J Supercomput (2020) 76:1850–1879
https://doi.org/10.1007/s11227-018-2359-9

L-PowerGraph: a lightweight distributed graph-parallel communication mechanism

Yue Zhao[1] · Kenji Yoshigoe[2] · Mengjun Xie[2] · Jiang Bian[3] · Ke Xiong[4]

Published online: 9 April 2018

Abstract In order to process complex and large-scale graph data, numerous distributed graph-parallel computing platforms have been proposed. PowerGraph is an excellent representative of them. It has exhibited better performance, such as faster graph-processing rate and higher scalability, than others. However, like in other distributed graph computing systems, unnecessary and excessive communications among computing nodes in PowerGraph not only aggravate the network I/O workload of the underlying computing hardware systems but may also cause a decrease in runtime performance. In this paper, we propose and implement a mechanism called L-PowerGraph, which reduces the communication overhead in PowerGraph. First, L-PowerGraph identifies and eliminates the avoidable communications in PowerGraph. Second, in order to further reduce the required communications L-PowerGraph proposes an edge direction-aware master appointment strategy, in which L-PowerGraph appoints the replica with both incoming and outgoing edges as master. Third, L-PowerGraph proposes an edge direction-aware graph partition strategy, which optimally isolates the outgoing edges from the incoming edges of a vertex during the graph partition process. We have conducted extensive experiments using real-world datasets, and our results verified the effectiveness of the proposed mechanism. For example, compared with PowerGraph under Random partition scenario L-PowerGraph can not only reduce up to 30.5% of the communication overhead

✉ Yue Zhao
yxzhao0@gmail.com

1 Epithelial Systems Biology Laboratory, National Heart Lung and Blood Institute (NHLBI), National Institutes of Health (NIH), Bethesda, MD, USA

2 Department of Computer Science, University of Arkansas at Little Rock, Little Rock, AR, USA

3 Health Outcomes and Policy, College of Medicine, University of Florida, Gainesville, FL, USA

4 School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China

but also cut up to 20.3% of the runtime for PageRank algorithm while processing Livejournal dataset. The performance improvement achieved by L-PowerGraph over our precursor work, LightGraph, which only reduces the synchronizing communication overhead, is also verified by our experimental results.

Keywords 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 [1–8], which process large-scale graph under the shared-memory multiprocessor computing environment, and the distributed graph-parallel computing systems [9–20], 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. PowerGraph is one of the most advanced and popular distributed graph-parallel computing systems. However, like in other distributed graph-parallel computing systems, in PowerGraph, 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 efficiency of the computing system and the overall runtime of jobs.

In this paper, we propose a lightweight distributed graph-parallel communication mechanism, L-PowerGraph, and implement it upon PowerGraph. In particular, compared with PowerGraph the main contributions of L-PowerGraph are:

a. L-PowerGraph refers to the edge direction information of each active vertex replica to identify and eliminate the redundant communications in PowerGraph.
b. In order to further reduce the required communications L-PowerGraph proposes an edge direction-aware master appointment strategy, in which a vertex replica with both incoming and outgoing edges has priority to be appointed as master.
c. L-PowerGraph proposes an edge direction-aware graph partition strategy to further reduce the required communications in PowerGraph. This strategy optimally separates the outgoing edges from the incoming edges of a vertex in a graph placement.

Compared with the paper: LightGraph [21], which is our precursor work of L-PowerGraph, L-PowerGraph makes several main additional contributions: (a) Not only reducing the communication overhead in the synchronization phase, L-PowerGraph also diminishes the communication overhead happening in the Gather phase of PowerGraph. (b) L-PowerGraph appoints the vertex replica with both incoming and outgoing edges as master to guarantee the volume of required communications minimal. (c) This paper conducts the quantitative comparison analysis on the communication volume. (d) This paper carries out more extensive experiments to evaluate the proposed solution. More real-world benchmarking data sets are used and more comparison experiments are performed. Also, comparison between LightGraph and L-PowerGraph is conducted.

The rest of the paper is organized as follows. Section 2 introduces the background of this work. The challenges of communication and our countermeasures are presented in Sect. 3. Section 4 details the design and results of the experiment. Section 5 introduces the related work. Section 6 concludes this paper.

2 Background

In a graph-parallel abstraction, the data to process are 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, $v \in V$. In distributed graph-parallel abstractions, the vertex-program instances can communicate with each other by exchanging node-to-node messages. In the literatures, much attention has been paid to reduce the communication overhead in distributed computing systems. This work tries to reduce the communication overhead for PowerGraph specially. Thus, in the following section, we give a detailed introduction of PowerGraph platform.

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].

3 Lightweight communication mechanism for PowerGraph

3.1 Challenges of communication in PowerGraph

Like other distributed graph-parallel computing systems, to process a large-scale graph, PowerGraph needs to partition a graph into smaller sub-graphs and distribute the sub-graphs among the computing nodes. As mentioned above, PowerGraph adopts a vertex-cut strategy to conduct the partition. Nevertheless, 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 can traverse the sub-graphs placed on different machines via the communications between the master and the mirrors. In PowerGraph, communications mainly happen in the Gather phase and the synchronization phase. The volume of communications happening elsewhere is negligible compared to those happening in these two phases. In particular, after Gather function, which runs locally on each machine, 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. The overall communication volume is proportional to the number of mirrors. For complex and large-scale graphs, the number of mirrors may quickly expand with more computing machines used. The high communication overhead limits not only the performance but also the scalability of the system.

3.2 Countermeasure analysis

PowerGraph blindly conducts Gather function execution, partial sum transmission, and synchronization for all mirrors. However, in some graph algorithms, for example, PageRank [22], for some special mirrors these operations are actually unnecessary.

Example 3.1 (PageRank Algorithm) The PageRank score of a node is the long-term probability that a random web surfer is at that node at a particular time step. The computation of the PageRank score of a webpage v is an iterative process where the PageRank algorithm recursively computes the rank, R_v, considering the scores of web pages (noted as u) that are connected to v, defined as:

$$R(v) = (1-\alpha) \sum_{u\ links\ to\ v} w_{u,v} \times R(u) + \frac{\alpha}{n} \tag{1}$$

where α is the damping factor (see [15,22] for a detailed description of PageRank).

In particular, during the graph computing, some mirrors actually have no contribution on the data change on master. On the other hand, data on some mirrors will never be accessed by any vertex program instances in the future computation. Thus, the communications for partial sum transmission or synchronization of these mirrors can actually be avoided.

We found that there is a kind of graph computing algorithms, in which the direction of data access on an edge is always consistent with the edge's direction. In particular, in a directed edge, the data on the target vertex will never be accessed by this edge or the source vertex during computation. Besides PageRank [22] shortest path algorithm, HITS [23] and SALSA [24] all fall into this category. We define this category of applications/algorithms as PageRank-like application/algorithm. In these algorithms, in a distributed graph, a vertex replica with no incoming edge has no contribution to the data change of its original vertex, and the data on a vertex replica with no outgoing edge will never be accessed by any other edges or vertices in the future computation. Thus, for the former replicas there is no need to calculate and transmit their partial sum. For the later replicas, there is no need to synchronize them with the new data on the master vertex.

Definition 3.1 (*PageRank-like Algorithm*) An algorithm is a PageRank-like algorithm if

$$\forall\, edge(u \rightarrow v) \in E \tag{2}$$

The computation happens on u and $edge(u \rightarrow v)$ are subject to

$$D_u = f(args[0], args[1], \ldots, args[i]) \tag{3}$$

and

$$D_{u\rightarrow v} = f(args[0]', args[1]', \ldots, args[j]') \tag{4}$$

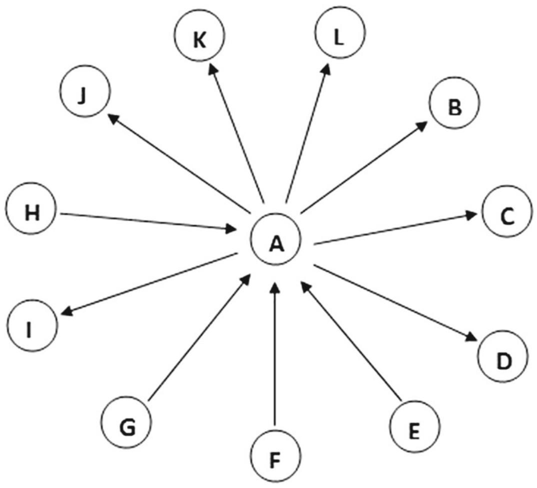
in which

$$D_v \notin \Big(\big[args[0], args[1], \ldots, args[i]\big] \cup \big[args[0]', args[1]', \ldots, args[j]'\big]\Big) \tag{5}$$

where E is the set of overall edges; D_v denotes the data associated with vertex v; $D_{u\rightarrow v}$ denotes the data associated with $edge(u \rightarrow v)$ and f is the update function of vertex v or $edge(u \rightarrow v)$.

In order to demonstrate our observation, we take the execution scenario of PageRank in PowerGraph for instance. Figure 1 shows a partial input sample graph. Figure 2 shows a possible 5-way vertex-cut partition of the graph under PowerGraph's partition strategy [16]. Let us assume that we are computing the PageRank score of vertex A, and the replica of vertex A located on machine 2 is randomly nominated as master. In PowerGraph, the Gather function is firstly executed locally on each machine to calculate the partial PageRank scores of vertex A. And then the partial sums are sent from each mirror to the master. Then, the master launches the Apply function to compute the total sum of these partial sums, and then sends the updated PageRank score of A back to all its mirrors. Finally, on each mirror the Scatter function is launched in parallel to write the new PageRank score for the future computation. However, as shown in Fig. 2, the mirror of vertex A located on machine 0 has no incoming edges. This indicates that the local graph on machine 0 actually has no contribution to the PageRank score of vertex A. Therefore, this mirror does not need to calculate and send

Fig. 1 A partial sample graph



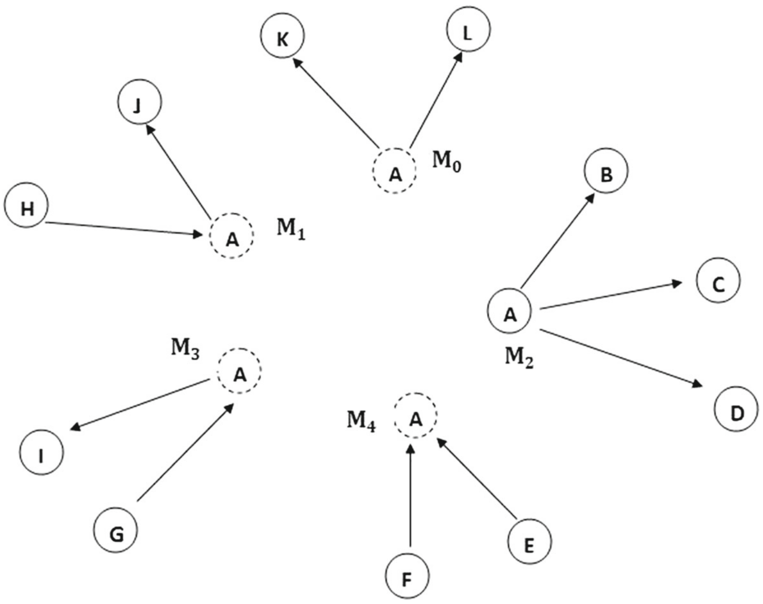


Fig. 2 Graph placement in PowerGraph (*M*: machine; real line circle: master node; dashed line circle: mirror node)

its partial sum, which is 0, to master. On the other hand, the mirror of vertex *A* located on machine 4 has no outgoing edges, which indicates that the data on this mirror will not be accessed by any other vertices in the local graph on machine 4 in the future computation. Thus, the master of *A* does not need to synchronize this mirror with new data. And also, the Scatter phase on this mirror turns to be meaningless and can be eliminated.

3.3 The L-PowerGraph mechanism

Based on the above observation and analysis, we propose and implement a lightweight distributed graph-parallel communication mechanism, L-PowerGraph. The main contribution of L-PowerGraph is threefold: (1) a streamlined Gather and synchronization process, which eliminates the unnecessary communications in PowerGraph; (2) an edge direction-aware master appointment strategy, in which the vertex replica with both incoming and outgoing edges has priority to be appointed as master; and (3) an edge direction-aware vertex-cut partition strategy, which minimizes the number of mirrors with both incoming and outgoing edges to further reduce the required communication.

3.3.1 Streamlined Gather and synchronization process

As discussed above, in PowerGraph the following is true for PageRank-like algorithms. First, mirrors without incoming edges do not need to calculate and send their Gather partial sums to master. Second mirrors without outgoing edges do not need to be synchronized by master. L-PowerGraph streamlines the Gather and synchronization process by identifying and eliminating all these avoidable communications. Figures 3 and 4 compare the communication patterns in PowerGraph and L-PowerGraph while processing mirrors with no incoming and no outgoing edges, respectively. Figures 5

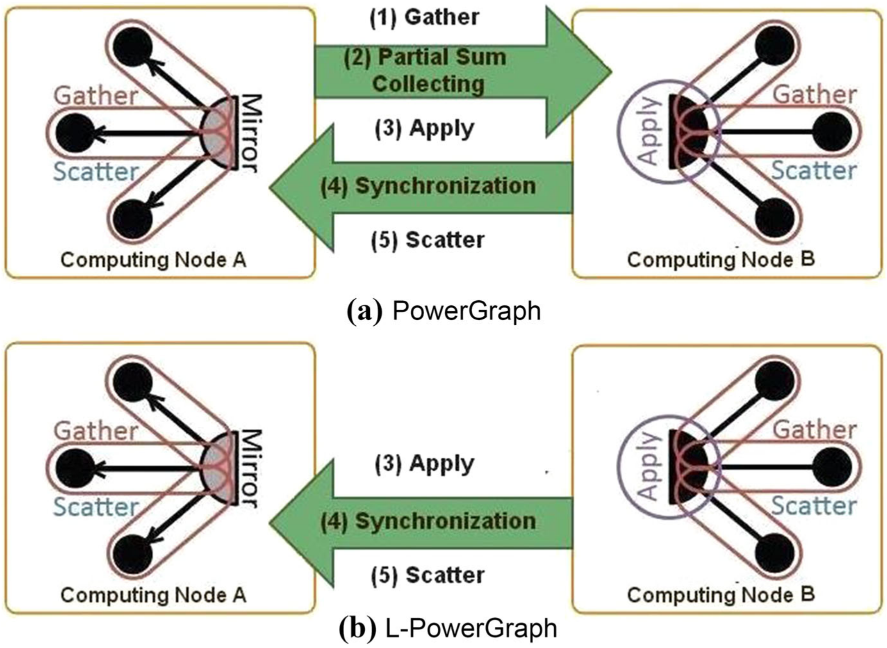


Fig. 3 Communication pattern of PowerGraph and L-PowerGraph when master communicates with a mirror having no incoming edges in PageRank-like algorithm. In L-PowerGraph the Gather and accumulator communication phases of this mirror are eliminated

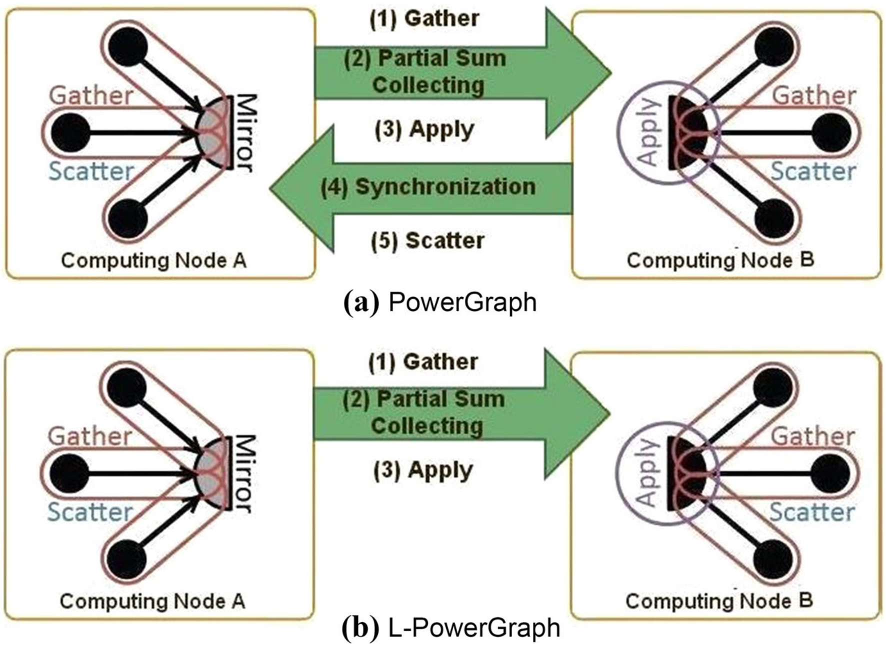


Fig. 4 Communication pattern of PowerGraph and L-PowerGraph when master communicates with a mirror having no outgoing edges in PageRank-like algorithm. In L-PowerGraph the synchronization and Scatter phases of this mirror are eliminated

and 6 demonstrate the communication scenarios while executing PageRank-like algorithms on the distributed graph illustrated by Fig. 2 in PowerGraph and L-PowerGraph, respectively. As compared by Figs. 5 and 6, L-PowerGraph reduces the communication volume by 20%.

3.3.2 Edge direction-aware master appointment

In PowerGraph, one of the original vertex and its replicas is randomly appointed as master, and others are appointed as mirrors. In order to further reduce the required communications, instead of randomly appointing one of all the vertex replicas as master, L-PowerGraph randomly chooses one from the replicas with both incoming and outgoing edges as master. The motivation is both the partial sum transmission and the synchronization are necessary for this kind of replica if they are mirror. Choosing one of them as master will minimize the required communications. Figure 7 illustrates the communications in L-PowerGraph with edge direction-aware master appointment. Compared with Fig. 6 the volume of communications is further reduced.

3.3.3 Edge direction-aware graph partition

In PageRank-like algorithms, the Gather communication of the mirrors without incoming edges and the synchronizing communication of the mirrors without outgoing edges

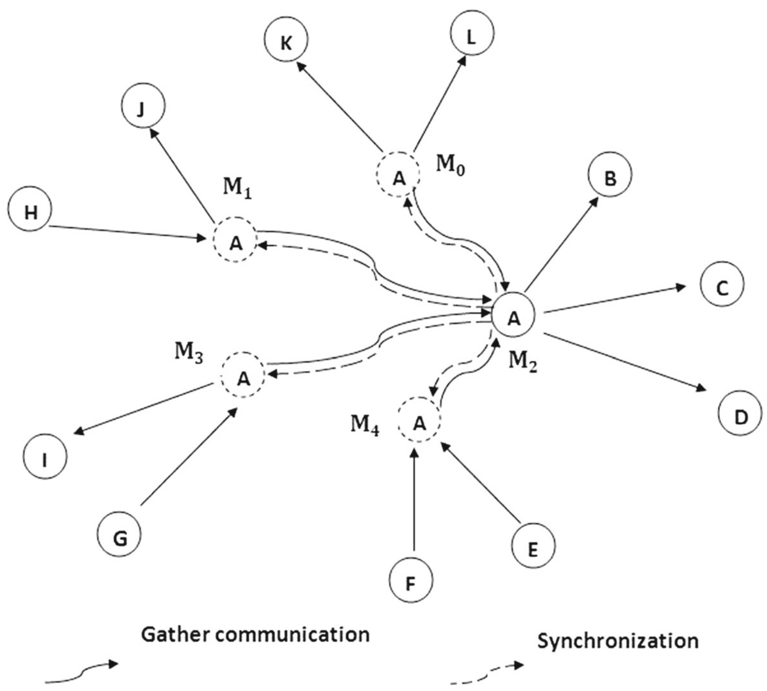


Fig. 5 An illustration of communications in PowerGraph

can be eliminated. Naturally, we can reason, assuming the number of overall mirrors in a distributed graph is same, the less mirrors having both incoming and outgoing edges, the less communications are required. Therefore, to reduce the number of mirrors with both incoming and outgoing edges L-PowerGraph takes the direction of edge as a heuristic parameter in the initial graph partition phase. This graph partition method is called edge direction-aware partition (EDAP) strategy. In particular, for each vertex EDAP places vertex v's incoming/outgoing edges on machines, which do not have v's outgoing/incoming edges yet. By doing so, EDAP optimally isolates the incoming edges from the outgoing edges of a vertex. EDAP inherits other partition mechanisms used by PowerGraph to guarantee good workload balance and low number of vertex replicas. EDAP is detailed in Algorithm 1 (*Note:* $In\_Degree/Out\_Degree(m; v)$ denotes the in-degree/out-degree of vertex v on the local graph located on machine m). EDAP only introduces a heuristic parameter (the direction of edge) to existing partitioning strategies. Thus, for graph $G(V, E)$, upon existing partitioning strategies the additional space complexity induced by EDAP is $O(1)$, and the additional time complexity induced by EDAP is $O(|V|)$.

Figure 8 demonstrates the new 5-way placement of the sample graph illustrated in Fig. 1 under EDAP. Different from that in Fig. 2, the incoming edges of vertex A, edge $(H \rightarrow A)$ and edge $(G \rightarrow A)$, are assigned to the same machine (machine 3), and the outgoing edges of vertex A: edge $(A \rightarrow I)$ and edge $(A \rightarrow J)$, are placed together on machine 1. Consequently, like the mirror on machine 0 the mirror on machine 1 now

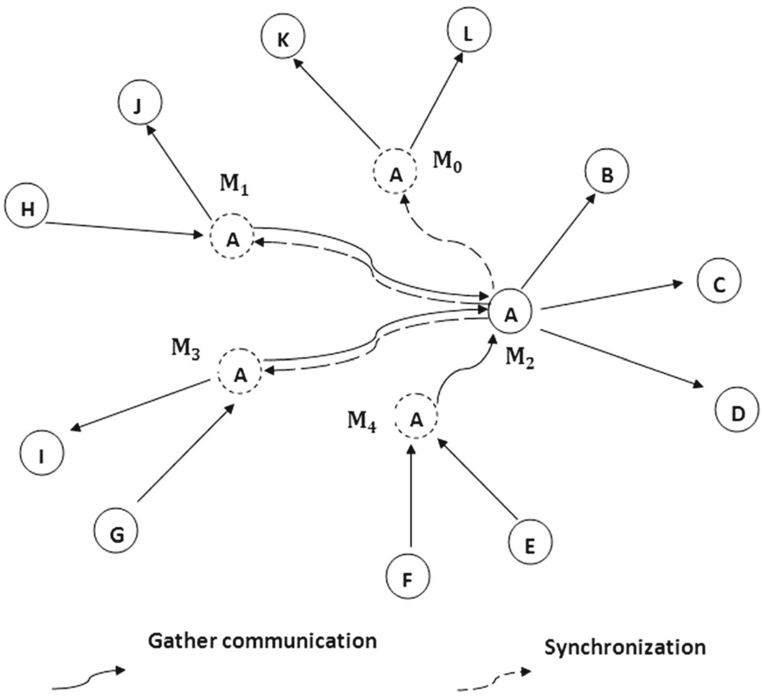


Fig. 6 An illustration of communications in L-PowerGraph

Algorithm 1 Edge direction-aware graph partition

```
1: procedure EDAP $(G(V, E), M)$
2:    define machine set $M' \leftarrow \emptyset$;
3:    for each $v(source, target) \in E$ do
4:    for each machine $m \in M$ do
5:    if $Out\_Degree(m, source) > 0$ or $In\_Degree(m; target) > 0$ then
6:       $M' \leftarrow m$
7:       end for
8:       if $|M'| > 0$ then
9:          Rondom ($M$', $v$)/Oblivious ($M$', $v$)/...... ;
10:       else
11:          Rondom ($M$, $v$)/Oblivious ($M$, $v$)/...... ;
12:       end for
13:       Return;
```

also has no incoming edges. And in addition to the mirror on machine 4, the mirror on machine 3 currently also has no outgoing edge. Under this new placement, the Gather communication of only mirrors on machine 3 and machine 4 are needed. And the master only needs to synchronize the mirrors on machine 0 and machine 1 when the data is changed. Figure 9 shows the corresponding communication scenario under this new placement. When compared with the communication scenarios in Figs. 5 and 6, the volume of communications is further reduced.

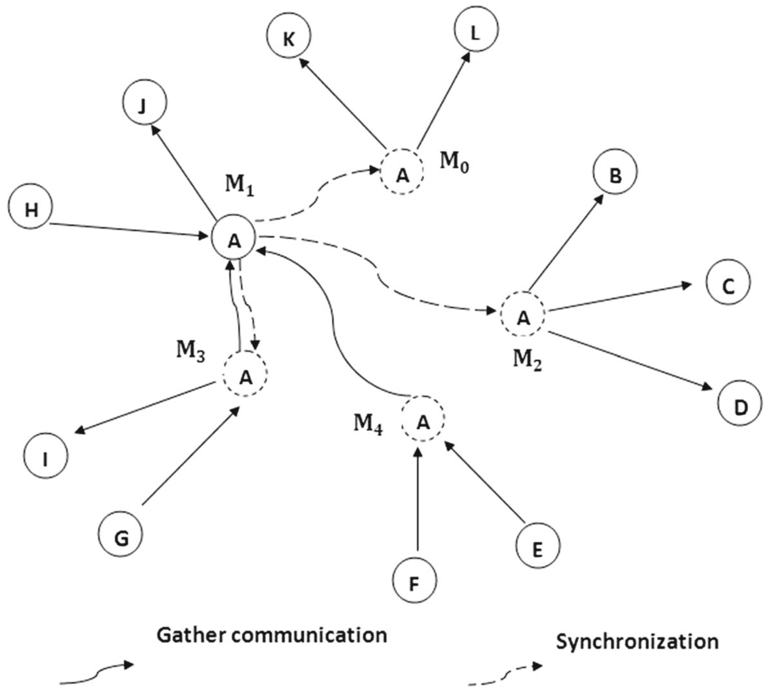


Fig. 7 An illustration of communications in L-PowerGraph with edge direction-aware master appointment

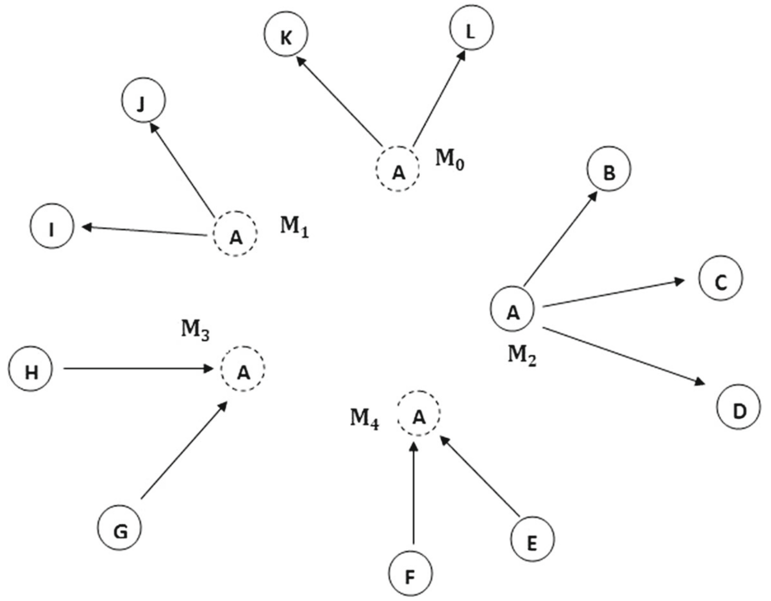


Fig. 8 An example of graph placement using the edge direction-aware partition strategy

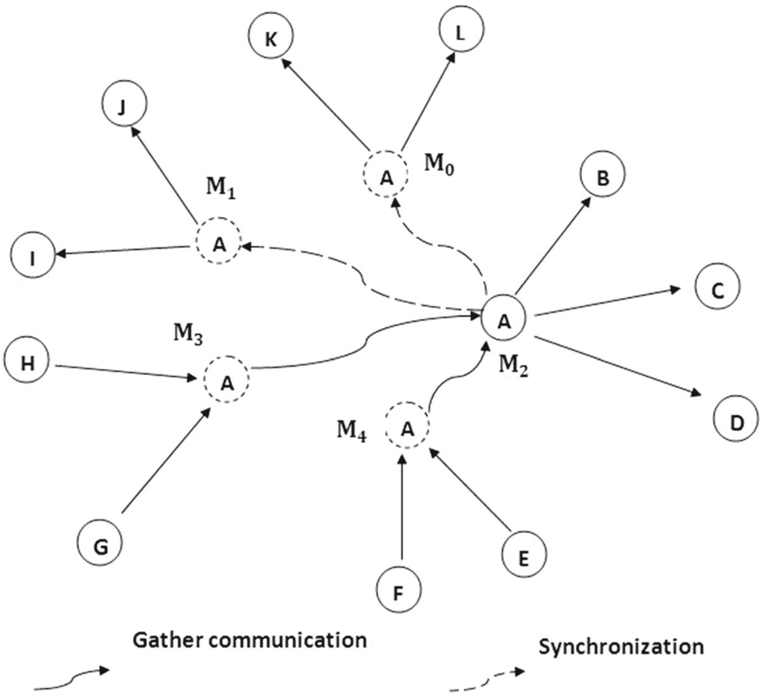


Fig. 9 An example of communications under EDAP-based graph placement in L-PowerGraph

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:

$$\text{Replication factor} = \frac{1}{|V|}\sum_{v\in V}|A(v)| \tag{6}$$

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

$$\arg\min_{j} \mathbb{E}\left[\sum_{v\in V}|A(v)|\middle| A_{e_1}\cdots A_{e_i}, A_{(e_{i+1})} = j\right] \tag{7}$$

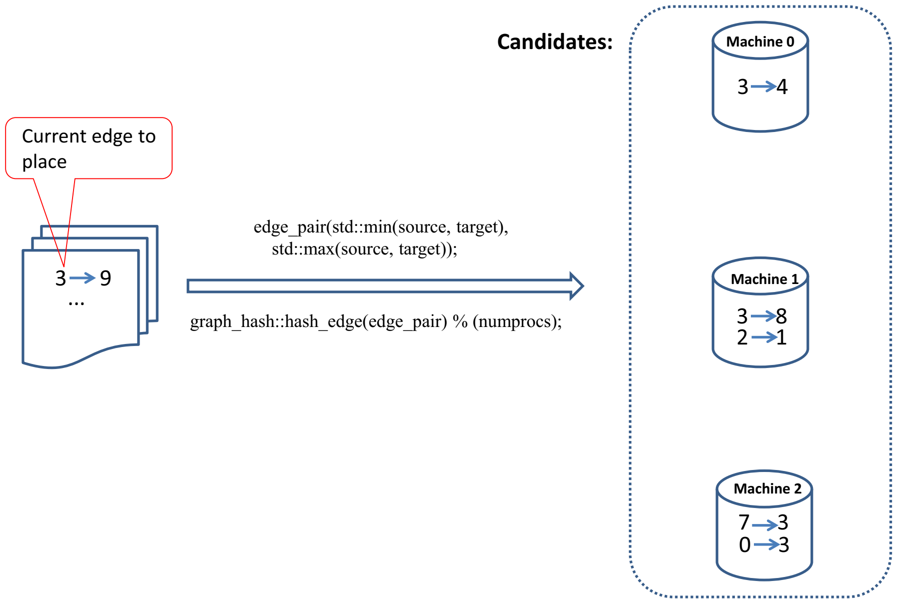

Candidates:
Machine 0
3→4
Current edge to place
3→9
...
edge_pair(std::min(source, target),
std::max(source, target));
graph_hash::hash_edge(edge_pair) % (numprocs);
Machine 1
3→8
2→1
Machine 2
7→3
0→3

Fig. 10 Random edge-placement uses a hash function to randomly distributes edges to machines

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.

In L-PowerGraph, we introduce the direction of edges as a heuristic parameter to the Random and Oblivious partition strategies and create the partition methods: EDAP_Random and EDAP_Oblivious detailed in Algorithm 1, respectively. Figures 10, 11, 12, and 13 illustrate the mentioned graph placement strategies.

3.4 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 L-PowerGraph the majority of overall communications happen in the Gather partial sum transmitting phase and the synchronization phase. Thus, our analysis mainly focuses on this part of communications. Table 1 explains the related notations.

3.4.1 General analysis

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

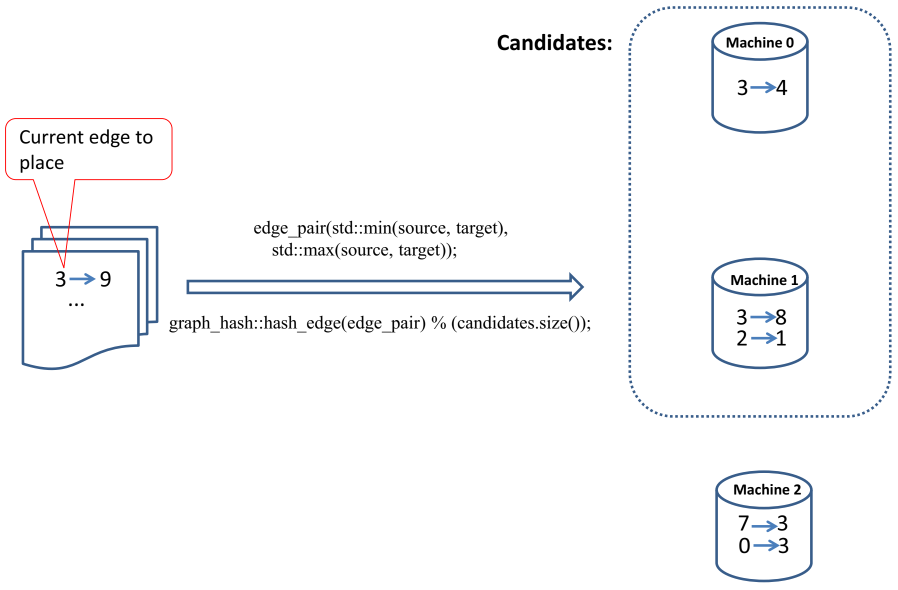

Candidates:
Machine 0
3→4
Machine 1
3→8
2→1
Machine 2
7→3
0→3
Current edge to place
3→9
…
edge_pair(std::min(source, target), std::max(source, target));
graph_hash::hash_edge(edge_pair) % (candidates.size());

Fig. 11 EDAP_Random edge-placement places vertex v's incoming/outgoing edges on machines, which do not have v's outgoing/incoming edges yet

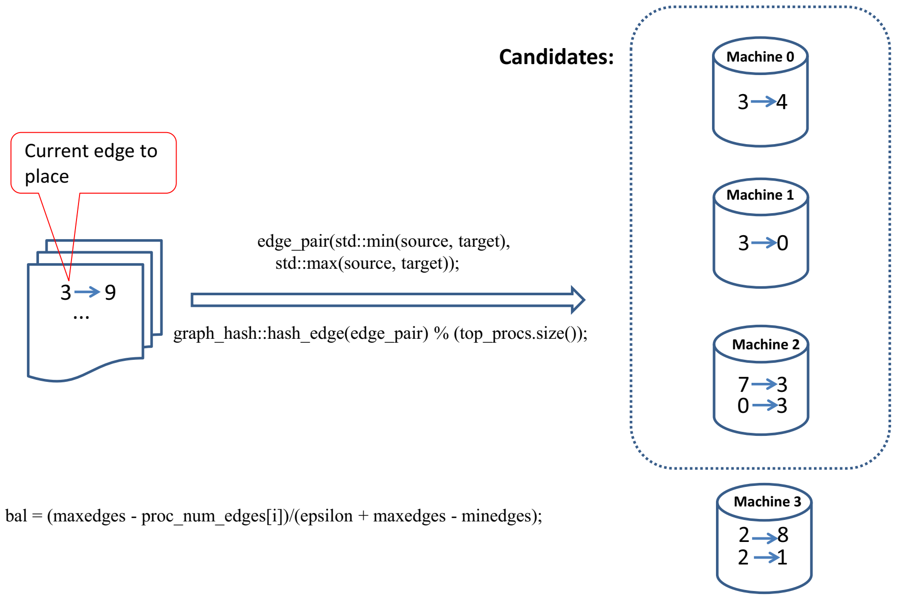

Candidates:
Machine 0
3→4
Machine 1
3→0
Machine 2
7→3
0→3
Machine 3
2→8
2→1
Current edge to place
3→9
…
edge_pair(std::min(source, target), std::max(source, target));
graph_hash::hash_edge(edge_pair) % (top_procs.size());
bal = (maxedges - proc_num_edges[i])/(epsilon + maxedges - minedges);

Fig. 12 Oblivious edge-placement uses greedy heuristic to minimize the expected replication factor

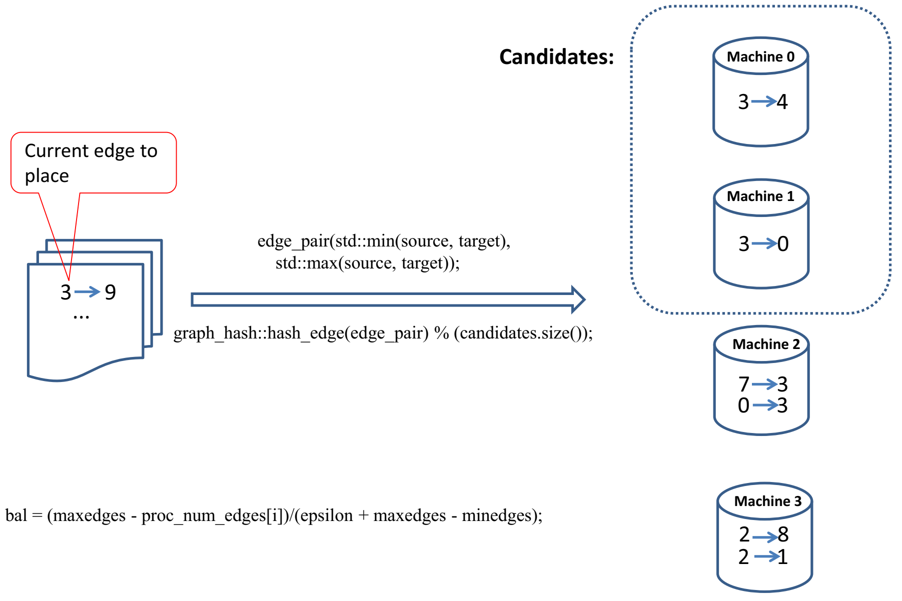

Candidates:
Machine 0
3→4
Machine 1
3→0
Machine 2
7→3
0→3
Machine 3
2→8
2→1
Current edge to place
3→9
...
edge_pair(std::min(source, target), std::max(source, target));
graph_hash::hash_edge(edge_pair) % (candidates.size());
bal = (maxedges - proc_num_edges[i])/(epsilon + maxedges - minedges);

Fig. 13 EDAP_Oblivious placement places vertex v's incoming/outgoing edges on machines, which do not have v's outgoing/incoming edges yet

Table 1 Notations

Symbol	Description
n	The number of iterations in a graph computing job
$\varnothing(v, i)$	The flag indicating whether v is active or not 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
$N\_sync\_comm\_v$	The number of synchronizing messages happening on vertex v
$N\_gather\_comm\_v$	The number of Gather communication messages happening on vertex v
N_total_comm	The number of communication messages happening on all vertices computing
$P\_reduced\_comm$	Percentage of reduced communication messages

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;

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

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 are 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:

$$N\_sync\_comm\_v = \sum_{i=0}^{n-1} \left[\varnothing(v,i) \sum_{j=0}^{2} v\_m_j \right] \tag{8}$$

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:

$$N\_gather\_comm\_v = \sum_{i=0}^{n-1} \left[\varnothing(v,i) \sum_{j=0}^{2} v\_m_j \right] \tag{9}$$

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

$$\begin{aligned} &N\_total\_comm\_v \\ &\quad = N\_sync\_comm\_v + N\_gather\_comm\_v \\ &\quad = \sum_{i=0}^{n-1} \left[\varnothing(v,i) \sum_{j=0}^{2} v\_m_j \right] + \sum_{i=0}^{n-1} \left[\varnothing(v,i) \sum_{j=0}^{2} v\_m_j \right] \\ &\quad = 2\sum_{i=0}^{n-1} \left[\varnothing(v,i) \sum_{j=0}^{2} v\_m_j \right] \end{aligned} \tag{10}$$

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

$$\begin{aligned} N\_total\_comm &= \sum_{i=0}^{|V|-1} N\_total\_comm\_v_i \\ &= 2\sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i,j) \sum_{k=0}^{2} v_i\_m_k \right] \end{aligned} \tag{11}$$

In L-PowerGraph

L-PowerGraph just eliminates the unnecessary communications between mirror and master. Thus, L-PowerGraph does not induce any impact on the mediated computing results of a vertex. Therefore, for each vertex v, v_p_i is the same with that under PowerGraph.

In L-PowerGraph, v's mirror with no outgoing edge does not need to be synchronized by v's master. Thus, the number of synchronizing messages happening on vertex v is:

$$N\_sync\_comm\_v = \sum_{i=0}^{n-1} [\varnothing(v,i)(v\_m_0 + v\_m_1)] \tag{12}$$

In L-PowerGraph, v's mirror with no incoming edge does not need to compute and send its Gather partial sum to v's master. Thus, the number of Gather communication messages happening on vertex v is:

$$N\_gather\_comm\_v = \sum_{i=0}^{n-1} [\varnothing(v,i)(v\_m_0 + v\_m_2)] \tag{13}$$

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

$$\begin{aligned} &N\_total\_comm\_v \\ &\quad = N\_sync\_comm\_v + N\_gather\_comm\_v \\ &\quad = \sum_{i=0}^{n-1} [\varnothing(v,i)(v\_m_0 + v\_m_1)] + \sum_{i=0}^{n-1} [\varnothing(v,i)(v\_m_0 + v\_m_2)] \\ &\quad = \sum_{i=0}^{n-1} [\varnothing(v,i)(2v\_m_0 + v\_m_1 + v\_m_2)] \end{aligned} \tag{14}$$

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

$$\begin{aligned} N\_total\_comm &= \sum_{i=0}^{|V|-1} N\_total\_comm\_v_i \\ &= \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} [\varnothing(v_i,j)(2v_i\_m_0 + v_i\_m_1 + v_i\_m_2)] \\ &= \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i,j) \sum_{k=0}^{2} v_i\_m_k + \varnothing(v_i,j) v_i\_m_0 \right] \end{aligned} \tag{15}$$

Thus, the number of reduced communication messages achieved by L-PowerGraph over PowerGraph is:

$$\begin{aligned} &N\_total\_comm_{Reduced} \\ &\quad = 2 \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i,j) \sum_{k=0}^{2} v_i\_m_k \right] \end{aligned}$$

$$
\begin{aligned}
&- \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=0}^{2} v_i\_m_k + \varnothing(v_i, j) v_i\_m_0 \right] \\
&= \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=0}^{2} v_i\_m_k - \varnothing(v_i, j) v_i\_m_0 \right] \\
&= \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=1}^{2} v_i\_m_k \right]
\end{aligned} \tag{16}
$$

Thus,

$$
\begin{aligned}
&P\_reduced\_comm \\
&= \frac{\sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=1}^{2} v_i\_m_k \right]}{2 \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=0}^{2} v_i\_m_k \right]} \times 100
\end{aligned} \tag{17}
$$

3.4.2 Optimal analysis

The analysis above indicates that $P\_reduced\_comm$ will be larger when $\sum_{i=0}^{|V|-1} v_i\_m_1$ and $\sum_{i=0}^{|V|-1} v_i\_m_2$ are larger or $\sum_{i=0}^{|V|-1} v_i\_m_0$ is smaller.

In the optimal case $\sum_{i=0}^{|V|-1} v_i\_m_0 = 0$, namely in this distributed graph all incoming and outgoing edges of each vertex are separated. Then,

$$
\begin{aligned}
&P\_reduced\_comm \\
&= \frac{\sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=1}^{2} v_i\_m_k \right]}{2 \sum_{i=0}^{|V|-1} \sum_{j=0}^{n-1} \left[\varnothing(v_i, j) \sum_{k=1}^{2} v_i\_m_k \right]} \times 100 \\
&= 50
\end{aligned} \tag{18}
$$

Thus, in the optimal case, compared with PowerGraph, 50% of the communication overhead can be reduced by L-PowerGraph.

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

The proposed approaches can be applied to other vertex-centric distributed graph computing systems, such as GraphX [25], PowerLyra [17], GrapH [26], because all key actions required by our approaches are feasible and implementable in a system that iteratively executes a user-defined program over vertices of a graph. These actions are (1) identifying the direction of data transfer on each vertex according to the application characteristics; (2) identifying and eliminating the unnecessary communications happening between vertex replicas according to the data transfer direction; (3) introducing the direction of edge as a heuristic parameter to the partition strategies used in the system.

Table 2 Comparing key characteristics of L-PowerGraph with existing graph-parallel platforms

Metrics	Pregel	Distributed GraphLab	PowerGraph	L-PowerGraph
Volume of communications	$\propto$ # of ghosts	$\propto$ # of ghosts	$\propto$ # of mirrors	$\propto$ # of partial mirrors
Graph partitioning method	Edge-cut	Edge-cut	Vertex-cut	Edge direction-aware vertex-cut
Computation model	Synchronous	Synchronous & Asynchronous	Synchronous & Asynchronous	Synchronous & Asynchronous

4 Experimental evaluation

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

4.1 Experiment environment

Our experiments were conducted on a 65-node (528 cores) 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 connected via 10 GigE Ethernet. Each computing node has 16 GB of RAM and 2 quad-core Intel Xeon 2.66 GHz CPUs. 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.

4.2 Benchmarking application and dataset

We selected PageRank and SSSP (shortest path algorithm) as benchmarking applications and processed three data sets listed in Table 3. 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. L-PowerGraph mechanism is only applicable for directed graphs. Thus, all graphs selected are directed.

4.3 Experiment design and results

We ran PageRank and SSSP, and compared measured benchmarking metrics in L-PowerGraph with those 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.

Figure 14 compares the ingress time of Twitter under different partitioning strategies. Compared to the volume of communications eliminated and several times of job runtime saved by EDAP (see Figs. 17, 19) the cost of EDAP is quite worth.

Table 3 Summary of data sets

Name	Description	# of vertices	# of edges	Graph density ($\times 10^{-5}$)	Average degree
soc-LiveJournal [27]	LiveJournal friendship social network	4,847,571	68,993,773	0.59	14
Twitter [28]	Social website	41,652,230	1,468,365,182	0.08	35
BFS1 [29]	Facebook social networks	61,876,615	336,776,269	0.02	5

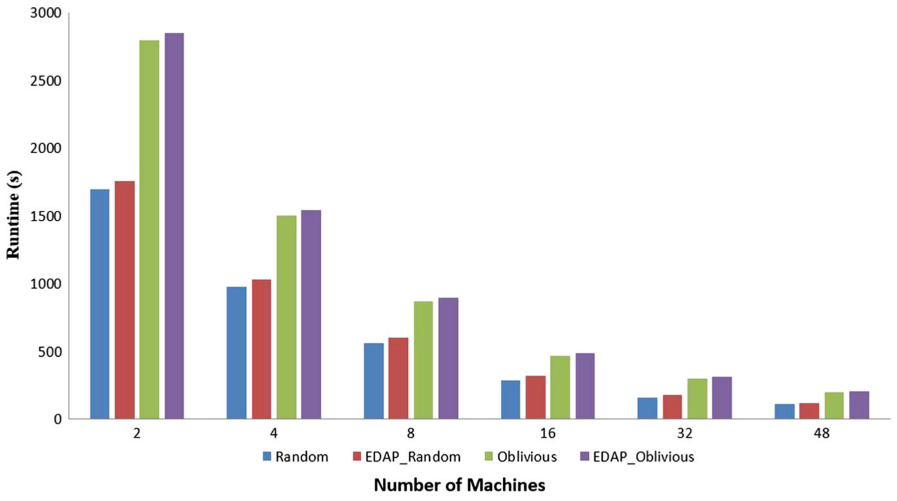


Fig. 14 Ingress time of Twitter

We ran PageRank on sov-LiveJournal data set. And we measured the numbers of each kind of mirrors and the numbers of various communication messages happening on the graph computing. In particular, we measured the Gather partial sum communications and the synchronizing communications, which dominates the communications in the whole job in PowerGraph and L-PowerGraph.

Figures 15 and 16 plot the results of the tests under synchronous mode using 16 machines. In Fig. 15 the numbers of mirrors needing to be synchronized under L-PowerGraph are actually the numbers of mirrors with outgoing edge. As Fig. 15 shows, 86.8 and 89.6% of overall mirrors have outgoing edge under Random and Oblivious placement, respectively. And L-PowerGraph 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% by L-PowerGraph (Random) and L-PowerGraph (Oblivious) over PowerGraph (Random) and PowerGraph (Oblivious), respectively. EDAP partition strategy tries to further isolate the incoming and outgoing edge for each vertex. And as expected, under EDAP_Random and EDAP_Oblivious the percentages of mirrors with outgoing edge

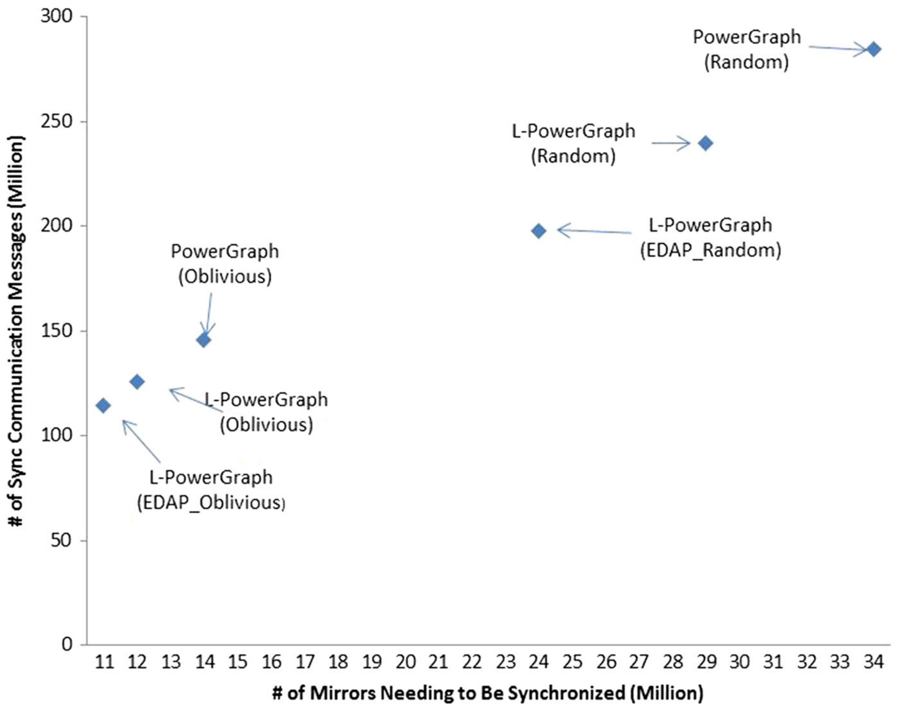


Fig. 15 Number of synchronizing communication messages versus the number of mirrors needing to be synchronized

are reduced to 71.5 and 82.8%, respectively. Consequently, the numbers of synchronizing communication messages are reduced by 26.4 and 16.5% by L-PowerGraph (EDAP_Random) and L-PowerGraph (EDAP_Oblivious) over PowerGraph (Random) and PowerGraph (Oblivious), respectively. Figure 16 demonstrates the similar effectiveness of L-PowerGraph and EDAP in reducing the number of mirrors needing to launch Gather communication and the number of Gather communication messages.

Figure 17 shows the volume of communications in PowerGraph and L-PowerGraph processing the Twitter data set under asynchronous and synchronous computation mode, respectively. Figure 18 shows the corresponding results on LiveJournal data set. As expected, L-PowerGraph and its EDAP strategy can significantly reduce the communication overhead for PageRank. For example, for LiveJournal dataset L-PowerGraph (EDAP_Random) can consistently reduce at least 17.6% communications when the number of machines is larger than 2 under synchronous mode over PowerGraph (Random); Under asynchronous mode, over PowerGraph (Random) the maximal communication reduction achieved by L-PowerGraph (EDAP_Random) is 32.6% while processing Twitter data set. Moreover, as the number of machines increases, the volume of communications reduced by L-PowerGraph also increases.

Figures 19 and 20 show the PageRank runtime on Twitter and LiveJournal data set, respectively. By reducing the volume of communications, L-PowerGraph shortens the runtime of PageRank under both synchronous and asynchronous modes. Moreover,

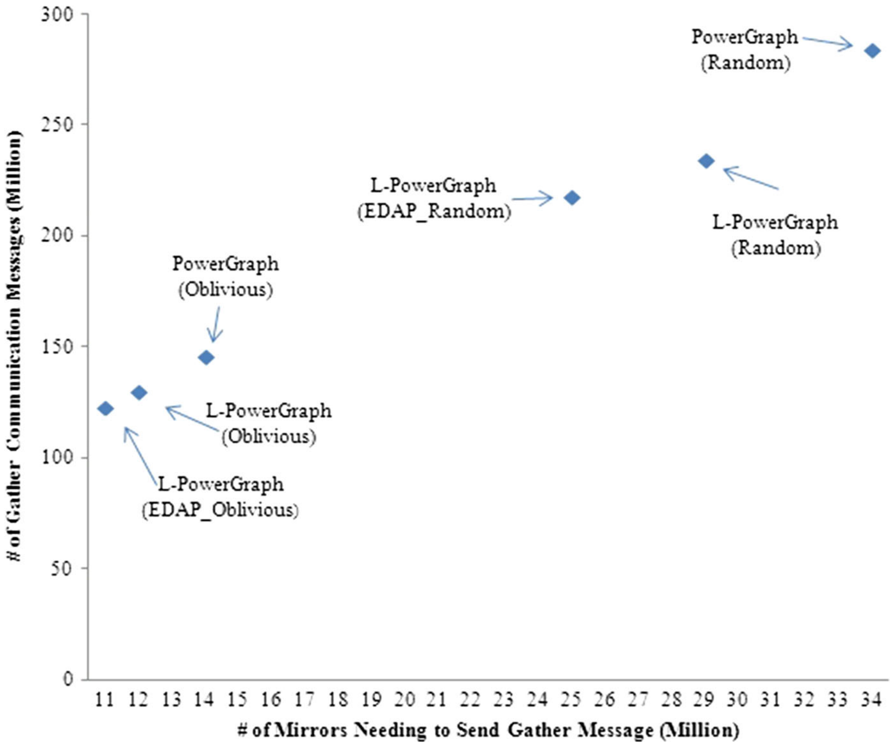


Fig. 16 Number of Gather communication messages versus the number of mirrors needing to send Gather message

by using EDAP strategy L-PowerGraph further accelerates the execution of PageRank. For example, for Twitter data set, over asynchronous PowerGraph (Random) the maximal runtime reduction achieved by asynchronous L-PowerGraph (EDAP_ Random) is 19.7%. Furthermore, L-PowerGraph shows consistent performance gains as the number of machines used increases. For instance, among all cases presented L-PowerGraph (EDAP_ Oblivious) can shorten at least 13.2% runtime over PowerGraph (Oblivious) on LiveJournal dataset under synchronous mode.

Our experiment results of SSSP also verify the effectiveness of L-PowerGraph and its EDAP partition strategy. We took the results processing BFS1 data set under synchronous mode as example and presented them in Fig. 21. As the figure shows, up to 29.1 and 15.2% improvements in volume of communication and runtime are achieved by L-PowerGraph with EDAP strategy, respectively. And, better effectiveness is achieved with the number of machines used increasing.

Our experiment results demonstrate that by reducing the communication overhead L-PowerGraph also exhibits a better scalability than PowerGraph. Take an instance, Fig. 22 demonstrates the runtime speedup for PageRank under various processing scenarios. The speedup of PowerGraph is low. Especially, that under Random placement is the worst, which is less than 1.5 with 48 machines used. The reason for this poor speedup is that Random placement creates a large number of replicas of vertices, which

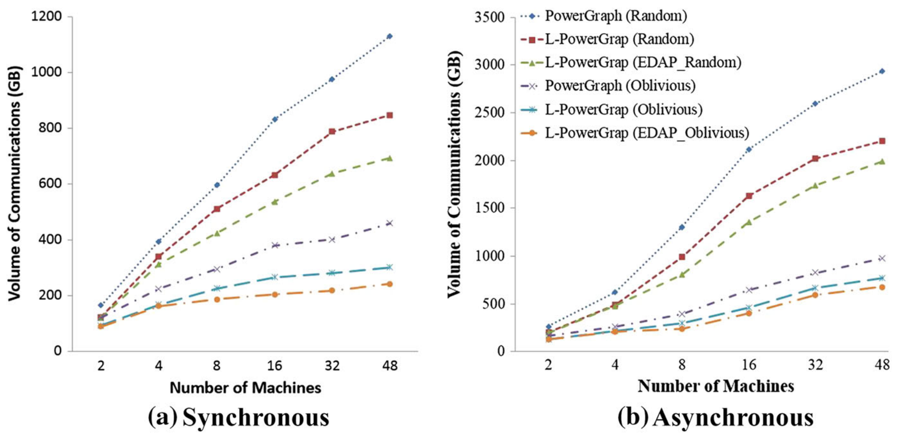


Fig. 17 Comparing the volume of communications for Twitter between L-PowerGraph and PowerGraph

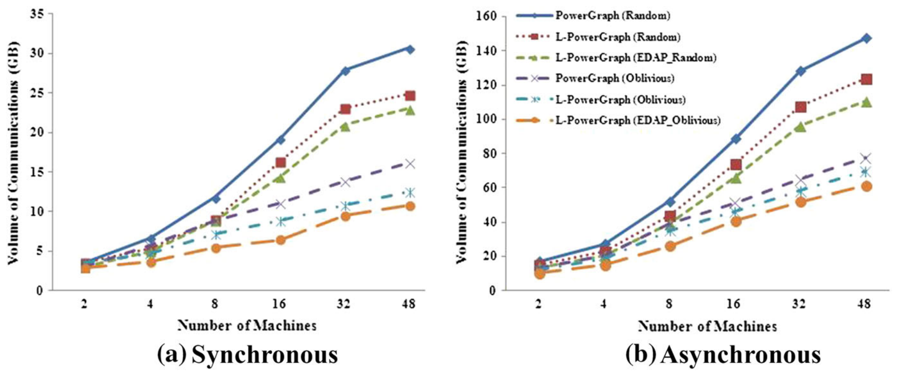


Fig. 18 Comparing the volume of communications for LiveJournal between L-PowerGraph and PowerGraph

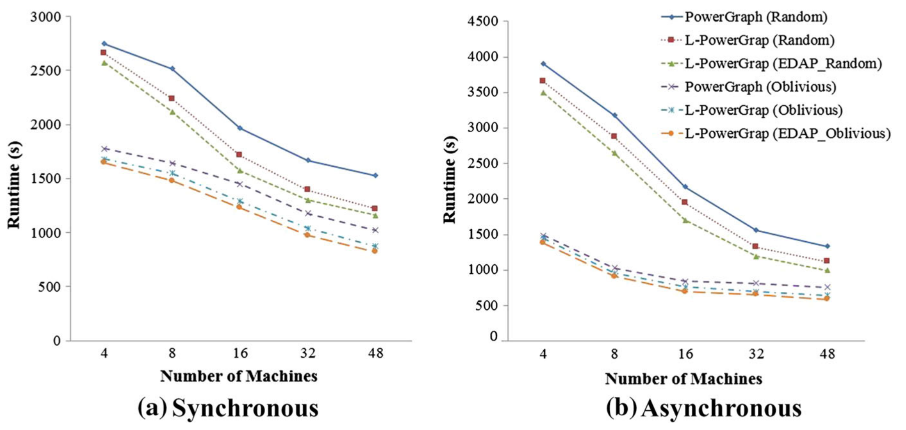


Fig. 19 Comparing the PageRank runtime for Twitter between L-PowerGraph and PowerGraph

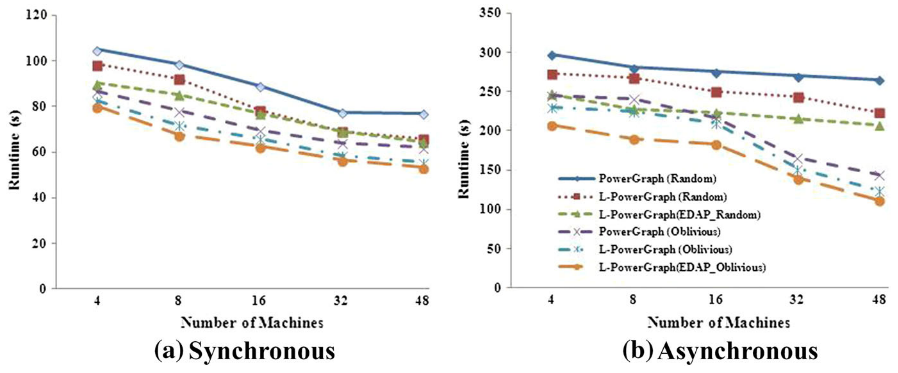


Fig. 20 Comparing the PageRank runtime for LiveJournal between L-PowerGraph and PowerGraph

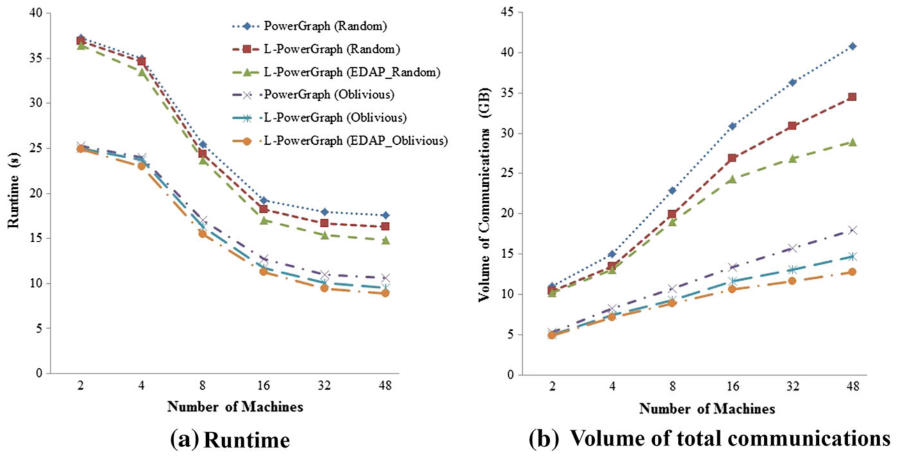


Fig. 21 Comparing the runtime and volume of communications for SSSP running on BFS1 dataset under synchronous computation mode between L-PowerGraph and PowerGraph

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, L-PowerGraph lightens the communication overhead during the graph computing. By this optimization, the speedup is increased up to 2.6 [L-PowerGraph (Oblivious)] with 48 machines used.

Although, for PageRank, 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 PageRank 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.

Our experiment results also demonstrate the performance improvement induced by L-PowerGraph over our precursor work, LightGraph. Figure 23 shows the vol-

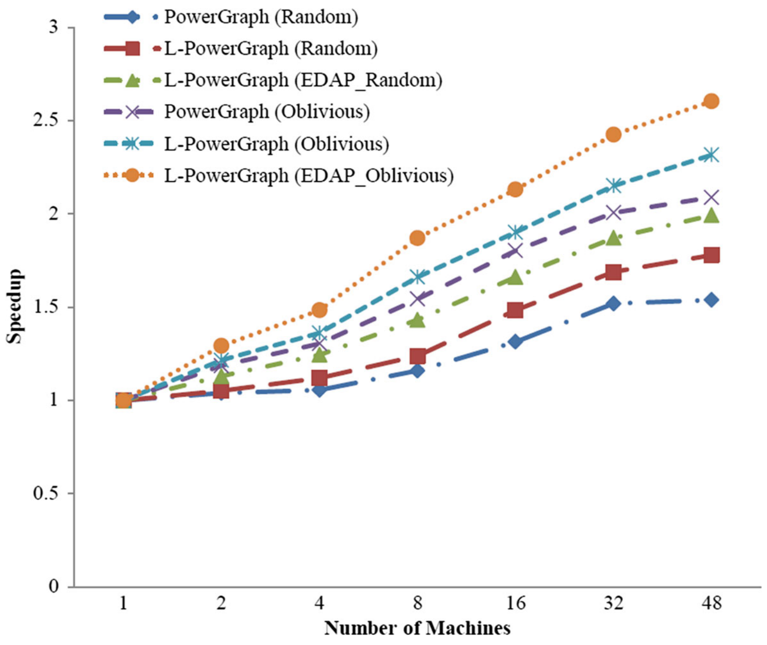


Fig. 22 PageRank runtime speedup in synchronous mode on LiveJournal data set

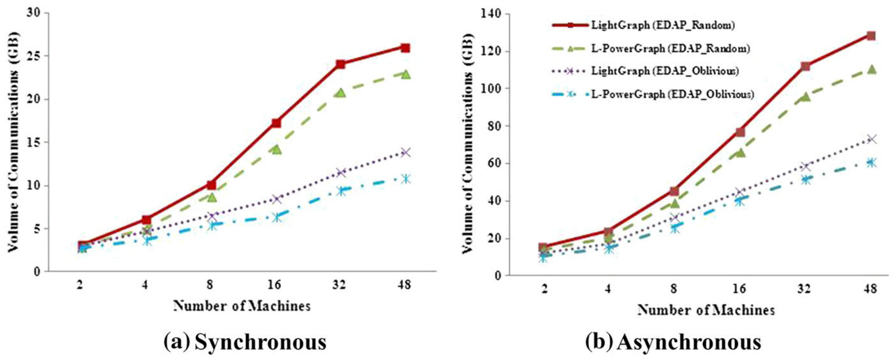


Fig. 23 Comparing the volume of communications for LiveJournal between L-PowerGraph and LightGraph

ume of communications of PageRank in LightGraph and L-PowerGraph processing the LiveJournal data set under asynchronous and synchronous computation mode, respectively. As the figure demonstrate, compared with LightGraph the volumes of communications in PageRank execution under both modes in L-PowerGraph are further reduced. For example, L-PowerGraph (EDAP_Oblivious) can consistently reduce at least 11.8% communications when the number of machines is larger than 2 under synchronous mode over LightGraph (EDAP_Oblivious); moreover, as the number of machines increases, the volume of communications reduced by L-PowerGraph also increases.

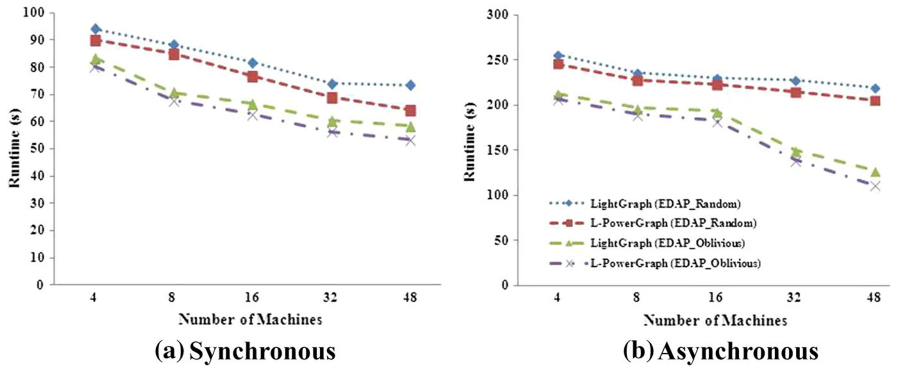


Fig. 24 Comparing the PageRank runtime for LiveJournal between L-PowerGraph and LightGraph

Figure 24 compare the PageRank runtime in LightGraph and L-PowerGraph on LiveJournal data set. As shown in the figure, compared with LightGraph L-PowerGraph further shortens the runtime of PageRank under both synchronous and asynchronous modes. For example, over synchronous LightGraph (EDAP_Random) the maximal runtime reduction achieved by synchronous L-PowerGraph (EDAP_ Random) is 13.1%. Furthermore, L-PowerGraph shows better performance gains as the number of machines used increases.

5 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 similar to Pregel such as GPS [11], Giraph [13]. PGX [30] developed by Oracle can process large-scale graphs under either single-machine shared-memory or distributed computing model. Gregor et al. propose the parallel BGL [9], a generic C++ library for distributed graph computation, and apply the paradigm of generic programming to the domain of graph computations. Kineograph [12] uses a stream of incoming data to construct a continuously changing graph, which captures the relationship that exists in the data feed. Stutz et al. propose the Signal-Collect [14] framework to concisely specify and execute a number of computations that are typical for Semantic Web. Naiad [31,32] is able to conduct incremental iterative computation. However, it adopts traditional synchronous check pointing for fault tolerance and cannot respond to stragglers [33]. 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 [34–36]. Like LightGraph [21], MOCgraph [37] supports both synchronous and asynchronous executions. Nguyen et al. [38] and Roy et al. [7] try to demonstrate better runtime performance can be achieved by some other graph computing systems, such as Galois [39], Ligra [8], and X-stream [7], than PowerGraph. However, these systems are all single-machine graph computing systems and the evaluation and comparison are conducted only under shared-memory multiprocessor computing environment. Note that PowerGraph is designed and developed to compute large-scale graph data in distributed computing environment. Cyclops [40] is also a vertex-oriented graph-parallel framework. However, compared with PowerGraph (written in C++) its java implementation based on Hama [41] drags its runtime performance down. Mayer et al. [26] uses vertex-cut graph partitioning that considers both diverse vertex traffic and heterogeneous network costs. However, its partitioning method does not take the application characteristics into account like EDAP partitioning strategy proposed in the work. Shi et al. [42] proposes a light-weight processing framework called Frog with a hybrid coloring model. However, Frog only supports asynchronous computing model. In general, asynchronous computing model introduces much more communications than synchronous computing model. On the other hand, Xiao et al. [43] and Yan et al. [44] only support synchronous computing model not like L-PowerGraph that can support both synchronous and asynchronous computing model.

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 super-steps. 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]. Pearce et al. [45] 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. LightGraph [21], the precursor of this work, tries to identify and eliminate the unnecessary communications in distributed graph-parallel platforms. However, it just looks at the synchronous communication.

6 Conclusions and future work

Although PowerGraph can provide high computational capabilities and scalability for large-scale graph-structured computation, it often suffers from heavy communication overhead. In this paper, targeting to PageRank-like applications, we proposed a lightweight communication mechanism, L-PowerGraph to reduce the communication overhead and accelerate the job execution. Our extensive experiment results on real-world network data sets have demonstrated that compared with PowerGraph L-PowerGraph can not only reduce the volume of communications significantly but also improve the runtime performance of PageRank-like applications.

Distributed big-data processing systems make it feasible to perform computations on large volumes of data with high complexity. However, the communication overheads 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.

The approaches proposed in this work are suitable for supporting multiple vertex-centric distributed graph systems not only the PowerGraph. Implementing and evaluating our approaches in other vertex-centric distributed graph systems under various network settings (e.g., 1 GigE, 10 GigE, InfiniBand) are our future work.

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

References

1. Kyrola A, Blelloch G, Guestrin C (2012) Graphchi: large-scale graph computation on just a PC. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12. USENIX Association, Berkeley, CA, USA, pp 31–46. URL http://dl.acm.org/citation.cfm?id=2387880.2387884
2. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM (2010) Graphlab: a new parallel framework for machine learning. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), Catalina Island, California
3. Han W-S, Lee S, Park K, Lee J-H, Kim M-S, Kim J, Yu H (2013) Turbograph: a fast parallel graph engine handling billion-scale graphs in a single PC. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13. ACM, New York, NY, USA, pp 77–85. https://doi.org/10.1145/2487575.2487581
4. Shun J, Blelloch GE (2013) Ligra: a lightweight graph processing framework for shared memory. In: Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13. ACM, New York, NY, USA, pp 135–146. https://doi.org/10.1145/2442516.2442530
5. Pearce R, Gokhale M, Amato NM (2010) Multithreaded asynchronous graph traversal for in-memory and semi-external memory. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10. IEEE Computer Society, Washington, DC, USA, pp 1–11. https://doi.org/10.1109/SC.2010.34
6. Prabhakaran V, Wu M, Weng X, McSherry F, Zhou L, Haridasan M (2010) Managing large graphs on multi-cores with graph awareness. In: Proceedings of the 2012 USENIX Conference on Annual Technical Conference, USENIX ATC'12. USENIX Association, Berkeley, CA, USA, p 4. http://dl.acm.org/citation.cfm?id=2342821.2342825
7. Roy A, Mihailovic I, Zwaenepoel W (2013) X-stream: edge-centric graph processing using streaming partitions. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, pp 472–488
8. Shun J, Blelloch GE (2013) Ligra: a lightweight graph processing framework for shared memory. In: ACM SIGPLAN Notices, vol 48. ACM, pp 135–146
9. Gregor D, Lumsdaine A (2005) The parallel BGL: a generic library for distributed graph computations. In: In Parallel Object-Oriented Scientific Computing (POOSC)
10. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10. ACM, New York, NY, USA, pp 135–146. https://doi.org/10.1145/1807167.1807184
11. Salihoglu S, Widom J (2013) GPS: a graph processing system. In: Proceedings of the 25th International Conference on Scientific and Statistical Database Management, SSDBM. ACM, New York, NY, USA, pp 22:1–22:12. https://doi.org/10.1145/2484838.2484843
12. Cheng R, Hong J, Kyrola A, Miao Y, Weng X, Wu M, Yang F, Zhou L, Zhao F, Chen E (2012) Kineograph: taking the pulse of a fast-changing and connected world. In: Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys '12. ACM, New York, NY, USA, pp 85–98. https://doi.org/10.1145/2168836.2168846

13. Apache incubator giraph http://incubator.apache.org/giraph/. Accessed June 2011
14. Stutz P, Bernstein A, Cohen W (2010) Signal/collect: graph algorithms for the (semantic) web. In: Proceedings of the 9th International Semantic Web Conference on the Semantic Web—Volume Part I, ISWC'10. Springer, Berlin, pp 764–780. http://dl.acm.org/citation.cfm?id=1940281.1940330. Accessed 7 Nov 2010
15. Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM (2012) Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proc VLDB Endow 5(8):716–727. http://dl.acm.org/citation.cfm?id=2212351.2212354. Accessed 1 Apr 2012
16. Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) Powergraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12. USENIX Association, Berkeley, CA, USA, pp 17–30. http://dl.acm.org/citation.cfm?id=2387880.2387883. Accessed 8 Oct 2012
17. Chen R, Shi J, Chen Y, Chen H (2015) Powerlyra: differentiated graph computation and partitioning on skewed graphs. In: Proceedings of the Tenth European Conference on Computer Systems. ACM, p 1
18. Zhao Y, Yoshigoe K, Bian J, Xie M, Xue Z, Feng Y (2016) A distributed graph-parallel computing system with lightweight communication overhead. IEEE Trans Big Data 2(3):204–218
19. Nai L, Xia Y, Tanase IG, Kim H, Lin C-Y (2015) GraphBIG: understanding graph computing in the context of industrial solutions. In: 2015 SC-International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 1–12
20. Sundaram N, Satish N, Patwary MMA, Dulloor SR, Anderson MJ, Vadlamudi SG, Das D, Dubey P (2015) Graphmat: high performance graph analytics made productive. Proc VLDB Endow 8(11):1214–1225
21. Zhao Y, Yoshigoe K, Xie M, Zhou S, Seker R, Bian J (2014) Lightgraph: lighten communication in distributed graph-parallel processing. In: 2014 IEEE International Congress on Big Data (BigData Congress). IEEE, pp 717–724
22. Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: bringing order to the web. Technical Report, Stanford InfoLab
23. Hung BQ, Otsubo M, Hijikata Y, Nishida S (2010) Hits algorithm improvement using semantic text portion. Web Intell Agent Syst 8(2):149–164. http://dblp.uni-trier.de/db/journals/wias/wias8.html_HungOHN10. Accessed 1 Jan 2010
24. Lempel R, Moran S (2005) Rank-stability and rank-similarity of link-based web ranking algorithms in authority-connected graphs. Inf Retr 8(2):245–264
25. Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. In: Proceedings of OSDI, pp 599–613
26. Mayer C, Tariq MA, Mayer R, Rothermel K (2018) GrapH: Traffic-Aware Graph Processing. In: IEEE Transactions on Parallel and Distributed Systems. https://doi.org/10.1109/TPDS.2018.2794989
27. Snap (2006) http://snap.stanford.edu/data/soc-LiveJournal1.html. Accessed 06 Aug 2016
28. Kwak H, Lee C, Park H, Moon S (2010) What is Twitter, a social network or a news media? In: Proceedings of the 19th International Conference on World Wide Web. ACM, pp 591–600
29. Gjoka M, Kurant M, Butts CT, Markopoulou A (2010) Walking in Facebook: a case study of unbiased sampling of OSNs. In: INFOCOM, 2010 Proceedings IEEE. IEEE, pp 1–9
30. Hong S, Depner S, Manhardt T, Van Der Lugt J, Verstraaten M, Chafi H (2015) PGX. D: a fast distributed graph processing engine. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, p 58
31. McSherry FD, Isaacs R, Isard MA, Murray DG (2012) Differential dataflow. US Patent App. 13/468,726
32. Murray DG, McSherry F, Isaacs R, Isard M, Barham P, Abadi M (2013) Naiad: a timely dataflow system. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, pp 439–455
33. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, pp 423–438
34. Zhao Y, Yoshigoe K, Xie M, Zhou S, Seker R, Bian J (2014) Evaluation and analysis of distributed graph-parallel processing frameworks. J Cyber Secur 3:289–316
35. Guo Y, Biczak M, Varbanescu AL, Iosup A, Martella C, Willke TL (2014) How well do graph-processing platforms perform? An empirical performance evaluation and analysis. In: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE, pp 395–404

36. Elser B, Montresor A (2013) An evaluation study of bigdata frameworks for graph processing. In: 2013 IEEE International Conference on Big Data. IEEE, pp 60–67
37. Zhou C, Gao J, Sun B, Yu JX (2014) MOCgraph: scalable distributed graph processing using message online computing. Proc VLDB Endow 8(4):377–388
38. Nguyen D, Lenharth A, Pingali K (2013) A lightweight infrastructure for graph analytics. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, pp 456–471
39. Kulkarni M, Pingali K, Walter B, Ramanarayanan G, Bala K, Chew LP (2007) Optimistic parallelism requires abstractions. In: ACM SIGPLAN Notices, vol 42. ACM, pp 211–222
40. Chen R, Ding X, Wang P, Chen H, Zang B, Guan H (2014) Computation and communication efficient graph processing with distributed immutable view. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing. ACM, pp 215–226
41. The apache hama project. http://hama.apache.org/. Accessed 30 Nov 2010
42. Shi X, Luo X, Liang J, Zhao P, Di S, He B, Jin H (2018) Frog: asynchronous graph processing on GPU with hybrid coloring model. IEEE Trans Knowl Data Eng 30(1):29–42
43. Xiao W, Xue J, Miao Y, Li Z, Chen C, Wu M, Li W, Zhou L (2017) Tux2: distributed graph computation for machine learning. In: NSDI, pp 669–682
44. Yan D, Huang Y, Liu M, Chen H, Cheng J, Wu H, Zhang C (2018) GraphD: distributed vertex-centric graph processing beyond the memory limit. IEEE Trans Parallel Distrib Syst 29(1):99–114
45. Pearce R, Gokhale M, Amato NM (2014) Faster parallel traversal of scale free graphs at extreme scale with vertex delegates. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, pp 549–559