BiFennel: Fast Bipartite Graph Partitioning Algorithm for Big Data

Lyu-Wei Wang[1], Shih-Chang Chen[1], Wenguang Chen[2], Hung-Chang Hsiao[3] and Yeh-Ching Chung*[1]
[1]Department of Computer Science, National Tsing Hua University,
Hsinchu, 30013, Taiwan
{lyuwei, shcchen}@sslab.cs.nthu.edu.tw, ychung@cs.nthu.edu.tw
[2]Department of Computer Science and Technology, Tsinghua University
Beijing, 100084, China
cwg@tsinghua.edu.cn
[3]Department of Computer Science and Information Engineering, National Cheng Kung University,
Tainan 70101, Taiwan
hchsiao@csie.ncku.edu.tw

***Abstract*— Graph computing is widely utilized today, which severely requires the ability of processing graphs of billion vertices rapidly for social network analyzing, bio-informational network analyzing and semantic processing. Therefore, graph processing play a significant role in the research and application development. Data of music and movie recommendation and LDA topics can be modeled as bipartite graph and perform the computation with graph processing engines. The most important step before graph computation is graph partitioning. Graph partitioning is a mature technology, however, most of classic graph partitioning algorithms require iterative calculation for several times, which causes high time complexity. Some algorithms with short partitioning time proposed these years, but they cannot be used in bipartite graph directly. This paper proposes a new bipartite graph partitioning algorithm, BiFennel, which effectively decreases graph processing time and network loading by reducing vertex replication factor and maintaining work balance. We implement BiFennel in a popular graph engine called PowerGraph. The performance results show that BiFennel has 29~55% improvement on communication cost and 21~49% improvement on overall runtime comparing with Aweto.**

Keywords— Graph processing; Graph partitioning; Bipartite graph; PowerGraph

I. INTRODUCTION

In recent decade of years, the universalization of World-Wide-Web promotes the booming of websites. According to statistics data of CNNIC in December, 2013, there were 150 billion websites in just mainland China, which increased 22.2% than 2012 [1]. As for the largest social network site (SNS) in the world, Facebook, the user number reached 2.2 billion in July, 2014, which occupied 1/3 population of the world [2]. Messages transferred by users on Facebook reached 12 billion per day and search commands are executed one billion times. Meanwhile, various kinds of SNSs such as Wikipedia, Twitter and Quora attract billions of cyber citizens and generate huge amount of information, which is useful for companies to acquaint their users' requirement, figure out new ideas and implement creative and useful functions for followers.

To better process these continuously, rapidly and massively generated data, graph structure is a proper way to describe, store and handle the metadata. Graph is one of the most common data structure in computer science and technology, which can present more complicated and general relations than linear arrays and trees in terms of structure and semantics. Because lots of realistic scenes can be presented in graph, graph processing and applications are utilized in widely range of areas. Traditionally, graph processing is applied in optimizing transporting routes, predicting path of disease outbreak, relationship of paper citing, and so on. Now, applications such as SNS analysis, semantic web analysis and bio-information network analysis model also process data using graph format. Although graph theory and technology has been well developed, the booming of information enlarges the scale of graph when modeling the metadata into graph data. Correspondingly, the method of efficient and rapid processing for graphs which contains billions of vertices and edges faces severe challenge under this condition.

PageRank [3] is a classic algorithm in search engine, which is developed by Google. In this algorithm, every web page gets a score, which is calculated by all pages it links to. Therefore, web pages are presented with vertices and linking relationships with directed edges. If a graph contains 10 billion vertices and 50 billion edges, and every vertex and its edges occupied 100KB, the whole graph stored in memory will exceed 1TB. Obviously, to store, update, search and process such a huge graph, the time and space consumption is far beyond the ability of traditional centralized graph management strategy.

Cloud computing has advantages for processing graph [4]. Google put forward one of the most popular graph engine for graphs at the scale of billions vertices called Pregel [5] in 2010. Other popular graph engines such as GraphLab [6] developed by CMU, Presto [7] are widely used in distributed graph processing as well.

Engines mentioned above use hash edge-cut method for graph partitioning, which means all vertices are assigned to different machines by hash values and vertices on a same edge

978-1-5090-1893-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SmartCity.2015.153

in different machines communicating with each other via network. This method balances workload by uniformly assigning vertices rapidly, however, the inner structure of the graph is totally ignored. As a result, when there is a k-machines cluster, the ratio of cut edges of a graph will be up to (1 - 1/k), which leads immeasurable communication cost. To relieve this degradation of communication, PowerGraph [8] invented a balanced p-way vertex cut instead of edge cut. Simply, the engine places edges into machines by hash values of source vertices. In this way, just same vertices in different machines communicate with itself to update unified values, instead of transferring huge amount of edge data.

Nevertheless, both vertex and edge cut can decrease network communication and computation runtime with an advanced graph partitioning algorithm which increases localization within a part and reduces connections between parts. This kind of problem is called balanced graph partitioning problem, which is NP-hard [9]. There are some approximation algorithms for this problem, however, it is difficult for algorithms to consider load balance, localization, speed simultaneously.

As a special form of graph, bipartite graph is widely used in machine learning and data mining (MLDM), which divides its vertices to two disjoint parts and every edge join two vertices in the different parts. Typical bipartite graph in large-scale graph processing includes topic-document graph in semantic modeling and user-movie graph in movie recommendation system. Correspondingly, MLDM algorithms such as Latent Dirichlet Allocation (LDA), Alternating Least Squares (ALS) are used in these conditions.

This paper focuses on a new rapid balanced bipartite graph partitioning algorithm and applying to a popular graph processing engine, PowerGraph. The rest of the paper is organized as follows. Section 2 gives the description of important graph processing engines and explains why PowerGraph is chosen to as the performance evaluation platform. It also introduces related work of bipartite graph partitioning of large-scale graph. Section 3 describes the design and an example of the proposed method, BiFennel. In section 4, evaluation of BiFennel is presented and the paper compares it with BiCut, and Aweto. Finally, the conclusion and future works are given in Section 5.

II. Related Work

This section introduces graph engines, which are used for graph computing, and important algorithms for graph partition.

A. Graph Engines

Graph engines aim to improve computation speed by reducing communication cost, balancing workload or some other methods.

- **Pregel** is a calculation architecture mainly for algorithms such as Breadth First Search (BFS), Single Source Shortest Path (SSSP) and PageRank. Google replaces MapReduce [10] by PageRank because of the low efficiency MapReduce performs on graph processing. Pregel is inspired by Bulk Synchronous Parallel (BSP) model [11] which is a model of computation, communication and synchronization. One advantage of Pregel is simple programing model, which just need users define function of vertex calculation and which vertices should be calculated. Another is the whole performance can be predicted. Apache Giraph [12] is an open source version of Pregel. The system is built on Apache Hadoop and is utilized by Facebook.
- **GraphX** [13] is a component in Spark [14] for graph-parallel computation. GraphX extends the Spark RDD by introducing a directed multi-graph with properties attached to vertices and edges. Because of Scala language, GraphX makes graph computing programing simple. It utilizes vertex-cut rather than edge-cut, avoiding huge amount of communication cost. Asynchronous computing mechanism avoids the bottleneck of waiting time in synchronous computing.
- **PowerGraph** proposes a vertex-cut method for graph partition instead of an edge-cut method employed by Pregel. Fig. 1 [15] show the difference between vertex-cut and edge-cut concepts. Edge-cut methods may have low efficiency because of excessive message call. Vertex-cut methods divide a node to a master and mirrors to reduce communication of dense-degree vertices. Vertex-cut method and the mix of synchronous and asynchronous engine makes PowerGraph a fast, low-communication graph-parallel system. Unlike GraphX on Spark, PowerGraph is an independent graph engine which avoid extra efficiency loss. Although independent system makes it harder to interact with other distributed storage system, it still performs well in other aspects.

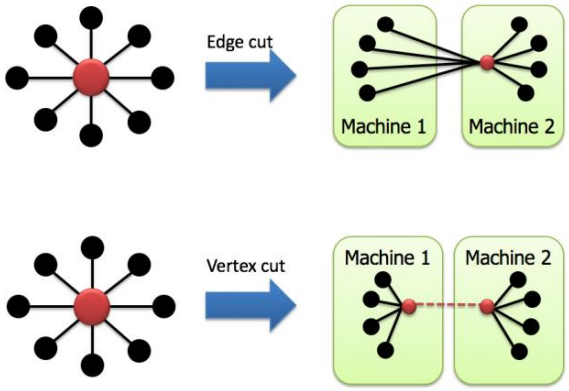

Fig. 1. An illustration of Edge-cut and vertex-cut methods from [15].

B. Graph Partitioning Algorithms

Graph partitioning algorithms aim at increasing localization of subgraphs to reduce communications among graph partitions. To keep scale of subgraphs balanced is also an important issue. However, algorithms provides higher localization and more balanced subgraphs usually cause worse time complexity.

- METIS [16] is a software package which provides balanced partitions and produces low fill orderings for sparse matrix. The algorithms implemented in METIS

are based on the multilevel graph partitioning paradigm including three phases: graph coarsening, initial partitioning and uncoarsening. In the coarsening phase, original input graph is reduced by collapsing vertices and edges to a set of successively smaller graphs, consisting of a maximal size series of adjacent vertices. Graph coarsening keeps processing until the size of the original graph reduced to hundreds of vertices which is called a coarsest graph. Next, in the initial partitioning phase, coarsest graph is partitioned by relatively simple approaches such as Kernighan-Lin algorithm [17]. Since the coarsest graph is consisted of just hundreds of vertices, this step finishes quickly. Finally, in the uncoarsening phase, the coarsest graph is transmitted to the successively and relatively larger graphs by assigning the pairs of vertices. After each transmitting step, heuristic methods will be used to iteratively exchange vertices between partitions to improve the quality of the partitioning result.

- Fennel [18] is a streaming balanced edge-cut graph partitioning algorithm for simple undirected graph. Streaming means vertices arrive while the decision of the assignment has to be done "on-the-fly" for the purpose of little computational overhead. To overcome computation complexity, this algorithm has no iterative processing mechanism. Like other algorithms, Fennel formulates the graph partitioning function to consider two elements: one represents the cost of edges cut and the other one represents the sizes of subgraphs, however, the difference is that Fennel relaxes the hard constraints as compensation. In some experiments, edge-cut proportion of Fennel is slightly more than METIS, which explains that METIS maybe a better graph partitioning tool in terms of edge-cut ratio. However, it takes METIS 8.6 hours to partition a large Twitter graph with more than 1 billion edges while Fennel just need 42 minutes. Although Fennel sacrifices a little edge-cut ratio that increases communication load, it dramatically reduces partitioning time of large-scale graph.

- BiCut [19] is a randomized bipartite graph partitioning algorithm implemented in PowerGraph. It takes advantage of bipartite graph structures, especially for skewed one. In a bipartite graph, there are two subsets of vertices. Vertices on a single edge must be from different subsets, which means vertices in the same subset are disjoint. BiCut avoids replication from one subset that keeps major number of vertices. This will dramatically reduce the replicas of vertices with vertex-cut strategy of PowerGraph. To distinguish the two subsets of bipartite graph, the algorithm represents the larger subset as favorite subset, marked as *V*. Reversely, the smaller one is represented as not favorite subset, marked as *U*. This algorithm contains two steps. First, the vertices of the favorite subset and the edges containing vertices in favorite subset are evenly sent to machines according to hash values of favorite vertices. These favorite vertices introduce no replicas of vertices. Second, the algorithm calls finalize function of PowerGraph to construct local graph on every machine. One replica of a vertex is marked as the master that manages mirrors which represent other replicas of the same vertex. Fig. 2 [19] shows the results of Hash, Grid (a default bipartite partition algorithm in PowerGraph) and BiCut partition methods. BiCut performs better than Hash and Grid partition because of fewer replicas.

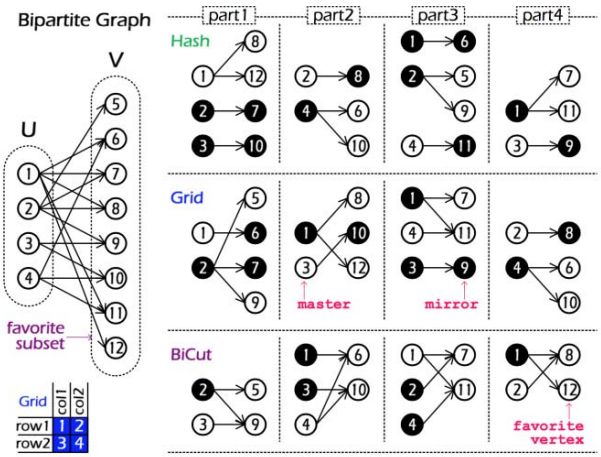

Fig. 2. An illustration of Partitioning results of Hash, Grid and BiCut from [19].

- Aweto, inspired by Ginger [20], is a greedy bipartite vertex-cut algorithm implemented in PowerGraph. Aweto is designed to improve the edge balance of BiCut. First, Aweto partitions the graph by assigning the vertices in the randomized method. After this step, the algorithm reassigns favorite subset to meet the localization of non-favorite vertices. Though, the implementation of algorithm slightly improves the performance of BiCut, it ignores the update of subgraphs when exchanging vertices among machines. Besides, the algorithm fixes the location of non-favorite vertices so that whole localization of graph structure is influenced. Moreover, the implementation utilizes some complex data structure of C++, which makes the program slow down. However, it is still one of the best competitor of all bipartite graph partitioning algorithms.

III. Proposed Method

This section introduces a new greedy bipartite graph partitioning algorithm, BiFennel, inspired by Fennel and BiCut. Throughout this thesis, we use the following definitions:

- A bipartite graph G is separated to k subgraphs nominated as $P=(S_1,\ldots,S_i,\ldots,S_k)$.
- *G* consists of *n* vertices *v* and *m* edges *e*, or favorite subset *V* and not favorite subset *U*.
- $N(v)$ represents the adjacent vertices of *v*.
- $|S_i|$ represents the edge number of partition *i*.

- *Replica* means replicated vertex in vertex-cut algorithm.
- *Replication factor* = 1 + *number of replicas*/(|*U*| + |*V*|)

As mentioned in previous section, Fennel is not designed for bipartite graph and omits the structure of bipartite graph. The replication factor generated by Fennel may be greater than those considering bipartite graph structure. Besides, Fennel is an edge-cut algorithm, which is proved to have more communication cost than vertex-cut algorithms in graph computing stage. BiCut only considers the vertex balance during graph partitioning, however in PowerGraph, edge balance is more important. Besides, as a hash randomized algorithm, the replication factor will be greater than greedy ones after partitioning. Though the speed of randomized partitioning algorithm will be faster, the result of partitioning will slow down the computation stage due to larger replication factor.

Thus, this paper proposes a new heuristic algorithm inspired by Fennel and BiCut, named BiFennel, which utilizes the speed and low replication factor of Fennel and the method that introduces no replicas of favorite subset of BiCut. The algorithm is implemented in PowerGraph, a vertex-cut graph engine, to reduce communication cost.

The steps of BiFennel are described as follows:

- **Step 1**: Load in all graph data and store it in an adjacent list, which represents each favorite vertex's neighbors.
- **Step 2**: Define $\delta g(v, S_i) = |N(v) \cap S_i| - sqrt(|S_i| + |N(v)|)$. Assign each favorite vertex v and its neighbors to partition i in stream to meet the function that for all $j \in [k], \delta g(v, S_i) \geq \delta g(v, S_j)$.
- **Step 3:** Construct local graph and other data structure. Meanwhile, mark master and mirror vertices in PowerGraph record.
- **Step 4:** Call finalize function in PowerGraph to output statistics.

In step 2, the gain function $\delta g(v, S_i) = |N(v) \cap S_i| - sqrt(|S_i| + |N(v)|)$ can be divided into positive gain function $|N(v) \cap S_i|$ that refers to localization of v in S_i and negative gain function $sqrt(|S_i| + |N(v)|)$ which controls edge balance of each partition. In the experiment of this paper, it is proved that edge balance can be less constrained because local computation is faster than communication if memory is enough. Thus, for better localization and less communication, the paper sets negative gain function as 1/2 times square instead of one in Fennel. To optimize $\delta g(v, S_i)$, this paper uses $sqrt(|S_i| + |N(v)|)$ to predict future edge balance rather than $|S_i|$ which represents edge number now in Fennel. Step 2 also utilizes the method of avoiding to replicate vertices in favorite subset to reduce vertex replication factor.

In step 3 and 4, the algorithm runs on every single local machine without any communication with each other. This design guarantees the speed of this algorithm. This algorithm helps construct all data structures of PowerGraph for further processing in PowerGraph.

The pseudo code of BiFennel is presented as bellow.

Algorithm 1 BiFennel Algorithm

Input: Create a k–bit bitmap to save vertices' location in U, named as $neighbor\_map$. Save metadata into adjacent list, named as adj_map.

if (the graph is finalized), **return**.
endif
$proc\_num\_edge[k] \leftarrow 0$
for each v in V, **do**
  create a k -element **vector** $proc\_degree$ to keep $|N(v) \cap S_i|$
  $best\_proc \leftarrow 1$
  $best\_score \leftarrow INT\_MIN$
  for each $v' \in |N(v)|$, **do**:
    for $i = 1$ to k, **do**:
      $proc\_degree[i] \leftarrow neighbor\_map[v'][i]$ //compute $|N(v) \cap S_i|$
    endfor
  endfor
  for $i = 1$ to k, **do**:
    $proc\_score \leftarrow proc\_degree[i] - sqrt(proc\_num\_edge[i] + |N(v)|)$
    if $proc\_score > best\_score$, **Then**
      $best\_score \leftarrow proc\_score$
      $best\_proc \leftarrow i$
    endif
  endfor
  for each $v' \in |N(v)|$, **do**:
    $neighbor\_map[v'][best\_proc] \leftarrow 1$
    $send(best\_proc, edge(v, v'))$ //send all v's edges to appointed machine
    set v' as a master if it is the first time to be sent
  endfor
  $proc\_num\_edge[k] \leftarrow proc\_num\_edge[k] + |N(v)|$
endfor
delete $neighbor\_map$, adj_map //save memory for local graph
Construct PowerGraph local graph: $lvid2record, vid2lvid$
Set $lvid2record.\_mirrors$ //mark all mirrors
$PowerGraph.finalize()$

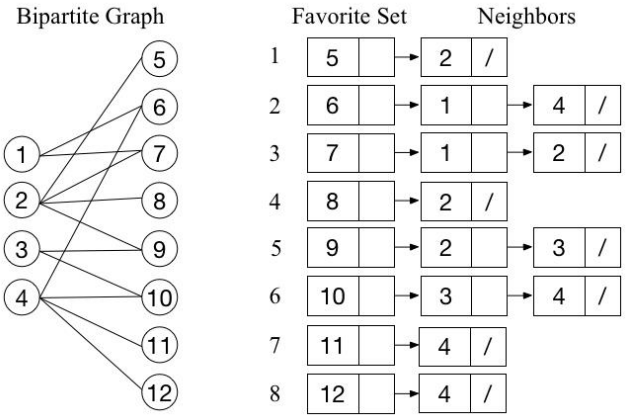

Fig. 3. A bipartite graph and its adjacent list.

Fig. 3 shows a bipartite graph and the adjacent list of the original graph stored by BiFennel in step 1. Step 2 places vertices of favorite set and related links to four machines. After assigning vertices and edges to four parts, Fig. 4 shows the final results with master and mirror vertices.

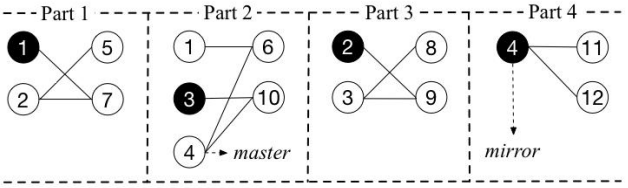

Fig. 4. The graph partitioning result with master and mirror vertices.

IV. Performance Evaluation

BiFennel is implement in PowerGraph 2.2 which is released in Jan. 2015. We compare BiFennel with three existing bipartite graph partitioning algorithms, e.g. BiCut and Aweto. BiCut and Aweto, both developed by SJTU, are the algorithms implemented separately in PowerGraph. Results of replication factor, partitioning time, overall runtime and scalability of these algorithms are also given in this section.

A. Environment Settings

The environment is a virtual cluster with virtual machines (VM) created using QEMU-KVM on two physical machines (PM). Each VM is equipped with 2 cores, 15GB RAM and 1TB storage space while each PM is equipped with 48 cores, AMD CPU, 120GB RAM and 15TB hard disk. A layer-2 1Gbps switch is used to connect PMs. Fig. 5 lists the collection of bipartite graphs downloaded the *Koblenz Network Collection* (KONECT) [21], collected by the *Institute of Web Science and Technologies* at the *University of Koblenz-Landau*, and *Stanford Large Network Dataset Collection* [22] and used as datasets in paper.

Graphs	*|U|*	*|V|*	*Vertex Sum*	*Edge Sum*
Actor-Movie	127,823	383,640	511463	1,470,418
DBLP	1,425,813	4,000,150	5,425,963	8,649,016
LiveJournal	4,308,452	4,489,241	8,893,693	68,993,778

Fig. 5. Collections of bipartite graphs.

B. Performance of Graph Partitioning with Different Graph Data

This section gives performance of BiFennel, Aweto and BiCut with graphs given in Fig. 5. Fig.6 shows the results of normalized graph partitioning time. The time consumption of BiFennel is from 1.27 to 1.6 times of BiCut and is slightly slower than Aweto. It is because that the time complexity of BiFennel is more related to edge number. In detail, the time complexity of BiFennel is $O(n) = n + k * m$, while Aweto is $O(n) = n + k * |V|$. However, Fig. 7 shows, the replication factors of BiFennel are all less than Aweto. Thus the slight time delay is valuable for localization of subgraphs.

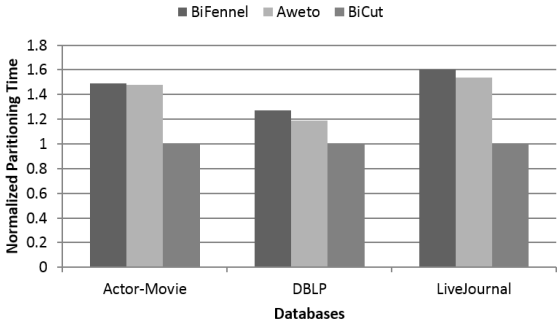

Fig. 6. Graph partitioning time of BiFennel, Aweto and BiCut.

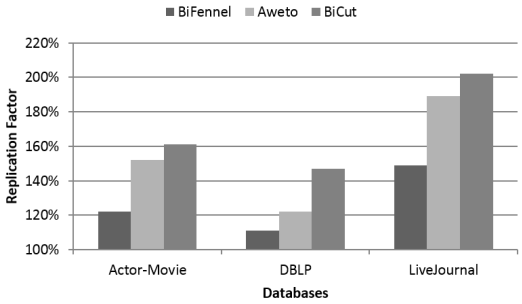

Fig. 7. Replication factor of BiFennel, Aweto and BiCut.

C. Overall Performance

This section compares the proposed algorithm with BiCut and Aweto in aspect of overall runtime and communication cost to show the improvement on graph computing. Singular Value Decomposition (SVD) algorithm, Alternating Least Squares (ALS) algorithm and Stochastic Gradient Descent (SGD) algorithm are used in graph computing stage on a four-node

virtual cluster. In this experiment, actor-movies (AM), DBLP(DB) and LiveJournal(LJ) data are processed for 10 iterations. Fig. 8 shows the normalized communication cost results and BiFennel outperforms its competitors. The normalized performance of BiFennel is 38~62% better than the performance of Aweto. Fig. 9 shows the normalized execution time. BiFennel also has best performance and is 31~53% better than the performance of Aweto. Although Fig. 6 shows BiFennel may has more partitioning time, Fig. 9 shows that low communication cost of the partitioning results of BiFennel can help achieve better overall performance.

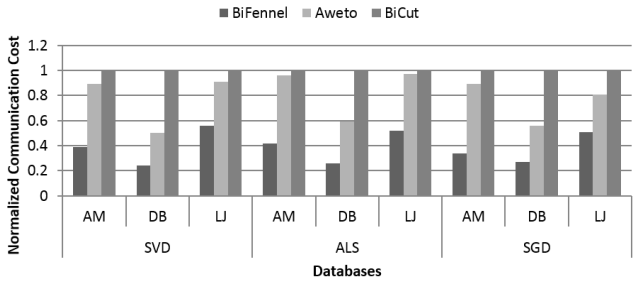

Fig. 8. Communication cost of BiFennel, Aweto and BiCut with SVD, ALS and SGD.

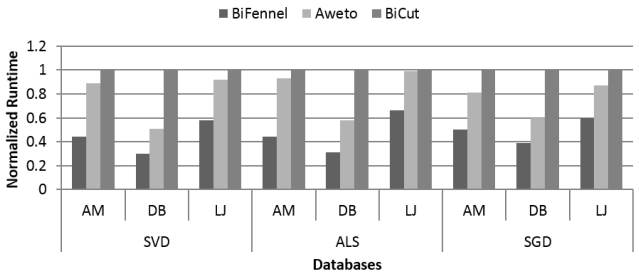

Fig. 9. Overall runtime of BiFennel, Aweto and BiCut with SVD, ALS and SGD.

V. Conclusions and Future Work

This paper proposes a new vertex-cut bipartite graph partitioning algorithm, BiFennel, for large-scale graph data. BiFennel is inspired by Fennel and BiCut, and implemented in PowerGraph. The experiment results show that BiFennel has better performance in terms of replication factor, communication cost and overall runtime. As for future work, some of data structure used in the implementation can be modified for more rapid speed and less memory allocation. Besides, for tasks requiring many computing iterations, the edges on each node can be exchanged for several times with exactly the same gain function to further reduce vertex replicas for faster runtime.

References

[1] http://finance.chinanews.com/it/2014/01-16/5745005.shtml

[2] http://tech.qq.com/a/20140725/000288.htm

[3] S. Brin., L. Page, "The anatomy of a large-scale hypertextual web search engine," Computer Networks, Vol. 56, No. 18, pp. 3825-3833, 17 December 2012.

[4] G. Yu, Y. Gu, Y.-B. Bao, and Z.-G. Wang, "Large scale graph data processing on cloud computing environments," Chinese Journal of Computers, Vol. 34, No. 10, pp. 1753-1765, 2010.

[5] G. Malewicz, M. H. Austern, et al, "Pregel: a system for large-scale graph pro-cessing," In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10), pp. 135-146, ACM, New York, NY, USA.

[6] Y. Low, D. Bickson, et al. "Distributed GraphLab: a framework for machine learning and data mining in the cloud," In Proceedings of the VLDB Endowment, pp. 716-727, April, 2012.

[7] B, N. Bershad, E. D. Lazowska, and H. M. Levy, "PRESTO: A system for object-oriented parallel programming," Journal of Software: Practice and Experience, Vol. 18, No. 8, pp. 713-732, August, 1988.

[8] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," *OSDI*. Vol. 12. No. 1. 2012.

[9] K. Andreev, and H. Racke, "Balanced graph partitioning." Theory of Computing Systems, Vol. 39, No. 6, pp. 929-939, 2006.

[10] J. Dean, and S. Ghemawat , "MapReduce: simplified data processing on large clusters," Communications of the ACM, Vol. 51, No. 1, pp. 107-113, 2008.

[11] L. G. Valiant, "A bridging model for parallel computation." Communications of the ACM, Vol. 33, No. 8, pp. 103-111, 1990.

[12] Apache Giraph: http://giraph.apache.org

[13] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," In First International Workshop on Graph Data Management Experiences and Systems, pp. 2, ACM, 2013.

[14] Spark. http://spark.apache.org

[15] J. E. Gonzalez, A presentation of PowerGraph, https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez

[16] G. Karypis, and V. Kumar, "Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[17] B. W. Kernighan, and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell system technical journal, Vol. 49, No. 2, pp. 291-307, 1970.

[18] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming graph partitioning for massive scale graphs," In Proceedings of the 7th ACM international conference on Web search and data mining, pp. 333-342, 2014

[19] R. Chen, J. Shi, B. Zang, and H. Guan, "Bipartite-oriented distributed graph partitioning for big learning." In Proceedings of 5th Asia-Pacific Workshop on Systems, pp. 14, 2014.

[20] R. Chen, J. Shi, Y. Chen, and H. Chen, "PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs," In Proceedings of the Tenth European Conference on Computer Systems, pp. 1, 2015

[21] KONECT. http://konect.uni-koblenz.de/

[22] SNAP: http://snap.stanford.edu/data/