Bipartite-oriented Distributed Graph Partitioning for Big Learning

Rong Chen†, Jiaxin Shi†, Binyu Zang†, Haibing Guan§
Shanghai Key Laboratory of Scalable Computing and Systems
†*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*
§*Department of Computer Science, Shanghai Jiao Tong University*
{*rongchen, shijiaxin, byzang, hbguan*}*@sjtu.edu.cn*

abstract

Many machine learning and data mining (MLDM) problems like recommendation, topic modeling and medical diagnosis can be modeled as computing on bipartite graphs. However, most distributed graph-parallel systems are oblivious to the unique characteristics in such graphs and existing online graph partitioning algorithms usually causes excessive replication of vertices as well as significant pressure on network communication. This article identifies the challenges and opportunities of partitioning bipartite graphs for distributed MLDM processing and proposes BiGraph, a set of bipartite-oriented graph partitioning algorithms. BiGraph leverages observations such as the skewed distribution of vertices, discriminated computation load and imbalanced data sizes between the two subsets of vertices to derive a set of optimal graph partition algorithms that result in minimal vertex replication and network communication. BiGraph has been implemented on PowerGraph and is shown to have a performance boost up to 17.75X (from 1.38X) for four typical MLDM algorithms, due to reducing up to 62% vertex replication, and up to 96% network traffic.

1 Introduction

As the concept of "Big Data" gains more and more momentum, running many MLDM problems in a cluster of machines has been a norm. This also stimulates a new research area called Big Learning, which uses leverage a set of networked machines for parallel and distributed processing of more complex algorithms and larger problem sizes. This, however, creates new challenges to efficiently partition a set of input data across multiple machines to balance load and reduce network traffic.

Currently, many MLDM problems concern large graphs such as social and web graphs. These problems are usually coded as vertex-centric programs by following the "think as a vertex" philosophy, where vertices are processed in parallel and communicate with their edges through their edges. For distributed processing of such graph-structured programs, graph partitioning plays a central role to distribute vertices and their edges across multiple machines, as well as to create replicated vertices and/or edges to form a locally consistent graph states in each machine.

Though graphs can be arbitrarily formed, real-world graphs usually have some specific properties to reflect their application domains. Among them, many MLDM algorithms model their input graphs as bipartite graphs, whose vertices can be separated as two disjoint sets $\mathbb{U}$ and $\mathbb{V}$ and every edge connects a vertex in $\mathbb{U}$ and $\mathbb{V}$. Example bipartite graphs include the customers/-goods graph in recommendation systems, topics/documents graph in topic modeling algorithms. Due to the wide existence and importance of bipartite graphs, there have been a number of popular MLDM algorithms that operate on such graphs, including Singular Value Decomposition (SVD), Alternating Least Squares (ALS), Stochastic Gradient Descent (SGD), Belief Propagation (BP) and Latent Dirichlet Allocation (LDA), with the application domains ranging from recommendation systems to medical diagnosis and text analysis.

Despite the popularity of bipartite graph, there is little study on how to efficient partition bipartite graphs in large-scale graph computation systems. Most existing systems simply apply general graph partitioning algorithms that are oblivious to the unique features of bipartite graphs. This results in suboptimal graph partition

APSys '14, June 25-26, 2014, Beijing, China

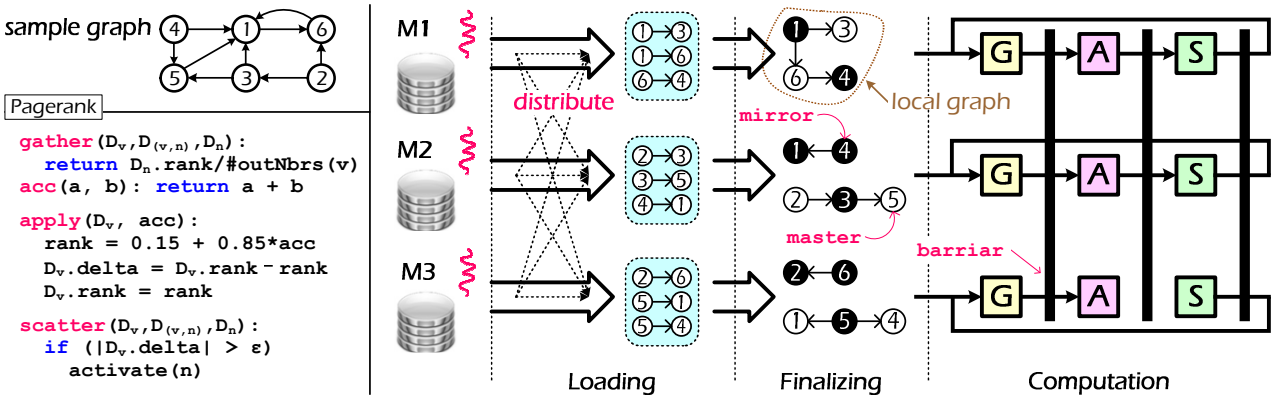

Fig. 1: *Pseudo-code of PageRank algorithm using GAS model, and execution flow of graph-parallel system for a sample graph.*

with significant replicas of vertices and/or edges, leading to not only redundant computation, but also excessive network communication to synchronize graph states. Though there are some graph partition algorithms for bipartite graphs, none of them satisfies the requirement of large-scale graph computation. Many of them [5, 15] are offline partitioning algorithms that require fully information of graphs, and thus are very time-consuming and not scalable to large graph dataset. Some others [7, 6] only work on a special type of graphs or algorithms, making them hard to be generalized to a wide range of MLDM problems.

In this paper, we make a systematic study on the characteristics of real-world bipartite graphs and the related MLDM algorithms, and describe why existing online distributed graph partitioning algorithms fail to produce an optimal graph partition for bipartite graphs. Based on the study, we argue that the unique properties of bipartite graphs and special requirement of bipartite algorithms demand *differentiated* partitioning [3] of the two disjoint sets of vertices. Based on our analysis, we introduce BiGraph, a set of distributed graph partition algorithms designed for bipartite applications. The key of BiGraph is to partition graphs in a differentiated way and loading data according to the data affinity.

We have implemented BiGraph[1] as separate graph partition modules of GraphLab [12, 8], a state-of-the-art graph-parallel framework. Our experiment using three MLDM algorithms on an in-house 6-machine multicore cluster with 144 CPU cores shows that BiGraph reduces up to 62% vertex replication, and saves up to 96% network traffic. This transforms to a speedup up to 17.75X (from 1.38X) compared to the state-of-the-art Grid [9] partitioning algorithm (the default partitioning algorithm in GraphLab).

[1]The source code and a brief instruction of how to use BiGraph are at `http://ipads.se.sjtu.edu.cn/projects/powerlyra.html`

2 Graph-parallel Systems and Partition Algorithms

2.1 Graph-parallel Systems

Many distributed graph-parallel systems, including Pregel [13] and GraphLab [12, 8], follow the 'think as a vertex' philosophy and abstract a graph algorithm as a vertex-centric program P , The program is executed in parallel on each vertex $v \in V$ in a sparse graph $G = \{V, E, D\}$. The scope of computation and communication in each $P(v)$ is restricted to neighboring vertices through edges where $(s,t) \in E$. Programmers can also associate arbitrary data D_v and $D_{s,t}$ to vertex $v \in V$, and edge $(s,t) \in E$ respectively.

As shown in Figure 1, the right part illustrates the overall execution flow for the sample graph on PowerGraph, the latest version of the GraphLab framework[2]. To exploit distributed computing resources of a cluster, the runtime splits an input graph into multiple partitions and iteratively executes user-defined programs on each vertex in parallel.

First, the runtime *loads* graph topology and data files from a secondary storage (e.g., HDFS or NFS), and *distributes* vertices and edges to target machines according to a graph partition algorithm. Then, replicas for vertices are created for each edges spanning machines to *finalize* in-memory local graphs such that each machine has a locally-consistent sub-graph . PowerGraph adopts the GAS (**G**ather, **A**pply, **S**catter) model to abstract graph algorithms and employs a loop to express *iterative computation* in many MLDM algorithms. The pseudo-code in Figure 1 illustrates an example implementation of the PageRank [2] algorithm implemented by the GAS model. In the Gather phase, the `gather` and `accum` functions accumulate the rank of neighboring vertices through in-edges; and then the `apply` function

[2]GraphLab after version 2.1 runs the PowerGraph engine

calculates and updates a new rank to vertex using accumulated values in the Apply phase; finally the `scatter` function sends messages and activates neighboring vertices through out-edges in the Scatter phase.

2.2 Distributed Graph Partitioning

A key to efficient Big Learning on graph-parallel systems is optimally placing graph-structured data including vertices and edges across multiple machines. As graph computation highly relies on the distributed graph structures to store graph computation states and encode interactions among vertices, an optimal graph partitioning algorithm can minimize communication cost and ensure load balance of vertex computation.

There are two main types of approaches: offline and online (streaming) graph partitioning. Offline graph partitioning algorithms (e.g., Metis and spectral clustering) require full graph information has been known by all workers (e.g. machines), which requires frequent coordination among workers in a distributed environment. Though it may produce a distributed graph with optimal graph placement, it causes not only significant resources consumption, but also lengthy execution time even for a small-scale graph. Consequently, offline partitions are rarely adopted by large-scale graph-parallel systems for Big Learning. In contrast, online graph partitioning aims at to find a near-optimal graph placement by distributing vertices and edges with only limited graphs information. Due to the significant less partitioning time yet still-good graph placement, it has been widely adopted by almost all large-scale graph-parallel systems.

There are usually two approaches in online graph partitioning: edge-cut, which divides a graph by cutting cross-partition edges among sub-graphs; and vertex-cut, which partitions cross-partition vertices among sub-graphs. General speaking, vertex-cut can evenly distributing vertices among multiple partitions, but may result in imbalanced computation and communication as well as high replication factor for skewed graphs. In contrast, vertex-cut can evenly distributing edges, but may incur high communication cost among partitioned vertices.

PowerGraph employs several vertex-cut algorithms [8, 9] to provide *edge* balanced partitions, because the workload of graph algorithms mainly lies to the number of edges. Figure 2 illustrates the hash-based (random) vertex-cut to evenly assign edges to machines, which has a high replication factor (i.e., $\lambda = \#replicas/\#vertices$) but very simple implementation; and currently the default graph partitioning algorithm in PowerGraph, Grid [9] vertex-cut, which uses a grid-based heuristic to reduce replication factor by constraining the location of edges. In this case, random partitioning simply assigns

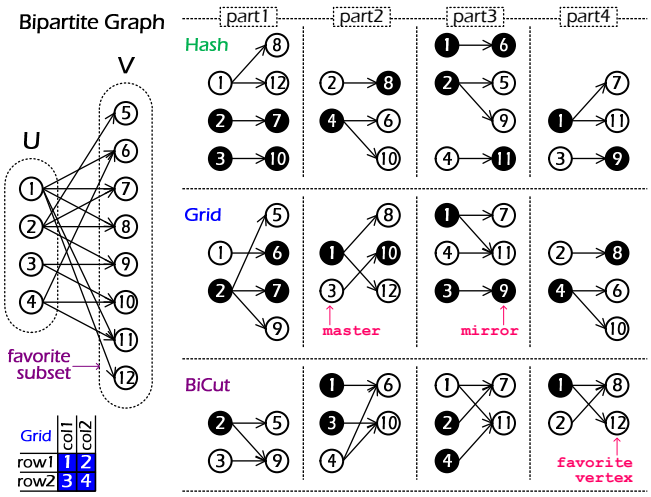

Fig. 2: *A comparison of various partitioning algorithms on a bipartite graph.*

edges by hashing the sum of source and target vertex-ids, while grid vertex-cut further specify the location to only an intersection of shards of the source and target vertices. For example, in random vertex-cut, the edge $(1,8)$ is assigned to partition 1 as the sum of 1 and 8 divided by 4 (the total partition number) is 1. In Grid vertex-cut, the edge $(1,8)$ is randomly assigned by Grid to one of the intersected partitions (2 and 3) according to the partitioning grid (the left bottom corner of Figure 2). This is because vertex 1 is hashed to part 1, which constrains the shards of vertex 1 to row 1 and column 1, while vertex 8 is hashed to part 4, which constrains the shards of vertex 8 to row 2 and column 2. Thus, the resulting shards of the intersection are 2 and 3. Unfortunately, both partitioning algorithms result in suboptimal graph placement and the replication factor is high (2 and 1.83 accordingly), due to no awareness of the unique features in bipartite graphs.

Our prior work, PowerLyra [3], also uses differentiated graph partitioning for skewed graphs. However, it does not consider the special properties of bipartite graphs as well as access locality.

3 Challenges and Opportunities

All vertices in a bipartite graph can be partitioned into two disjoint subsets $\mathbb{U}$ and $\mathbb{V}$, and each edge connects a vertex from $\mathbb{U}$ to one from $\mathbb{V}$, as shown in the left part of Figure 2. Such special properties of bipartite graphs and the special requirement of MLDM algorithms impede existing graph partitions to obtain a proper graph cut and performance. Here, we describe several observations from real-world bipartite graphs and the characteristics of MLDM algorithms.

First, real-world bipartite graphs for MLDM are usually imbalanced. This means that the size of two subsets in bipartite graph is significantly skewed, even in the scale of several orders of magnitude. For example, there is only ten thousands of terms in Wikipedia, while the number of articles has exceeded four millions. The number of grades from students may be dozen times to the number of courses. As an concrete example, the SLS [4] dataset, a 10 years of grade point scores at a large State University, has 62,729 objects (e.g., students, instructors and departments) and 1,748,122 scores (ratio: 27.87). This implies that a graph partitioning algorithm needs to employ differentiated mechanisms on vertices from different subsets.

Second, the computation load of many MLDM algorithms for bipartite graphs may also be skewed among vertices from the two subsets. For example, Stochastic Gradient Descent (SGD) [10], a collaborative filtering algorithm for recommendation systems, only calculates new cumulative sums of gradient updates for *user* vertices in each iteration, but none for *item* vertices. Therefore, an ideal graph partitioning algorithm should be able to discriminate the computation to one set of vertices and exploit the locality of computation by avoiding an excessive replication of vertices.

Finally, the size of data associated with vertices from the two subsets can be significantly skewed. For example, to use probabilistic inference on large astronomical images, the data of an *observation* vertex can reach several terabytes, while the latent *stellar* vertex has only very few data. If a graph partitioning algorithm distributes these vertices to random machines without awareness of data location, it may lead to excessive network traffic and significant delay in graph partitioning time. Further, the replication of these vertices and the synchronization among them may also cause significant memory and network pressure during computation. Consequently, it is critical for a graph partitioning algorithm to be built with data affinity support.

4 Bipartite-oriented Graph Partitioning

The unique features of real-world bipartite graphs and the associated MLDM algorithms demand a bipartite-aware online graph partitioning algorithm. BiCut is a new heuristic algorithm to partition bipartite graphs, by leveraging our observations. Based on the algorithm, we describe a refinement to further reduce replications and improve load balance, and show how BiCut supports data affinity for bipartite graphs with skewed data distribution to reduce network traffic.

4.1 Randomized Bipartite-cut

Existing distributed graph partitioning algorithms use the same strategy to distribute all vertices of a bipartite graph, which cannot fully take advantage of the graph structures in bipartite graphs, especially for skewed graphs.

In bipartite graph, two vertices connected by an edge must be from different disjointed subsets. This implies that *arbitrarily partitioning vertices belonging to the same subset may not introduce any replicas of vertices*, as there is no edge connecting them. Based on above observation, the new *bipartite-cut* algorithm applies a differentiated partitioning strategy to avoid replication for vertices in one favorite subset and provide fully local access to adjacent edges. First, the vertices in this set are first-class citizens during partitioning, which are evenly assigned to machines with all adjacent edges at first. Such vertices contain no replicas. Then, the replicas of vertices in the other subset are created to construct a local graph on each machine. One replica of the vertex is randomly nominated as the master vertex, which coordinates the execution of all remaining replicas.

As shown in Figure 2, since the favorite subset is $\mathbb{V}$, whose vertices (from 5 to 12) with all edges are evenly assigned to four partitions without replication. The vertices in subset $\mathbb{U}$ (from 1 to 4) are replicated on demand in each partition. All edges of vertices in the favorite set can be accessed locally. By contrast, the vertex in the subset $\mathbb{U}$ has to rely on its mirrors for accessing its edges (e.g., vertex 1). Hence, BiCut only results in a replication factor of 1.5.

By default, BiCut will use the subset with more vertices as the favorite subset to reduce the replication factor. However, if the computation on one subset is extremely sensitive to locality, this subset can also be designated as the favorite subset to avoid network traffic in computation, since the edges are always from one subset to the other. As there is no extra step with high cost, the performance of BiCut should be comparable to random vertex-cut.

4.2 Greedy Bipartite-cut

Our experiences show that the workload of graph-parallel system highly depends on the balance of edges. Unfortunately, randomized bipartite-cut only naturally provides the balance of favorite vertices. To remedy this issue, we propose a new greedy heuristic algorithm, namely Aweto, inspired by Ginger [3], a greedy heuristic hybrid-cut for natural graphs.

Aweto uses an additional round of edge exchange to exploit the *similarity* of neighbors between vertices in the favorite subset and the balance of edges in partitions. Af-

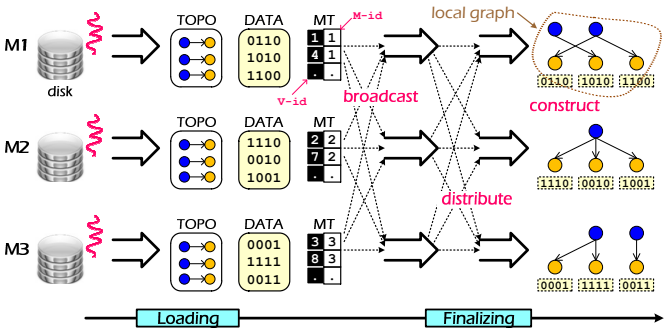

Fig. 3: *The execution flow of bipartite-cut with data affinity.*

ter the random assignment of the first round, each worker greedily re-assigns a favorite vertex v with all edges to partition i such that $\delta g(v,S_i) \geq \delta g(v,S_j), for\ all\ j \in \{1,2,\ldots,p\}$, where S_i is the current vertex set on partition $\in P = (S_1,S_2,\ldots,S_p)$. Here, $\delta g(v,S_i) = |N(v) \cap S_i| - B(|S_i|)$, where $N(v)$ denotes the set of neighbors of vertex v, $|S_i|$ denotes the current number of edges on partition i and $B(x) = |S_i|^{\frac{1}{2}}$. $B(x)$ is used to balance the edges re-assigned from current partition.

Because each partition maintains its own current vertex set S_i, the greedy-heuristic algorithm can independently execute on all machines without any communication. Further, the balance of edges re-assigned by each partition implies a global edge balance of partitions.

4.3 Data Affinity

A large number of graph data may be stored on each machine of a cluster. For example, the output files of a MapReduce job will not be merged and stored to the local secondary storage. The modern distributed filesystem (e.g., HDFS) also split files into multiple blocks and stored on multiple machines to improve availability and fault tolerance. Without data affinity, existing data partition only consider affinity within graph to reduce replication, which results in a large amount of data movement from the secondary storage of one machine to the main memory of other machines.

To reduce movement of a huge amount of data and avoid replication of favorite vertices, bipartite-cut is further extended to support data affinity. The favorite vertices with a huge amount of data is fixedly placed on machines holding their vertex data, and its edges are also re-assigned to those machines. In this way, the computation on favorite vertex is restricted to local machine without network traffic.

Figure 3 illustrates the execution flow of BiCut with data affinity. First, all topology information and data of graph are *loaded* from local secondary storage to memory, and a local mapping table (MT) from favorite vertices to the current machine is generated on each machine. Then the local mapping table on each machine is *broadcasted* to other machines to create a global mapping table on each machine. The edge *distribution* originally in graph loading is delayed to the end of exchanging mapping tables. Finally, the local graph is *constructed* as before by replicating vertices.

Table 1: *A collection of bipartite graphs. The symbol of (G|B|A) corresponds with Grid, BiCut and Aweto.*

Graphs	$\|U\|$	$\|V\|$	$\|E\|$	$\|U\|/\|V\|$	Rep-Factor (G|B|A)
LJ	4,489K	4,308K	69.0M	1.04	3.07 | 2.48 | 2.26
AS	1,696K	967K	11.1M	1.75	2.80 | 1.82 | 1.57
WG	739K	715K	5.1M	1.03	2.85 | 2.05 | 1.60
RUCCI	1,978K	110K	7.8M	18.00	3.10 | 1.26 | 1.26
SLS	1,748K	63K	6.8M	27.87	3.03 | 1.17 | 1.15
ESOC	327K	38K	6.0M	8.65	3.98 | 1.48 | 1.31
Netflix	480K	17K	100M	27.02	3.97 | 1.18 | 1.18

5 Performance Benefit of BiGraph

We have implemented BiGraph based on GraphLab 2.2 (release in July 2013), which runs the PowerGraph engine. BiCut and Aweto are implemented as separate graph-cut modules for GraphLab.

We evaluate BiGraph against the default graph partition, Grid [9], on GraphLab framework using three typical bipartite graph algorithms: *Singular Value Decomposition* (SVD), *Alternating Least Squares* (ALS) and *Stochastic Gradient Descent* (SGD).

5.1 Experimental Setup

All experiments were performed on an in-house 6-machine multicore cluster with a total of 144 CPU cores. Each machine has a 24-core AMD Opteron 6168 CPU, 64GB RAM, 2x1TB Hard Drivers and 1 GigE network ports. GraphLab runs 24 threads on each machine. We run NFS on the same cluster as the underlying storage layer.

Table 1 lists a collection of bipartite graphs used in our experiments. They are from Stanford Large Network Dataset Collection [14] and The University of Florida Sparse Matrix Collection [4]. The former three bipartite graph is balanced, the ratios of $|U|/|V|$ are from 1.03 to 1.75. These can be regarded as worst cases for bipartite-cut due to their balanced distribution. On the contrary, the latter three bipartite graph is relative skewed, $|U|/|V|$ are from 8.65 to 27.87. These are more suitable for bipartite-cut. The last Netflix prize dataset [1] is used as a building block to simulate large datasets with various sizes for the weak scalability experiment (section 5.4). All vertices in the subset $|U|$ and edges are duplicated to scale the dataset [11].

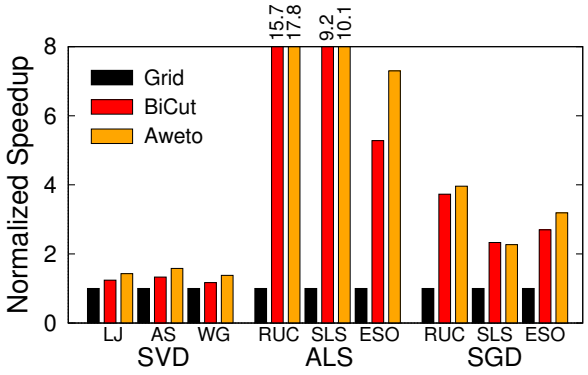

Fig. 4: *Normalized speedup over Grid on computation performance.*

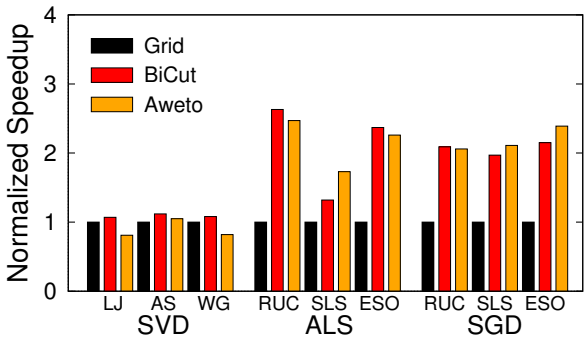

Fig. 5: *Normalized speedup over Grid on partitioning performance.*

5.2 Overall Performance

Figure 4 summarizes the normalized speedup of BiCut and Aweto compared to Grid graph partitioning on the computation phase with different algorithms and graphs. Since all partitioning algorithms use the same execution engine in computation phase, the speedup highly depends on the reduction of replication factors, which dominates the communication cost. For skewed bipartite graphs, BiCut can remarkably decrease replication factor by avoiding replication for all vertices in the favorite subset (e.g., U). BiCut reduces 59%, 61% and 63% replicas for the RUCCI, SLS and ESOC graphs. BiCut outperforms Grid partitioning by up to 15.65X (from 5.28X) and 3.73X (from 2.33X) for ALS and SGD accordingly. For balanced bipartite graphs, BiCut still moderately reduces the replication factor, and provides a speedup by 1.24X, 1.33X and 1.17X for LJ, AS and WG graphs on SVD. Aweto can further reduce replication factor and provides up to 38% additional improvement.

Figure 5 illustrates the graph partition performance of BiCut and Aweto against Grid, including loading and finalizing time. BiCut outperforms Grid by up to 2.63X (from 1.07X) due to lower replication factor, which reduces the cost for data movement and replication construction. Aweto is lightly slower than BiCut because of additional edge exchange.

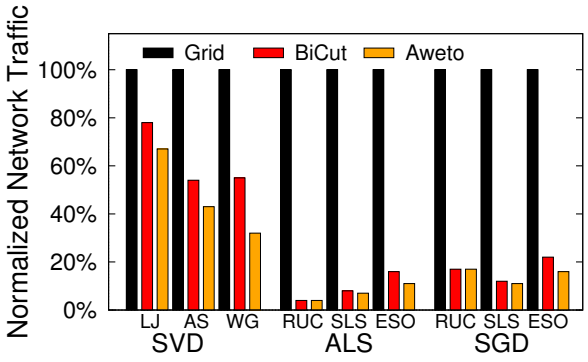

Fig. 6: *Percent of network traffic reduction over Grid.*

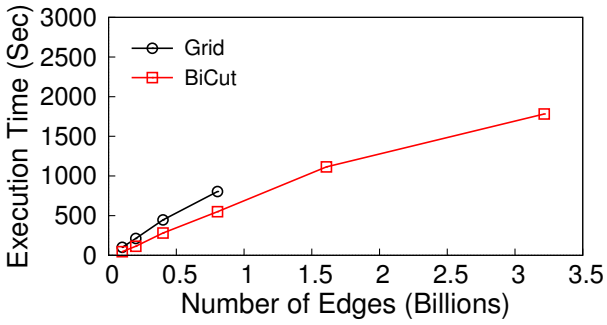

Fig. 7: *A comparison between Grid and BiCut with the increasing graph size.*

5.3 Reduced Network Traffic

Since the major source of speedup is from reducing network traffic in the partitioning and computation phases, we compare the total network traffic of BiCut and Aweto against Grid. As shown in Figure 6, the percent of network traffic reduction can perfectly match the performance improvement. BiCut and Aweto can reduce up to 96% (from 78%) and 45% (from 22%) network traffic against Grid for skewed and balanced bipartite graphs accordingly.

5.4 Weak Scalability

Figure 7 shows that BiCut has better weak scalability than Grid, and keeps the improvement with increasing graph size. For the increase of graph size from 100 to 800 million edges, BiCut outperforms Grid partitioning by up to 2.21X (from 1.46X). Note that using Grid partitioning even cannot scale past 800 million edges on a 6-machine cluster with 144 CPU cores and 384GB memory due to exhausting memory. While using BiCut partitioning can scale well with even more than 3.2 billion edges.

5.5 Benefit of Data Affinity Support

To demonstrate the effectiveness of data affinity extension, we use an algorithm to calculate the occurrences of a user-defined keyword touched by **Users** on a collection of **Webpages** at fixed intervals. The application models users and webpages as two subsets of vertices, and the access operations as the edges from users to webpages. In our experiment, the input graph has 4,000 users and 84,000 webpages, and the vertex data of user and webpage are the occurrences of the keywords (4 bytes integer) and the content of a page (a few ten to several hundreds of kilobytes) accordingly. All webpages are from Wikipedia (about 4.82GB) and separately stored on the local disk of each machine of cluster.

For this graph, Grid partitioning results in a replication factor of 3.55 and causes about **4.23GB** network traffic due to a large number of data movement for webpage vertices, while BiCut has only a replication factor of 1.23 and only causes **1.43MB** network traffic only from exchanging mapping table and dispatching of user vertices. This transforms to a performance speedup of 8.35X (6.7s vs. 55.7s) over Grid partitioning. It should be note that, without data affinity support, the graph computation phase may also result in a large number of data movement if the vertex data is modified.

6 Conclusion and Future Work

In this paper, we have identified the main issues with existing graph partitioning algorithms in large-scale graph analytics framework for bipartite graphs and the related MLDM algorithms. A new set of graph partitioning algorithms, called BiGraph, leveraged three key observations from bipartite graph. BiCut employs a differentiated partitioning strategy to minimize replication of vertices, and also exploited locality for all vertices from the favorite subset of a bipartite graphs. Based on BiCut, a new greedy heuristic algorithm, called Aweto, was provided to optimize partition by exploiting the similarity of neighbors and load balance. In addition, based on the observation of skewed distribution of data size between two subsets, BiGraph was further refined with the support of data affinity to minimize network traffic. Our evaluation showed that BiGraph is effective in not only significantly reducing network traffic and but also resulting in a notable performance boost of graph processing.

7 Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is supported in part Doctoral Fund of Ministry of Education of China (Grant No. 20130073120040), the Program for New Century Excellent Talents in University of Ministry of Education of China, Shanghai Science and Technology Development Funds (No. 12QA1401700), a foundation for the Author of National Excellent Doctoral Dissertation of PR China, China National Natural Science Foundation (No. 61003002) and Singapore NRF (CREATE E2S2).

References

[1] Netflix prize. http://www.netflixprize.com/.

[2] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. In *WWW* (1998), pp. 107–117.

[3] CHEN, R., SHI, J., CHEN, Y., GUAN, H., ZANG, B., AND CHEN, H. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. http://ipads.se.sjtu.edu.cn/projects/powerlyra/PowerLyra-IPADSTR-2013-001.pdf, 2013.

[4] DAVIS, T., AND HU, Y. The university of florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse/matrices/index.html.

[5] DHILLON, I. S. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), ACM, pp. 269–274.

[6] GAO, B., LIU, T.-Y., FENG, G., QIN, T., CHENG, Q.-S., AND MA, W.-Y. Hierarchical taxonomy preparation for text categorization using consistent bipartite spectral graph copartitioning. *Knowledge and Data Engineering, IEEE Transactions on 17*, 9 (2005), 1263–1273.

[7] GAO, B., LIU, T.-Y., ZHENG, X., CHENG, Q.-S., AND MA, W.-Y. Consistent bipartite graph co-partitioning for star-structured high-order heterogeneous data co-clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (2005), ACM, pp. 41–50.

[8] GONZALEZ, J., LOW, Y., GU, H., BICKSON, D., AND GUESTRIN, C. PowerGraph: Distributed graph-parallel computation on natural graphs. In *OSDI* (2012).

[9] JAIN, N., LIAO, G., AND WILLKE, T. L. Graphbuilder: scalable graph etl framework. In *First International Workshop on Graph Data Management Experiences and Systems* (New York, NY, USA, 2013), GRADES '13, ACM, pp. 4:1–4:6.

[10] KOREN, Y., BELL, R., AND VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (2009), 30–37.

[11] KUMAR, A., BEUTEL, A., HO, Q., AND XING, E. P. Fugue: Slow-worker-agnostic distributed learning for big models on big data. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics* (2014), pp. 531–539.

[12] LOW, Y., BICKSON, D., GONZALEZ, J., GUESTRIN, C., KYROLA, A., AND HELLERSTEIN, J. M. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *VLDB Endow. 5*, 8 (2012), 716–727.

[13] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In *SIGMOD* (2010), pp. 135–146.

[14] PROJECT, S. N. A. Stanford large network dataset collection. http://snap.stanford.edu/data/.

[15] ZHA, H., HE, X., DING, C., SIMON, H., AND GU, M. Bipartite graph partitioning and data clustering. In *Proceedings of the tenth international conference on Information and knowledge management* (2001), ACM, pp. 25–32.