executing modes for graph algorithms. Naiad [30], [31] is able to conduct incremental iterative computation. However, it adopts traditional synchronous check pointing for fault tolerance and cannot respond to stragglers

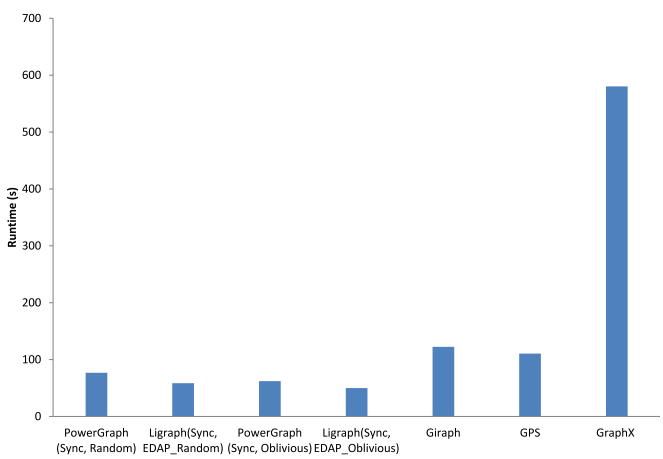


Fig. 19. PageRank runtime comparison between multiple systems on Livejournal data set while using 48 machines.

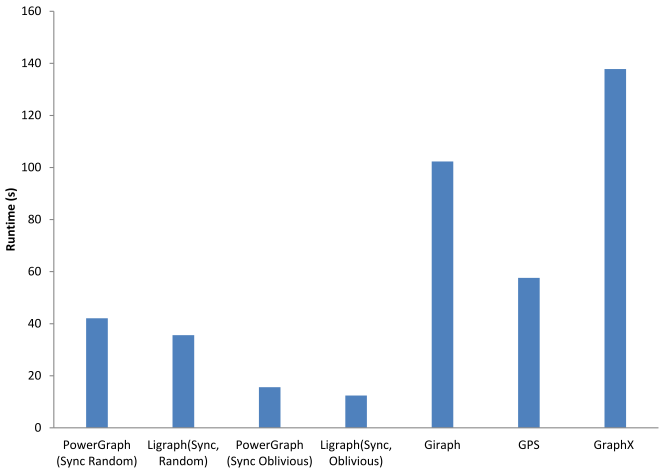


Fig. 20. Triangle counting runtime comparison between multiple systems on Twitter data set while using 48 machines.

[32]. PowerLyra [17] dynamically applies different computation and partitioning strategies for different vertices. Distributed GraphLab [15] and its successor, PowerGraph [16] exhibit more excellent performance than others with better graph processing rate and higher scalability [33], [34], [35]. Cyclops [36] is also a vertex-oriented graph-parallel framework. However, compared with PowerGraph (written in C ++) its java implementation based on Hama [37] drags its runtime performance down.

In order to deal with the inherent problem, communication overhead, in distributed computing systems, much effort has been done as well. In traditional message passing abstractions, such as Pregel [10], Giraph [13], and GPS [11], all vertex-programs run simultaneously in a sequence of supersteps. In each super-step, each program instance receives all messages sent by its neighbors in the previous super-step and sends messages to its neighbors for next super-step [16]. In order to reduce the number of communication messages, Pregel introduces a commutative associative message combiner, which merges messages destined to a same vertex [10]. Work [38] proposes asynchronous broadcast and reduction operations to reduce communication associated with high-degree vertices. PowerGraph [16] abstraction employs GAS (Gather, Apply, and Scatter) graph computing model and ensures the changes made to the vertex or edge data are automatically visible to adjacent vertices. Thus, PowerGraph eliminates the messages transferred between adjacent vertices. LFGraph [39] uses techniques such as cheap hash-based graph partitioning, publish-subscribe information flow, fetch-once communication, single-pass computation, in-neighbor storage and so on, which incur lower communication overhead than other systems. LightGraph [19] tries to identify and eliminate the unnecessary communications in distributed graph-parallel platforms. However it just looks at the synchronous communication.

8 Conclusion and Future Work

Driven by the need to process large-scale graph data numerous distributed graph-parallel computing systems have been proposed. However, the communication overhead in distributed graph-parallel computing systems drags down the performance of these systems. This work proposes Ligraph, a distributed graph-parallel communication system with lightweight communication overhead, to address this problem. Our extensive experiment results on real-world network data sets have demonstrated that compared with PowerGraph and LightGraph Ligraph can not only reduce the volume of communications significantly but also improve the graph computing runtime performance.

Distributed big-data processing systems make it feasible to perform computations on large volumes of data with high complexity. However, the communication overhead in these big-data computing frameworks are often overlooked. Research on this topic will not only help to accelerate the big-data processing jobs themselves but also alleviate network I/O workload of the underlying computing hardware systems, which are shared by a number of applications on HPC or cloud systems. This paper demonstrates the potential and positive results of work in this direction.

In the future we would like to attempt to introduce and implement some communication reducing mechanisms proposed in other systems (e.g., the Combiner mechanism proposed by Pregel [10] and the dynamic repartitioning scheme in GPS [11]) in Ligraph to further reduce the communication overhead suffered by distributed graph-parallel computing.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant CRI CNS-0855248, and Grant MRI CNS-0619069.

References

[1] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a PC," in *Proc. 10th USENIX Conf. Operating Syst. Design Implementation*, 2012, pp. 31–46.

[2] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in *Proc. Conf. Uncertainty Artif. Intell.*, Jul. 2010.

[3] W.-S. Han, S. Lee, K. Park, J.-H. Lee, M.-S. Kim, J. Kim, and H. Yu, "Turbograph: A fast parallel graph engine handling billion-scale graphs in a single PC," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 77–85.

[4] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proc. 18th ACM SIGPLAN Symp. Principles Practice Parallel Program.*, 2013, pp. 135–146.

[5] R. Pearce, M. Gokhale, and N. M. Amato, "Multithreaded asynchronous graph traversal for in-memory and semi-external memory," in *Proc. ACM/IEEE Int. Conf. High Perform. Comput., Netw. Storage Anal.*, 2010, pp. 1–11.

[6] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haridasan, "Managing large graphs on multi-cores with graph awareness," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2012, pp. 4–4.

[7] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proc. 24th ACM Symp. Operating Syst. Principles*, 2013, pp. 472–488.

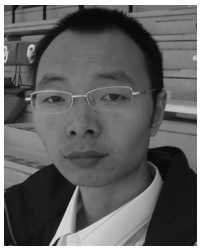
[8] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *ACM SIGPLAN Notices*, vol. 48, pp. 135–146, 2013.

[9] D. Gregor and A. Lumsdaine, "The parallel BGL: A generic library for distributed graph computations," in *Proc. Parallel Object-Oriented Sci. Comput.*, 2005, pp. 1–18.

[10] G. Malewicz, M. H. Austern, A. J. C Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.

[11] S. Salihoglu and J. Widom, "GPS: A graph processing system," in *Proc. 25th Int. Conf. Sci. Statist. Database Manage.*, 2013, pp. 22:1–22:12.

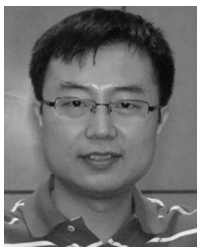
[12] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen, "Kineograph: Taking the pulse of a fast-changing and connected world," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 85–98.
[13] (2012). Apache incubator giraph [Online]. Available: http://incubator.apache.org/giraph/
[14] P. Stutz, A. Bernstein, and W. Cohen, "Signal/collect: Graph algorithms for the (semantic) web," in *Proc. 9th Int. Semantic Web Conf. Semantic Web - Vol. Part I*, 2010, pp. 764–780.
[15] 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," in *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012.
[16] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 17–30.
[17] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, p. 1.
[18] (2006). Snap [Online]. Available: http://snap.stanford.edu/data/
[19] Y. Zhao, K. Yoshigoe, M. Xie, S. Zhou, R. Seker, and J. Bian, "Lightgraph: Lighten communication in distributed graph-parallel processing," in *Proc. IEEE Int. Congr. Big Data*, 2014, pp. 717–724.
[20] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in Facebook: A case study of unbiased sampling of osns," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[21] (2011). Goldenorb [Online]. Available: http://www.raveldata.com/goldenorb/
[22] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, "Mizan: A system for dynamic load balancing in large-scale graph processing," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 169–182.
[23] (2010). Phoebus [Online]. Available: https://github.com/xslogic/phoebus/
[24] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proc. 20th ACM SIGPLAN Symp. Principles Practice Parallel Program.*, 2015, pp. 265–266.
[25] J. Zhong and B. He, "Medusa: Simplified graph processing on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1543–1552, Jun. 2014.
[26] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "Cusha: Vertex-centric graph processing on GPUs," in *Proc. 23rd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 239–252.
[27] Z. Fu, M. Personick, and B. Thompson, "Mapgraph: A high level API for fast development of high performance graph analytics on GPUs," in *Proc. Workshop Graph Data Manage. Exp. Syst.*, 2014, pp. 1–6.
[28] G. Wang, W. Xie, A. J. Demers, and J. Gehrke, "Asynchronous large-scale graph processing made easy," in *Proc. CIDR*, 2013.
[29] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 505–516.
[30] F. D. McSherry, R. Isaacs, M. A. Isard, and D. G. Murray, "Differential dataflow," U.S. Patent App. 13/468,726, May, 10, 2012.
[31] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: A timely dataflow system," in *Proc. 24th ACM Symp. Operating Syst. Principles*, 2013, pp. 439–455.
[32] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proc. 24th ACM Symp. Operating Syst. Principles*, 2013, pp. 423–438.
[33] Y. Zhao, K. Yoshigoe, M. Xie, S. Zhou, R. Seker, and J. Bian, "Evaluation and analysis of distributed graph-parallel processing frameworks," *J. Cyber Security*, vol. 3, pp. 289–316, 2014.
[34] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke, "How well do graph-processing platforms perform? An empirical performance evaluation and analysis;, in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2013, pp. 395–404.
[35] B. Elser and A. Montresor, "An evaluation study of bigdata frameworks for graph processing," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 60–67.
[36] R. Chen, X. Ding, P. Wang, H. Chen, B. Zang, and H. Guan, "Computation and communication efficient graph processing with distributed immutable view," in *Proc. 23rd Int. Symp, High-Perform. Parallel Distrib. Comput.*, 2014, pp. 215–226.
[37] (2012). The apache hama project [Online]. Available: http://hama.apache.org/
[38] R. Pearce, M. Gokhale, and N. M. Amato, "Faster parallel traversal of scale free graphs at extreme scale with vertex delegates," in *Proc. Int. Conf. High Perform. Comput., Netw. Storage Anal.*, 2014, pp. 549–559.
[39] I. Hoque and I. Gupta, "Lfgraph: Simple and fast distributed graph analytics," in *Proc. 1st ACM SIGOPS Conf. Timely Results Operating Syst.*, 2013, p. 9.



Yue Zhao received the PhD degree in integrated computing from the University of Arkansas at Little Rock, Little Rock, AR in 2015. He is a research fellow in the Epithelial Systems Biology Laboratory, NatiHeart Lung and Blood Institute (NHLBI), National Institutes of Health (NIH). His research interest includes big data analytics, distributed computing, high-performance computing, and bioinformatics.

Kenji Yoshigoe (S'98-M'04-SM'13) received the PhD degree in computer science and engineering from the University of South Florida, Tampa, FL in 2004. He is a professor in the Department of Computer Science and the director of the Computational Research Center at the University of Arkansas at Little Rock. His current research explores the reliability, security, and scalability of various interconnected systems ranging from high-performance computing to resource-constrained wireless sensor networks to dynamically evolving social networks. He is a senior member of the IEEE.

Jiang Bian (M'11) received the MS degree in computer science and the PhD degree in integrated computing from the University of Arkansas at Little Rock, Little Rock, AR in 2007 and 2010, respectively. He is an assistant professor of biomedical informatics in the Department of Health Outcomes and Policy at the University of Florida. His current research interests focus on complex networks, social media, natural language process, machine learning, and data privacy. In particular, his research exploits big data-driven approaches to reveal patterns in massive heterogeneous datasets and make health relevant predictions. He is a member of the IEEE.



Mengjun Xie (S'08-M'10) received the PhD degree in computer science from the College of William and Mary, Williamsburg, VA in 2009. He is an assistant professor in the Department of Computer Science at the University of Arkansas at Little Rock. His research interests lie in the areas of security, mobile computing, network systems, and computer education. He is a member of the IEEE.

Zhe Xue is currently working toward the PhD degree from the School of Preclinical Medicine, Beijing University of Chinese Medicine since September 2014. Her major is diagnostics of Chinese medicine. Currently, she is enrolling in a joint training PhD student program in the National Institute of Neurological Disorders and Stroke (NINDS), National Institutes of Health (NIH). Her research interest includes the standardization and modernization of traditional Chinese medicine diagnosis, acupuncture and bioinformatics.

Yong Feng received the PhD degree in management science and engineering from Northeastern University, Shenyang, China in 2007. He is a professor in the Department of Information Management and Information System, School of Information, Liaoning University. His research interest includes big data analytics, data mining, personalized recommendation, and business intelligence.