LightGraph: Lighten Communication in Distributed Graph-Parallel Processing

Yue Zhao, Kenji Yoshigoe, Mengjun Xie, and Suijian Zhou
Department of Computer Science
University of Arkansas at Little Rock
Little Rock Arkansas, U.S.A
{*yxzhao, kxyoshigoe, mxxie, sxzhou*}*@ualr.edu*

Remzi Seker* and Jiang Bian†
**Department of ECSSE*
Embry-Riddle Aeronautical University
Daytona Beach FL, U.S.A.
sekerr@erau.edu
†*Division of Biomedical Informatics*
University of Arkansas for Medical Sciences
Little Rock Arkansas, U.S.A.
jbian@uams.edu

***Abstract*—A number of graph-structured computing abstractions have been proposed to address the needs of solving complex and large-scale graph algorithms. Distributed Graphlab and its successor, PowerGraph, are two such frameworks that have demonstrated excellent performance with high scalability and fault tolerance. However, excessive communication and state sharing among nodes in these frameworks not only reduce network efficiency but may also cause a decrease in runtime performance. In this paper, we first propose a mechanism that identifies and eliminates the avoidable communication during synchronization in existing distributed graph structured computing abstractions. We have implemented our method on PowerGraph and created LightGraph to reduce communication overhead in distributed graph-parallel computation systems. Furthermore, to minimize the required intra-graph synchronizations for PageRank-like applications, LightGraph also employs an edge direction-aware graph partitioning strategy, which optimally isolates the outgoing edges from the incoming edges of a vertex when creating and distributing replicas among different machines. We have conducted extensive experiments using real-world data, and our results verified the effectiveness of LightGraph. For example, when compared with the best existing graph placement method in PowerGraph, LightGraph can not only reduce up to 27.6% of synchronizing communication overhead for intra-graph synchronizations but also cut up to 17.1% runtime for PageRank.**

***Keywords*-Graph-parallel Computing, Big-data, Communication Overhead**

I. Introduction

Complex networks such as social, biological and computer networks can be mathematically modeled as graphs. These real-world networks are often large in size, consisting of millions or even billions of vertices, and hundreds of billions of edges. Making sense of large real-world networks, ranging from social networks of friends to links between web pages in the World Wide Web to gene regulatory networks, is an increasingly important problem. Thus, the task of designing effective and scalable computing systems for analyzing and processing huge real-world graphs has attracted significant attention and research effort in recent years.

A number of graph-parallel computation frameworks [1]–[14] have been proposed and applied in real-world applications to address this need. These frameworks can be categorized into two types: single machine graph-processing systems [1]–[6] and distributed graph-processing systems [7]–[14]. Compared to single machine graph-processing systems, distributed graph-parallel computing systems feature higher scalability and greater computational and storage resources for handling larger networks. However, in distributed computing systems, communication and synchronization between subtasks allocated among different machines are significant obstacles for achieving good parallel program performance. For complex distributed graph-parallel computing systems, the communication problem deserves more consideration. Taking Pregel [8], distributed Graphlab [13] and PowerGraph [14] as illustrations, they all have to launch heavy communication among replicas of vertices or edges across multiple machines for synchronization, which burdens the I/O system of the underlying cloud/cluster platform and potentially impacts the overall runtime.

To alleviate the communication overhead and accelerate the graph-structured application execution, we propose a mechanism that identifies and eliminates the avoidable communication during synchronization in existing distributed graph structured computing abstractions. We implemented our method on PowerGraph and created LightGraph, a light communication distributed graph-parallel computation system. Furthermore, to minimize the required intra-graph synchronizations for PageRank-like applications, LightGraph also employs an edge direction-aware graph partitioning strategy, which optimally isolates the outgoing edges from the incoming edges of a vertex when creating and distributing replicas among different machines.

The rest of the paper is organized as follows. Section II introduces the background of this work. The challenges of communication and our countermeasures are presented in Section III. Section IV details the design and results of our experiments. Section V concludes the paper.

978-1-4799-5057-7/14 $31.00 © 2014 IEEE
DOI 10.1109/BigData.Congress.2014.106

II. BACKGROUND

The exponential growth in the volume and complexity of data has driven the development of large-scale data processing systems especially the raise of graph-parallel computing systems. In a graph-parallel abstraction, the input of an algorithm is presented as a graph, $G = \{V, E\}$, and the computation is conducted by executing a vertex-program Q in parallel on each vertex, $v \in V$. The vertex-program $Q_{(v)}$ can communicate (e.g., through shared-state in GraphLab [2], [13], or messages in Pregel [8] with neighboring instances $Q_{(u)}$ where $(u, v) \in E$.

A number of graph-parallel abstractions have emerged in the literature. Pregel [8] explores graph-parallelization through the use of a bulk synchronous distributed message-passing system. Several other systems are similar to Pregel including GPS [9] and Giraph [11]. Giraph runs on the Hadoop platform, where Giraph jobs are MapReduce jobs without the reduce phase. Giraph leverages the task-scheduling component of Hadoop clusters by running workers as special mappers, which communicate with each other to deliver messages between vertices. GPS [9] is drawn from Pregel with some new features: an extended API facilitating the expression of the global computations, a dynamic repartitioning scheme, and an optimization that distributes adjacency lists of high-degree vertices across all compute nodes to improve performance. Gregor et al. proposed the parallel BGL [7], a generic C++ library for distributed graph computation, and applied the paradigm of generic programming to the domain of graph computations. Kineograph [10] uses a stream of incoming data to construct a continuously changing graph.

In this paper, we focus our discussion on distributed GraphLab [13] and PowerGraph [14] as they are closely related to our study. Distributed GraphLab [13] extended the shared memory GraphLab [2] abstraction into a distributed setting by refining the execution model, relaxing the scheduling requirements, and introducing a new distributed data-graph execution engine with fault-tolerance mechanisms. In distributed GraphLab, vertex-programs have shared access to a distributed graph with data stored on each vertex and edge. Each vertex-program may directly access information on the current vertex, adjacent edges, and adjacent vertices irrespective of the direction of the edges. Moreover, vertex programs can schedule neighboring vertex-programs to be executed in the future. Distributed GraphLab also ensures serializability by preventing neighboring program instances from running simultaneously. In order to place a graph on a cluster of p machines, distributed GraphLab partitions the original graph using a balanced p-way edge-cut approach, in which vertices are evenly assigned to machines while minimizing the number of edges spanning across machines. However, the balanced edge-cuts approach performs poorly [15] on scale-free networks whose degree distributions follow the power-law [16].

PowerGraph [14] brings a number of improvements to the distributed GraphLab framework. First, PowerGraph proposes a three-phase programing model – Gather, Apply and Scatter (GAS) - for constructing a vertex-program. By directly exploiting the GAS decomposition to factor vertex-programs over edges, PowerGraph eliminates the degree dependence of the vertex-program [14]. Second, PowerGraph incorporates the best features from both Pregel [8] and GraphLab [2], [13]. From GraphLab, PowerGraph adopts the shared-memory and data-graph view of computation that frees users from architecting a communication protocol for sharing information. PowerGraph borrows the commutative associative message combiner from Pregel, which reduces communication overhead in the Gather phase. At last, PowerGraph uses a vertex-cut approach to address the issue of partitioning power-law graphs [14], since it can quickly shatter a large power-law graph by cutting a small fraction of very high degree vertices. Moreover, to make vertex-cut feasible, PowerGraph allows a single vertex program to span across multiple machines by factoring the vertex program along the edges in the graph. PowerGraph supports both the bulk-synchronous computation model like in Pregel and the asynchronous computation model as in GraphLab [14].

III. LIGHTGRAPH: LIGHTEN COMMUNICATION IN DISTRIBUTED GRAPH-PARALLEL ABSTRACTIONS

A. Challenges of Communication and Synchronization

In order to process a large-scale graph, a distributed graph-parallel computing system needs to partition the graph into smaller sub-graphs and distribute the sub-graphs to different machines. As mentioned above, GraphLab uses an edge-cut approach while PowerGraph adopts a vertex-cut strategy. Nevertheless, replicas (noted as ghosts in GraphLab and PowerGraph) have to be created for the vertices and edges across the cutting-line. Through synchronizing these replicas, computation states and data can traverse the sub-graphs placed on different machines.

In both GraphLab and PowerGraph, communication occurs during synchronizations of replicas, and the volume is proportional to the number of ghosts. One prominent problem in GraphLab is that when partitioning a power-law graph, it has to resort to a hashed (random) vertex placement algorithm that cuts across most of the edges and creates many unnecessary ghosts [14]. The communication overhead could then substantially increase and seriously impact the execution efficiency of graph-structured applications for power-law graphs. On the other hand, under PowerGraph, the vertex-cut partitioning process stores each edge exactly once, thus eliminating the need for edge-ghosts and removing the need for data updates on edges to be communicated to other sub-graphs [14].

In other words, with PowerGraph, only vertices are replicated and only vertex data need to be synchronized. From

the original vertex and its replicas, one is randomly chosen as the master, and the remaining vertices are noted as mirrors. In a typical vertex-program, the master runs the apply function and sends the updated vertex data to all mirrors. Although PowerGraph reduces the communication overhead significantly, when compared to GraphLab, its high communication overhead limits its performance and scalability [14].

B. Communication Overhead in PowerGraph

Existing distributed graph-parallel computing systems, including PowerGraph, blindly synchronize all replicas of a vertex or edge when there is a data change in one of the replicas. However, there are certain graph algorithms such as PageRank [17], in which the data on some of the replicas will never be accessed in future computation iterations. Therefore, the communication for synchronizing these mirrors can be avoided.

Through experimenting with PowerGraph, we found that in certain graph computing applications/algorithms, the direction of information or data flow is consistent with the direction of the edges. More specifically, in a directed edge, the data on the target vertex is not needed by that edge or that edge's source vertex during computation. Thus, in a distributed graph, the data on a vertex replica that has no outgoing edge will never be accessed by any other edges or vertices in future computation iterations: such replicas do not need to be synchronized. PageRank [17], HITS [18] and SALSA [19] all fall into this category. We define this category of applications/algorithms as a PageRank-like application/algorithm as follows:

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

$$\forall\ edge(u \to v) \in E \tag{1}$$

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

$$D_u = f(args[0], args[1], ..., args[i]) \tag{2}$$

and

$$D_{u \to v} = f(args[0]', args[1]', ..., args[j]') \tag{3}$$

in which

$$D_v \notin \Big(\big[args[0], args[1], ..., args[i]\big] \cup \big[args[0]', args[1]', ..., args[j]'\big] \Big) \tag{4}$$

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

We use a simplified example of running PageRank in PowerGraph with a sample graph to demonstrate our observation, as shown in Figure 1. Conceptually, the PageRank

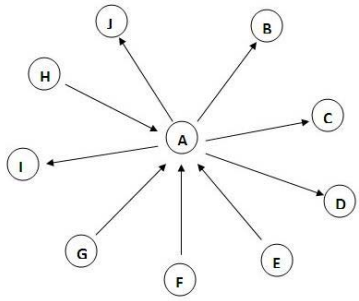

Figure 1. A partial sample graph for PageRank algorithm

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, (u), that are connected to v, defined as:

$$R(v) = (1 - \alpha) \sum_{u\ links\ to\ v} w_{u,v} \times R(u) + \frac{\alpha}{n} \tag{5}$$

where α is the damping factor (see [13], [17] for a detailed description of PageRank).

Figure 2 shows an example of a 4-way vertex-cut of the graph based on PowerGraph's partitioning algorithm [14]. Let us assume that we are computing the PageRank score of vertex A, and the replica of vertex A located on machine 2 is nominated as the master. In PowerGraph, the gather function first runs locally on each machine to calculate the partial PageRank score of vertex A based on the local sub-graph, then the partial value is sent from each mirror to the master. Second, the master runs the apply function to compute the new vertex value of A then sends the updated vertex data to all mirrors of A. Finally, the scatter phase is run in parallel on all mirrors of A and writes the new value back to the data graph for the next computing iteration. As shown in Figure 2, the mirror of vertex A located on machine 4 has no outgoing edges, which indicates that the data (PageRank score of vertex A) on this mirror will not be used by any other vertices within the sub-graph on machine 4 in future computations. Thus, the master of A does not need to synchronize this mirror during the update; therefore, the communication between the master (machine 2) and this mirror (machine 4) can be avoided. Moreover, the scatter phase on this mirror can also be eliminated.

C. The LightGraph Abstraction

Based on the above observation and analysis, we propose a mechanism that identifies and eliminates these avoidable communications during synchronizing master and replicas. We implemented our method on PowerGraph and created LightGraph to alleviate communication overhead for

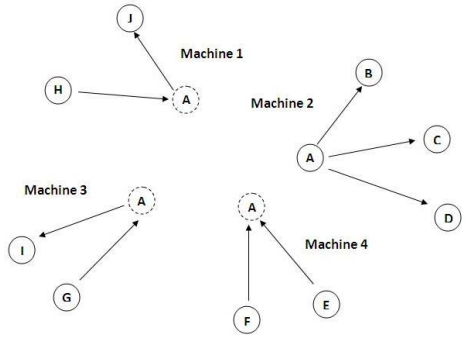

Figure 2. Graph placement under PowerGraph

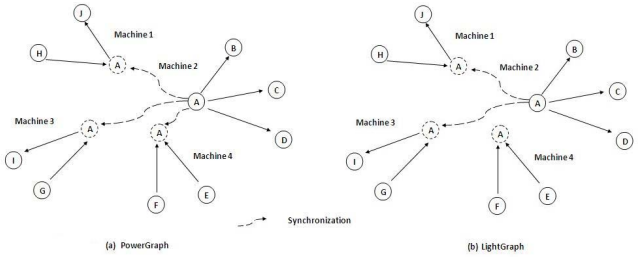

Figure 3. An illustration of reduced communication during synchronization in LightGraph compared with that of PowerGraph.

PageRank-like algorithms in distributed graph-parallel computation systems. More specifically, to achieve a light communication distributed graph-parallel computing system, we propose two novel methods in LightGraph: 1) a streamlined synchronization process that eliminates the unnecessary communications; and 2) an edge direction-aware vertex-cut partitioning strategy to maximize the proportion of mirrors with no outgoing edges to further reduce the communication overhead.

1) Streamlined Synchronization Process: Under PowerGraph, as illustrated above, the data on mirrors without outgoing edges will not be accessed in future computations for PageRank-like algorithms. Thus, even if the data on the master of a vertex has been updated there is still no need for the master to synchronize these mirrors. LightGraph identifies these mirrors during the initial partitioning process by checking their outgoing degree and then eliminates the synchronizing operations pertaining to these mirrors during the execution stage of the graph computing application. Consequently, LightGraph is able to reduce the overall communications required for PageRank-like algorithms. Figure 3 demonstrates the reduced communication workload of the synchronization process in LightGraph when compared with that of PowerGraph.

2) Edge Direction-Aware Distributed Graph Placement: As shown above, for PageRank-like algorithms, the synchronization to the mirrors without any outgoing edges can be eliminated. Naturally, we can reason, assuming

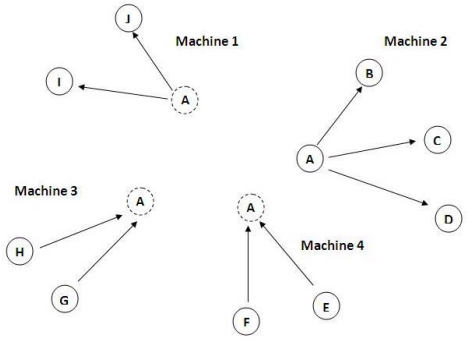

Figure 4. An example of graph placement using the proposed edge direction-aware partitioning strategy.

the same (or a similar) number of overall mirrors, that the less mirrors that are created with any outgoing edges, the less the synchronization communication overheads are required. Therefore, we propose a new graph displacement method in LightGraph to take into account the direction of edges during the initial graph partitioning phase, namely the edge direction-aware partition (EDAP) strategy. First, for a particular vertex, EDAP tries to assign edges with the same direction (inbound or outbound edge of a vertex) to the same machine during the graph partitioning process. In doing so, EDAP maximizes the proportion of replicas that have no outgoing edge among the overall vertex replicas. Second, instead of randomly appointing one of the vertex replicas as the master, EDAP chooses the master among the replicas that do have outgoing edges, since the synchronization communication only occurs from the master to mirrors. EDAP optimally isolates the outgoing edges from the incoming edges of a vertex on different machines while maintaining other partitioning mechanisms used by PowerGraph to guarantee good workload balance and low number of vertex replications.

Figure 4 illustrates the new 4-way placement of the sample graph shown in Figure 1 that is achieved by EDAP. Distinct from existing vertex-cut approaches (as demonstrated in Figure 2), the inbound edges of vertex A, edge ($H \rightarrow A$)) and edge ($G \rightarrow A$), are assigned to the same machine (machine 3), and the outbound edges of vertex A: edge ($A \rightarrow I$) and edge ($A \rightarrow J$), are placed together on machine 1. Consequently, in addition to the mirror in machine 4, the mirror on machine 3 is now also a no-outgoing-edge mirror of vertex A, Under this placement, once the data of vertex A is updated, the master of A only needs to synchronize the mirror in machine 1. Figure 5 shows the corresponding synchronizing scenario under the new graph placement achieved by EDAP. When compared with the synchronizing scenario using existing vertex-cut approaches as shown in Figure 3, the synchronizing communication is further reduced.

We implemented the proposed EDAP approach based on

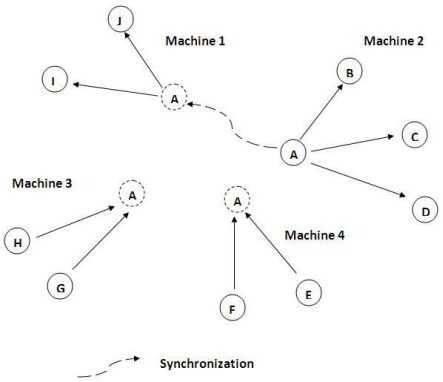

Figure 5. An example of synchronizing communications under EDAP-based graph placement.

two existing partitioning strategies in PowerGraph: Random and Oblivious [14]. The Random strategy uses a hash function that randomly distributes edges to machines. The Random strategy is fully data-parallel during the partitioning process and can achieve a near-perfect balance in workload distribution on large graphs. Alternately, the Oblivious partitioning strategy uses a sequential greedy heuristic whose goal is to place subsequent edges on appropriate machines to minimize the conditional expected replication factor. As defined in [14], the replication factor is the ratio of the number of overall replicas in the distributed graph over the number of vertices 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 that contain its adjacent edges, the replication factor can be formally defined as:

$$ReplicationFactor = \frac{1}{|V|}\sum_{v \in V}|A(v)|. \quad (6)$$

Therefore, the objective of the Oblivious strategy is to place the i+1 edge after having placed the previous i edges, such that:

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

where A_{e_i} is the assignment for the ith edge, j is the ID of a machine in the distributed system.

Oblivious runs the greedy heuristic independently on each machine without additional communication and it has the best performance of all the partitioning strategies implemented in PowerGraph [14].

In LightGraph, we extended the Random and Oblivious partitioning strategies using the edge direction awareness feature and created EDAP_Random and EDAP_Oblivious, respectively. More specifically, based on the Random and Oblivious algorithm, we added a heuristic that prefers to place the edge (i+1) following the placement of previous i edges to a machine on which the source vertex of the edge already has other outgoing edges or the target vertex of the edge already has other incoming edges. By doing so, we maximize the chances of creating mirrors with only incoming (or outgoing) edges.

Table I compares the key characteristics of LightGraph with three state-of-the-art graph parallel abstractions.

IV. Experimental Evaluation

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

A. 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 2.66GHz 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.

B. Benchmarking Application and Dataset

We utilized the PageRank application as the benchmarking application in all of our experiments and used a directed friendship social network dataset: soc-LiveJournal1 [20]. The soc-LiveJournal1 dataset is 1GB in size, and the corresponding graph has 4.8 million vertices and 68.9 million edges.

C. Experiment Design and Results

We ran the PageRank application and compared measured benchmarking metrics - such as runtime and volume of synchronizing communication - under LightGraph with those under PowerGraph. Our first evaluation goal was to measure the reduction in communication of the streamlined synchronization process in LightGraph using the existing Random and Oblivious partitioning strategies in PowerGraph. Then, we repeated the experiment and took the same set of measurements using the proposed EDAP_Random and EDAP_Oblivious partitioning strategies to demonstrate the effectiveness of direction-aware partitioning approaches. We also conducted all experiments using both synchronous and asynchronous computation modes.

To demonstrate that LightGraph has the potential to eliminate unnecessary synchronizing communication, we measured the number of overall mirrors produced by each partitioning schema and the number of mirrors that need to be synchronized during the execution of the PageRank application under each execution scenario. Note that PowerGraph blindly synchronizes all mirrors while LightGraph

Table I
COMPARING KEY CHARACTERISTICS OF LIGHTGRAPH WITH EXISTING GRAPH-PARALLEL ABSTRACTIONS.

Metrics	Pregel	Distributed GraphLab	PowerGraph	LightGraph
Required Sync communication	$\propto$ # of ghosts	$\propto$ # of ghosts	$\propto$ # of mirrors	$\propto$ # of mirrors having out-edges
Graph partitioning method	Edge-cut	Edge-cut	Vertex-cut	Edge direction-aware vertex-cut
Computation model	Synchronous	Synchronous & Asynchronous	Synchronous & Asynchronous	Synchronous & Asynchronous

only synchronizes mirrors that have outgoing edges. In other words, synchronizations for mirrors that have only incoming edges are eliminated under LightGraph. Figure 6 shows that by identifying the mirrors without outgoing edges and eliminating the synchronization communication to them, the number of mirrors synchronized in LightGraph is reduced significantly compared with PowerGraph; this is the case for both Random and Oblivious partitioning strategies. For example, in the 16-way cut distributed graph produced in our experiment by partitioning the input graph using the Random strategy, 24.1% of all mirrors have no outgoing edges. Instead of synchronizing all mirrors as in PowerGraph (Random), LightGraph (Random) only synchronizes the remaining 75.9% of mirrors during the execution of PageRank. Combining Figure 6 and Figure 7 we can see that both edge direction awareness partitioning approaches (EDAP_Random and EDAP_Oblivious) in LightGraph increase the percentage of mirrors that do not have outgoing edges compared with the respective Random and Oblivious partitioning strategies in PowerGraph. Thus, the proposed edge direction-aware partitioning methods further reduced the number of mirrors that need to be synchronized. For instance, when using LightGraph (EDAP_ Random), in the 8-way cut distributed graph, the number of mirrors that need to be synchronized is 0.73 million, which is 22.0% and 16.8% of that required when using LightGraph (Random) and PowerGraph (Random), respectively. Although, as shown in Figure 7, EDAP_Random induces a higher replication factor (i.e., creates more mirrors than the Random strategy), the number of mirrors that need to be synchronized is still significantly lower, since the EDAP_Random strategy produces a significantly higher number of mirrors without any outgoing edges. Alternately, EDAP_Oblivious not only increased the percentage of zero out-degree vertices, but also reduced the number of overall mirrors compared to Oblivious.

Figure 8 and Figure 9 show the volume of synchronizing communication under PowerGraph and LightGraph under synchronous and asynchronous computation mode, respectively. As expected, LightGraph and its edge direction-aware partitioning strategies can significantly eliminate avoidable synchronizing communication during the PageRank application execution under both modes. Moreover, as the number of machines increases, the volume of synchronizing communication saved in LightGraph also increases.

Figure 10 and Figure 11 show PageRank runtime using different partitioning strategies in LightGraph and PowerGraph under synchronous mode and asynchronous mode, respectively. As shown in the figures, through eliminating the avoidable synchronizing communication, LightGraph reduced the execution time of PageRank under both syn-

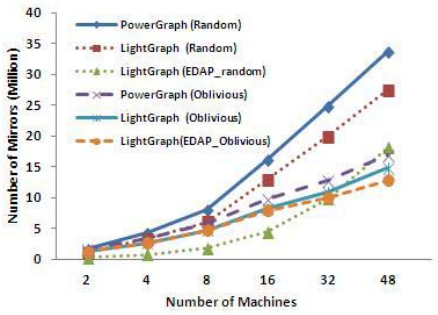

Figure 6. Number of mirrors that need to be synchronized under different partitioning strategies in PowerGraph and LightGraph for the PageRank application.

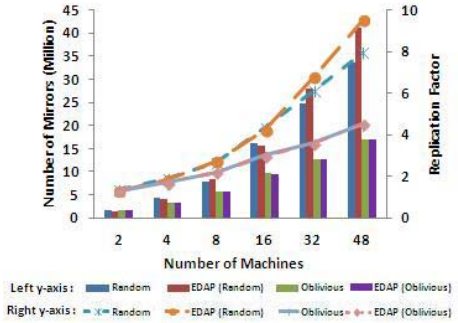

Figure 7. Number of overall mirrors and replication factors under different partitioning strategies.

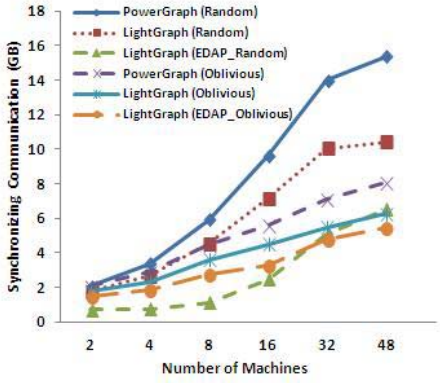

Figure 8. Comparing the volume of synchronizing communication in synchronous computation mode.

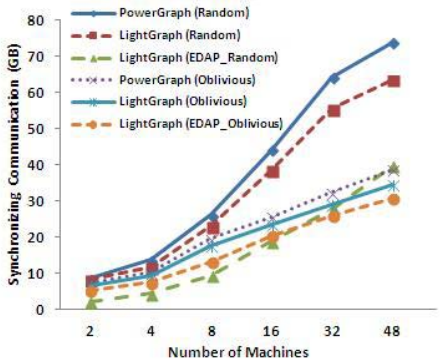

Figure 9. Comparing the volume of synchronizing communication in asynchronous computation mode.

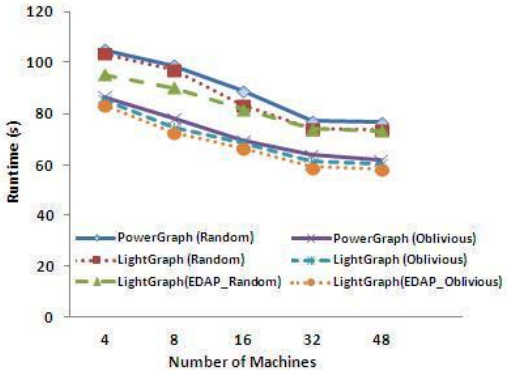

Figure 10. PageRank runtime in synchronous mode.

chronous and asynchronous modes. Moreover, the proposed edge direction-aware partitioning strategies further improved the performance of the PageRank application. Furthermore, LightGraph shows consistent performance gains through reducing synchronizing communication as the number of machines increases in the cluster. Although, for PageRank, the volume of synchronizing communication has been drastically reduced, the overall completion time did not decrease significantly. The reason is that a bulk of communication overhead generated by the PageRank application can be already hidden in the GAS three-phase programing model of the distributed PowerGraph. Moreover, for a CPU-bound algorithm such as PageRank, the effect of eliminating communication overhead on runtime performance will not be evident.

V. Conclusion

Although distributed graph-parallel computing systems such as PowerGraph can provide high computational capabilities and scalability for large-scale graph-structured computation, they often suffer from heavy communication overhead. In this paper, we proposed LightGraph that eliminates the avoidable communications during synchronization of mirrors in existing distributed graph structured computing

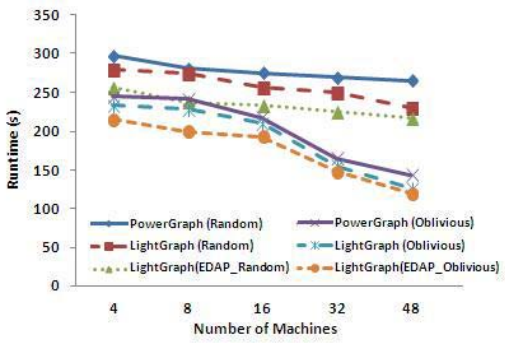

Figure 11. PageRank runtime in asynchronous mode.

abstractions. Through extensive experiments with real-world network data, we show that LightGraph can not only reduce synchronizing communication 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 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.

Acknowledgment

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 Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 31–46. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387884

[2] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A new parallel framework for machine learning," in Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), Catalina Island, California, July 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 Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 77–85. [Online]. Available: http://doi.acm.org/10.1145/2487575.2487581

[4] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP '13. New York, NY, USA: ACM, 2013, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442530

[5] R. Pearce, M. Gokhale, and N. M. Amato, "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, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.34

[6] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haridasan, "Managing large graphs on multi-cores with graph awareness," in Proceedings of the 2012 USENIX Conference on Annual Technical Conference, ser. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342821.2342825

[7] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graph computations," in In Parallel Object-Oriented Scientific Computing (POOSC, 2005.

[8] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807184

[9] S. Salihoglu and J. Widom, "Gps: A graph processing system," in Proceedings of the 25th International Conference on Scientific and Statistical Database Management, ser. SSDBM. New York, NY, USA: ACM, 2013, pp. 22:1–22:12. [Online]. Available: http://doi.acm.org/10.1145/2484838.2484843

[10] 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 Proceedings of the 7th ACM European Conference on Computer Systems, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 85–98. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168846

[11] Apache incubator giraph. [Online]. Available: url{http://incubator.apache.org/giraph/}

[12] P. Stutz, A. Bernstein, and W. Cohen, "Signal/collect: Graph algorithms for the (semantic) web," in Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I, ser. ISWC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 764–780. [Online]. Available: http://dl.acm.org/citation.cfm?id=1940281.1940330

[13] 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," Proc. VLDB Endow., vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2212351.2212354

[14] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 17–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387883

[15] K. Lang, "Finding good nearly balanced cuts in power law graphs," Tech. Rep., 2004.

[16] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs." in IPDPS. IEEE, 2006. [Online]. Available: http://dblp.uni-trier.de/db/conf/ipps/ipdps2006.html#Abou-RjeiliK06

[17] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," 1999.

[18] B. Q. Hung, M. Otsubo, Y. Hijikata, and S. Nishida, "Hits algorithm improvement using semantic text portion." Web Intelligence and Agent Systems, vol. 8, no. 2, pp. 149–164, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/wias/wias8.html#HungOHN10

[19] R. Lempel and S. Moran, "Rank-stability and rank-similarity of link-based web ranking algorithms in authority-connected graphs," Inf. Retr., vol. 8, no. 2, pp. 245–264, 2005.

[20] SNAP. (2006) Livejournal social network. [Online]. Available: http://snap.stanford.edu/data/soc-LiveJournal1.html