Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

A heuristic graph partitioning method to minimize remote communication costs

[Shing Ki Wong, Siu Ming Yiu]

***Abstract*—Existing graph partitioning algorithms rarely focus on the choice of the replicas. Most of them make use of random hashing to allocate vertices to partitions due to its convenience and simplicity. However, such random property may result in undesirable partitioning results with huge communication cost if many high degree vertices are being replicated. We address this problem and propose a greedy algorithm to minimize the number of replications of high degree vertices and at the same time minimize the replication factor by sorting the vertices before allocating them to different partitions. We compare our algorithm with one of the well performed existing graph partitioning algorithms, PowerLyra's hybrid-cut, and prove that our algorithm gives better results in most practical situations. Experimental results show that our algorithm gives much lower replication factor compared to PowerLyra's hybrid-cut algorithm generally in random graphs, power-law graphs and real-life graphs.**

***Keywords*—graph, partitioning, heuristic, replica, insert**

I. Introduction

Graph computation has become significantly important nowadays in many areas involving network concepts like the Internet connectivity framework, social network, physical transport network and protein network. By representing individuals in a network with nodes and their corresponding relationships with edges, the network can be transformed into a graph structure, which can then be further analysed by a large amount of calculations and operations in data mining. Among these wide range of graph operations, Graph partitioning is one of the main computations making use of the graph structure representation of network systems. It divides a large graph into smaller sub-pieces, making it more convenient for parallel systems such as MapReduce to manipulate the data in smaller scales. In the era of big data, the size of graph grows exponentially and therefore efficient data placement is essential. Efficient graph partitioning algorithms need to guarantee high data locality to minimize data migration costs, while at the same time maintaining similar sizes for each partition for effective parallel computations.

The logic behind a graph partitioning algorithm plays a very important role in this foundation. It is because poor partitioning results may incur huge overhead and significantly degrade the performance of subsequent graph analysis procedures. In frameworks such as PowerGraph [1] and Giraph [2], the quality of the partitioning result is even more critical.

As shown in Fig. 1, many existing graph partitioning algorithms give less focus on how to replicate vertices in an efficient way by just doing it with random hashing. They give poor partitioning results especially for graphs which contain large number of clusters This kind of graphs is very common nowadays in social networks, where each cluster represents an individual social circle of the community. Furthermore, graph partitioning usually generates replicas in order to preserve the full relationships between vertices after partitioning. Existing algorithms pay little attention to the choice of replicas and may incur huge communication costs by replicating high-degree vertices as they are the most active vertices in the graph.

There are different criteria to determine whether a graph partitioning method is good or not. Common considerations include the time spent and the memory consumed (i.e. how long does the algorithm takes and how big is the partitioned graphs). The time spent usually depends on the complexity of the algorithm, while the memory consumed depends on the number of mirrors created. Because of this, many existing algorithms try to minimize the number of replicas during partitioning (i.e. reducing the replication factor). It is a usual approach and most of the time it gives good partitioning results. Our proposed algorithm also focuses on reducing the number of replicas but we also put much effort in deciding which vertices to be replicated in order to achieve a good partitioning result. Notice that there are differences in the degree of importance for the vertices, which means replicating vertices with different degrees poses different level of communication costs on the whole system. The higher the vertex degree, the higher the cost and vice versa. Therefore, it is better off to avoid replicating high degree vertices during partitioning. However, most of the existing algorithms overlooked this issue since it is not easy to obtain a balanced partitioning with a low replication factor, and at the same time maintaining the connections of vertices as local as possible.

To illustrate the importance of the degree of a vertex, consider an example of an IP network. In the graph of the network, each vertex represents a node and an edge is connected between any two nodes if they are linked with each other. Generally, a node which has more links with others will generate more communication per period of time then a node that with fewer links. Therefore, the communication cost of a high-degree vertex will be higher than a low-degree vertex per unit of time. Because of this, more attention should be put on high-degree vertices than low-degree vertices during partitioning. The same situation also applies in other networks with similar structure such as social networks (e.g. Facebook and Twitter).

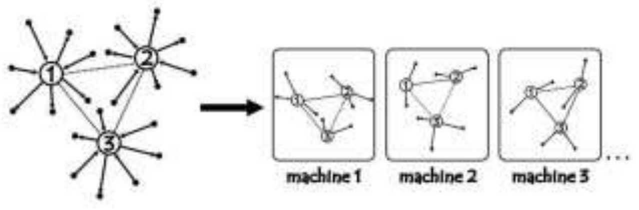

Figure 1. General graph partitioning result using random hashing on vertices.

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

In this paper, we propose a new graph partitioning algorithm, which efficiently assigns vertices into machine using a greedy algorithm to avoid the creation of replicas for high-degree vertices and minimize the communication cost. It gives guaranteed improvement from PowerLyra's hybrid-cut graph partitioning algorithm, which already gives promising performance compared to other existing graph partitioning algorithms.

This paper is structured as follow:

- We discuss the suitable criteria for an efficient graph partitioning algorithm. We state our view and focus on the criteria and illustrate it with real life situation. We also point out the deficiency of existing algorithms which may lead to low performance in graph partitioning in certain situations.
- We introduce our algorithm and state the objectives in behind, then give a detailed explanation on that.
- We prove in theory that our algorithm is guaranteed to have improvement over one of the best algorithms, PowerLyra's hybrid-cut algorithm, under a reasonable circumstance.
- We conduct simulations with randomly generated graphs to verify our theoretic deduction.
- We conduct simulations with large network graphs on our algorithm and PowerLyra's hybrid-cut algorithm and show that our algorithm gives better results.

II. Related Work

The two common approaches for graph partitioning used are edge-cut and vertex-cut. To avoid complexity, many existing graph partitioning algorithms simply make use of hashing to randomly assign vertices and edges into the machines. It is simple and convenient but may result in poor partitioning as it does not pay much attention on the graph structure.

Pregel [3][4] and GraphLab [5] make use of random edge-cut (hashing) to distribute vertices evenly into machines. Pregel only support push-mode algorithms while GraphLab creates replicas and duplicate edges to eliminate this restriction. PowerGraph [1] is a distributed system aiming at balancing the number of edges in different machines to minimize communication overhead. GraphX [6] is another graph processing system on Spark [7] that makes use of vertex-cut to create replicas of vertices in partitioning to support dynamic computations. SBV-Cut [8] introduces the balance vertices concept to achieve a balanced vertex-cut of graphs. SPAR [9] criticizes that most graph partitioning algorithms are not incremental (offline) which will cause huge computational cost for highly dynamic graphs. It focuses on its incremental (online) property which updates the graph according to the six possible events: inserting or deleting on vertices, edges and machines. S-CLONE [10] aims at optimizing the placement of replicas in order to minimize the reading cost of the neighbours. Costs optimizing approach is carried out in [11] to find out the best data placement by considering different aspect of costs in a graph system. A distributed decentralized local searching algorithm is proposed [12] for extremely large graphs which only reads in local information for local operations.

METIS [13] is a multilevel graph partitioning scheme which goes through three phases during partitioning (i.e. coarsening, partitioning of the coarsest graph and refinement). It uses a coarsening heuristic which limit the size of partition to achieve a better result. KaFFPa [14] is another multilevel graph partitioning scheme which makes use of max-flow min-cut computations with local improvements for partitioning, constraining the maximum partition size and minimizing cut size. The parallel version of METIS and KaFFPa are presented in [15] and [16] to produce good partitions of large unstructured graphs a short amount of time. Evolutionary search techniques is adopted in [17] together with multilevel graph partitioner to improve the overall fitness for each generation. Another use of evolutionary algorithm is conducted in [18] to perform local optimization when exploring spaces with standard graph partitioning methods. MITS [19] is another multilevel algorithm which uses tabu search approach during refinement procedure. A parallel genetic algorithm is proposed in [20] which is easy to be implemented in massive architectures. DFEP [21] introduces the concept of edge partitioning and gives a heuristic algorithm to obtain reasonable balanced partitions.

Sheep [22] transforms the input graph into an elimination tree and do partition on the tree which the tree partitions will later be converted back to graph partitions. It scales to larger graphs and gives promising runtime results. It gives promising performance in skewed graphs such as power-law graphs which are very common in real life scenario. However, it only works for undirected graphs, while many of the social network graphs (e.g. Twitter, Facebook) are directed graphs which are not applicable.

For directed graphs, PowerLyra [23] performs pretty good in terms of low execution time and communication cost. It gives significant improvements by proposing a p-way hybrid-cut, which offers differentiated partitioning methods on low-degree and high-degree vertices in order to reduce the replication factor (λ). It pays attention in low-degree vertices to avoid creating mirrors of them. Although PowerLyra offers good results in minimizing the replication factor by reducing the number of mirrors for low-degree vertices, it puts little focus on reducing the number of mirrors for high-degree vertices, which are indeed of higher importance.

III. The Proposed Method

This section explains our concepts behind and gives the detailed algorithm of our proposed graph partitioning method. Our algorithm focuses on minimizing the numbers of replicas for high degree vertices to optimize the partitioning result.

A. *Minimizing High-degree Replicas*

Our main objective is to minimize the number of replicas for high-degree vertices in partitions. Compared to low-degree vertices, high-degree vertices experience more communication per period of time due to its higher number of neighbour vertices. Having more replicas in different partitions implies more remote communications is needed (e.g. network overhead). Hence, replicating high-degree vertices will pose more communication overhead to the system then replicating low-degree vertices. This is the

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

reason why we aim at minimizing the number of replicas for high degree vertices by keeping as many as possible their neighbours with them together locally in the same partition. As shown in Fig. 2, we try to avoid partitioning on the clusters of the graph and put each whole cluster into the same partition so as to minimize the number of replicas of high degree vertices.

B. *Choice of Neighbours*

Our algorithm chooses to assign the target vertices instead of source vertices together with the high-degree vertex. The reason is that it is unlikely that a vertex will be updated by all its source vertices at the same time (Fig. 3 -- left). Therefore, the remote communication cost from the source vertices is usually not maximized in a certain time interval. However, on the other hand, once a vertex is being updated, all of its neighbours will be notified at the same time interval and the remote communication cost will be much higher compared to that from the above case (Fig. 3 -- right). Therefore, it is better to keep the target neighbours locally with the vertex to avoid excessive remote communication cost in a certain time interval. We decided not to keep both target vertices and source vertices together in order to simplify the complexity of our algorithm.

C. *Definition*

To represent the importance of a certain vertex in a graph. We define the *traffic* as the degree (i.e. total number of in-edges and out-edges) of a vertex in the original pre-partitioning graph. The *total traffic cost* (τ) is the sum of all the traffic of the replicas in all machines after the partitioning. This definition reflects the idea that replication of higher degree vertices will incur more communication cost. The lower the cost is, the better the partitioning will be. We also define the *maximum capacity* (C) as the benchmark of the maximum number of vertices allowed to be put into one partition.

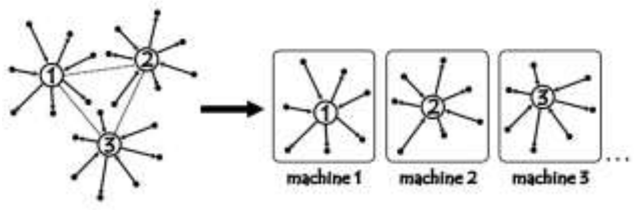

Figure 2. Dividing a graph into different machines in groups of clusters.

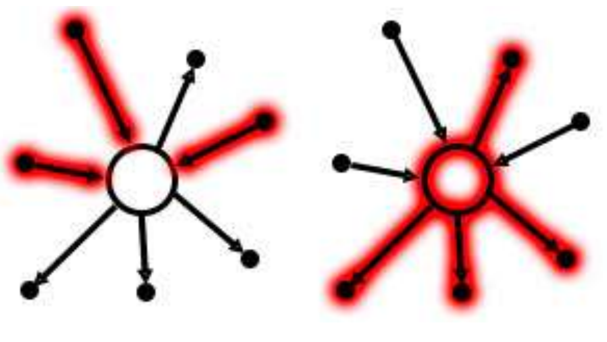

Figure 3. Updates from source vertices(left) and updates to target vertices(right).

D. *The Algorithm*

Our proposed algorithm consist of two steps, it first does a sorting, followed by an allocation of vertices into the partitions accordingly.

[Step 1] Sort the vertices in descending order of degrees.

[Step 2] Starting from the vertex with highest degree, check whether it already exists in any one of the partitions.

- If yes, for low-degree vertices, put all its target vertices into the least occupied machine. For high-degree vertices, fill the least occupied partition with its target vertices. If it reaches the *maximum capacity*, continue assign the remaining vertices into the next least occupied partition.
- If no, put the vertex and all its target vertices into the least occupied partition.

Repeat until all vertices have been inserted.

Sorting the vertices in step 1 costs $O(n)$ time and allocating the vertices in step 2 costs $O(mn^2)$ time, where m is the number of machines and n is the number of vertices. Thus the total time complexity of the algorithm is $O(mn^2)$.

IV. Performance Guarantee

This section gives a theoretic proof on the performance of our proposed algorithm over the hybrid-cut graph partitioning algorithm in PowerLyra.

A. *Guarantee of Lower Traffic Cost*

To guarantee that out proposed algorithm gives a better result, we show that the total traffic cost of our algorithm is lower than that of PowerLyra, that is:

$$\tau_{PowerLyra} > \tau_0$$

Theorem 1. *If more than 37.5% of neighbour vertices are not staying together with the source after partitioning, our algorithm gives a lower value of total traffic cost and thus a better partitioning by creating less replicas on high-degree vertices.*

Proof. We express the total traffic cost of PowerLyra ($\tau_{PowerLyra}$) and our proposed method (τ_0) in terms of the highest number of degree and deduce the required percentage of neighbour vertices are not staying together with the source after partitioning.

Lemma 1. *Total traffic cost for PowerLyra's hybrid-cut*

PowerLyra's hybrid-cut makes use of hashing to assign vertices into partitions. We here assume that the portion of neighbours not staying with a certain vertex (outlying portion) is p after the partitioning. Then the expected number of replicas for a certain vertex of degree k is:

$$p \times k$$

and the total traffic cost of a vertex of degree k is:

$$k(pk).$$

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

Summing up all the degrees from 1 to n we get:

$$\begin{aligned}\tau_{PowerLyra} &= \sum_{k=1}^{n} k(pk)\\ &= p\sum_{k=1}^{n} k^2\\ &= p\left(\frac{n^3}{3}+\frac{n^2}{2}+\frac{n}{6}\right)\\ &= \frac{p}{6}(2n^3+3n^2+n)\end{aligned}$$

Lemma 2. *Total traffic cost for our algorithm*

For our algorithm, since we sort the vertices in advance and assign them in descending order one by one, the maximum number of replicas of a vertex with degree n - k is k until the degree reaches n / 2. After that, the maximum number of replicas will be bounded at n - k for k > n / 2. Summing up all the degrees we get:

$$\begin{aligned}\tau_0 &= \sum_{k=1}^{\frac{n}{2}-1}(n-k)k + \sum_{k=\frac{n}{2}}^{n}(n-k)^2\\ &= n\sum_{k=1}^{\frac{n}{2}-1} k - \sum_{k=1}^{\frac{n}{2}-1} k^2 + \sum_{k=0}^{\frac{n}{2}} k^2\\ &= \frac{n}{2}\left(1+\frac{n}{2}-1\right)\left(\frac{n}{2}-1\right)+\left(\frac{n}{2}\right)^2\\ &= \frac{n^2}{4}\left(\frac{n-2}{2}\right)+\frac{n^2}{4}\\ &= \frac{n^3}{8}\end{aligned}$$

Setting $\tau_{PowerLyra} > \tau_0$, we get:

$$\begin{aligned}\frac{p}{6}(2n^3+3n^2+n) &> \frac{n^3}{8}\\ p &> \frac{6n^3}{8(2n^3+3n^2+n)}\end{aligned}$$

Taking limit for $n \to \infty$ we get $p > 3 / 8$ (37.5%).

V. Evaluation

We implement our algorithm in C++ using The Boost Graph Library 1.60.0 [24]. The hybrid-cut algorithm of PowerLyra is also being implemented for comparison and we assume that vertices with degree more than half of the maximum degree are regarded as high-degree vertices. The maximum capacity is given by

$$n \times \frac{\text{number of vertices}}{\text{number of machines}}$$

where $n \approx 4$ is optimal in terms of minimizing the maximum capacity while at the same time optimizing the partitioning results.

A. *Measuring Criteria*

We judge the performance of partitioning with three criteria in different dimensions, which include the replication factor (λ), outlying portion (p) and epsilon (ε). The meaning of each criteria is mentioned in the following subsections.

1) Replication Factor

The replication factor (λ) is a common measure for partitioning performance. It is given by the average number of replications made for each vertex among all partitions. The lower the replication factor, meaning that fewer replica are created and a smaller size of partitioning, the better the partitioning result..

2) Outlying Portion

The outlying portion (p) is defined by the average percentage of neighbours not staying with each host vertex (the vertex with the highest degree among its replica) after partitioning. It is calculated by

$$1-\frac{\max_{u\in V} deg(u)}{deg(v)}$$

for each vertex v, where V is the set of replica of v among the machines. The lower the outlying portion, the more the neighbours are partitioned with the source vertex in the same partition, and thus the better the partitioning result.

3) Epsilon

The epsilon (ε) is defined in the (k, $1+\varepsilon$) balanced partition problem. It tries to find a minimum cost partition of a graph G into k components with each component containing a maximum number of $(1+\varepsilon)(n / k)$ nodes, which means

$$\max_i |V_i| < (1+\varepsilon)\left\lceil\frac{|V|}{k}\right\rceil$$

where k is the number of machines in the partition. In our analysis, we calculate the corresponding value of ε for each partition result. The lower the value of ε, the more even the distribution of vertices among the partitions, and thus the better the partitioning result.

B. *Random Graphs*

We generate random graphs of vertex size 1000 with edge size 2000, 4000 and 6000 for 100 simulations each. We calculate the average replication factors (λ), outlying portions (p) and epsilon (ε) for both methods for the corresponding graphs with partition number ranging from 5 to 30. The results are shown in Fig. 4, 5 and 6. Generally speaking, all of the replication factor, outlying portion and epsilon increase with the edge size because the partitioning quality decreases with the increase of graph complexity. Our proposed method always gives lower replication factor, outlying portion and epsilon compared to PowerLyra's hybrid-cut algorithm. With the increase of the number of partitions involved, our algorithm gives even better results.

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

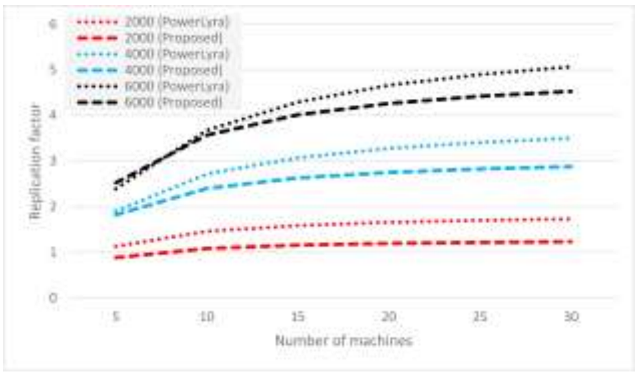

Figure 4. Replication factor of our algorithm and PowerLyra's hybrid-cut in random graphs.

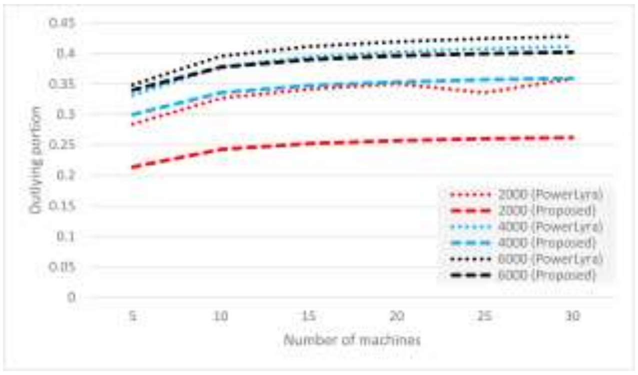

Figure 5. Outlying portion of our algorithm and PowerLyra's hybrid-cut in random graphs.

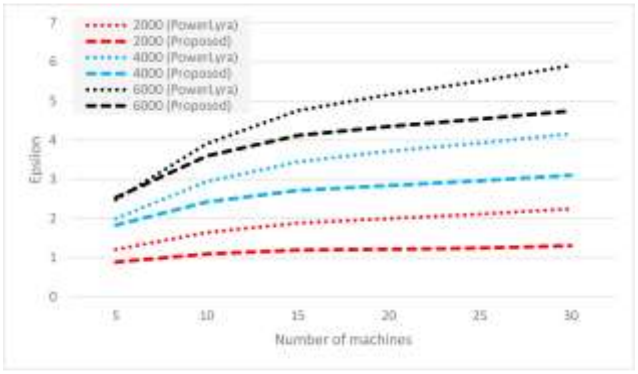

Figure 6. Epsilon of our algorithm and PowerLyra's hybrid-cut in random graphs.

1) Practicability

We can see that in Fig. 5, the outlying portion p for PowerLyra's hybrid-cut keeps increasing with the edge size increasing from 2000 to 6000. The outlying portion reaches 0.4 for $|E| = 6000$ (i.e. around 0.6% of max $|E| = 990000$) when the number of machines equals 10, which already exceed our theoretic p (0.375). It indicates that our algorithm fits into practical situation and outperform in most of the random graphs (≈ 99.4% of graphs).

C. Power-law Graphs

Most of the real-life graphs (e.g. network graphs) are power-law like. To examine our algorithm on such a common type of graph, we generate power-law graphs of vertex size 1000 with edge size 2000, 4000 and 6000 for 100 simulations each. We calculate the average replication factors (λ), outlying portions (p) and epsilon (ε) for both methods for the corresponding graphs with partition number ranging from 5 to 30. The results are shown in Fig. 7, 8 and 9. Similar to our simulations on random graphs, our proposed method guarantees lower replication factor, outlying portion and epsilon compared to PowerLyra's hybrid-cut algorithm. Note that when $|E| = 6000$, the results deteriorates and give less advantage compared to $|E| = 2000$ and 4000. It is due to the fact that the graphs generated with higher number of edges become less power-law like compared to that with lower number of edges. As a result, our algorithm takes less advantage on the power-law property, thus affecting the results.

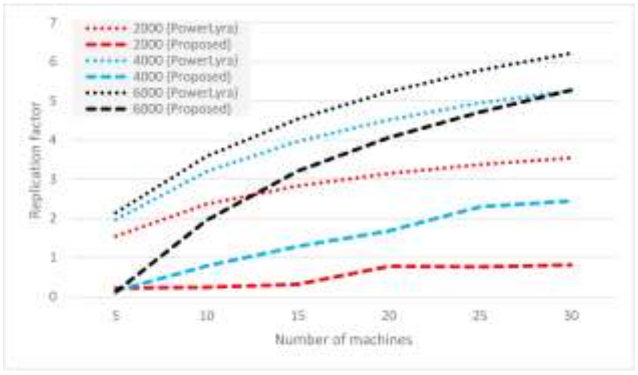

Figure 7. Replication factor of our algorithm and PowerLyra's hybrid-cut in power-law graphs.

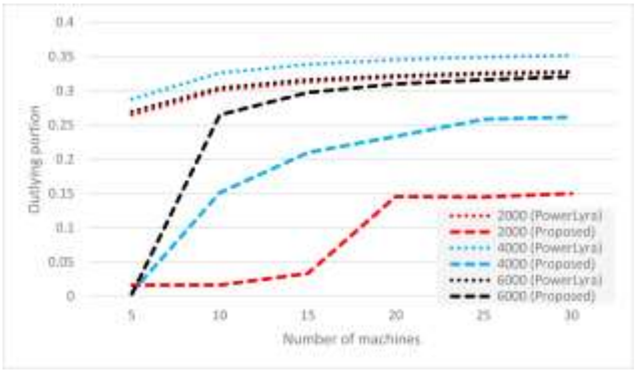

Figure 8. Outlying portion of our algorithm and PowerLyra's hybrid-cut in power-law graphs.

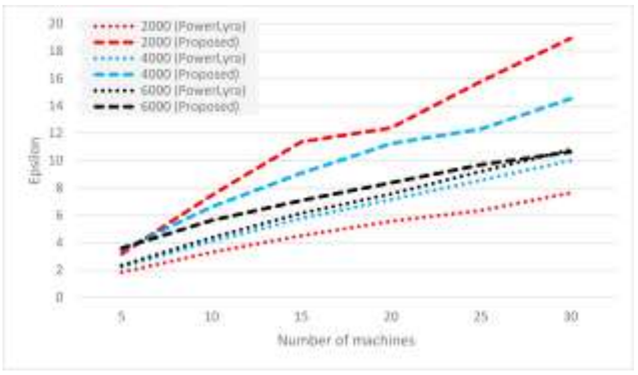

Figure 9. Epsilon of our algorithm and PowerLyra's hybrid-cut in power-law graphs.

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

D. *Real-life Graphs*

To validate the practicability of our algorithm in real life situations, we run the two algorithms with large network graphs in three categories, including social network graphs, peer-to-peer network graphs and signed network graphs. All of the graphs are directed and are obtained from the Stanford Large Network Dataset Collection [25]. We calculate the replication factor, outlying portion and epsilon for each of the graph with partition number ranging from 5 to 30 and take the average for each of the three measurements. Fig. 10, 11 and 12 show the corresponding results.

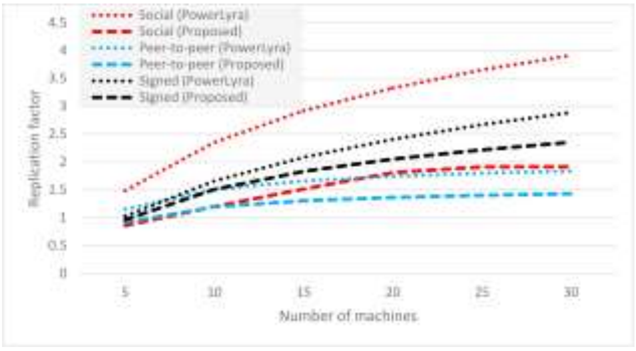

Figure 10. Replication factor of our algorithm and PowerLyra's hybrid-cut in various large networks.

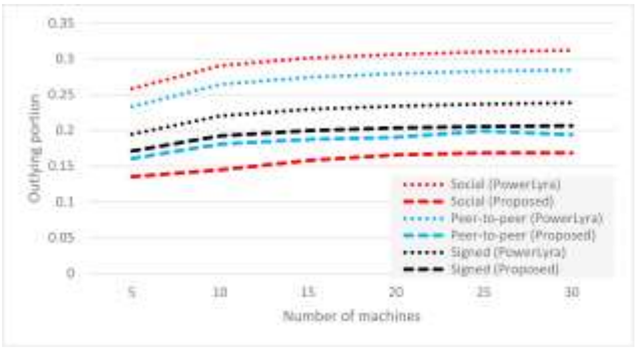

Figure 11. Outlying portion of our algorithm and PowerLyra's hybrid-cut in various large networks.

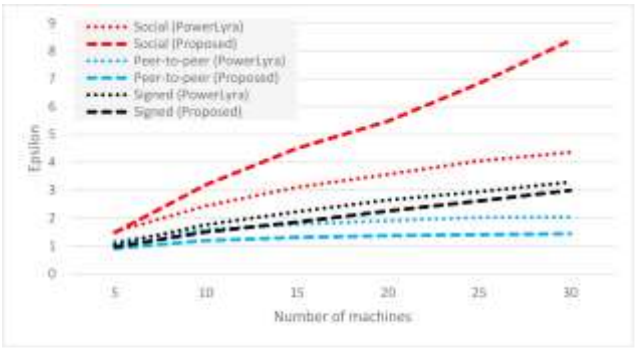

Figure 12. Epsilon of our algorithm and PowerLyra's hybrid-cut in various large networks.

1) Replication Factor and Outlying Portion

As shown in Fig. 10 and 11, our proposed algorithm outperforms PowerLyra's hybrid-cut in all the tested scenarios by giving lower values of replication factor. Results shows that our algorithm gives guaranteed improvement in reducing replication factor on partitioning results. Among the three categories of real-life graph, our algorithm outperforms most in social network graphs, followed by peer-to-peer network graphs and then signed network graphs.

2) Epsilon

As shown in Fig. 12, our proposed algorithm outperforms PowerLyra's hybrid-cut in peer-to-peer graphs and signed network graphs. For social network graphs, our algorithm gives lower value of epsilon. Table 1 gives the breakdown of the average ε for different number of partitions ranging from 5 to 30 of the social network graphs used for simulation. Our algorithm actually performs better for the two Slashdot graphs, while perform worse for Epinions and Wiki-Vote. It is due to the characteristic of our proposed algorithm that aims at putting high-degree vertices with their neighbours together in the same machine, sacrificing the balance of the partitions to minimize the number of high-degree replica in exchange of a low replication factor. The large amount of closely located high-degree vertices in the Epinions and Wiki-Vote network graphs cause our algorithm to put them together in a single machine, resulting in a larger value of ε.

TABLE I. AVERAGE EPSILON FOR SOCIAL NETWORK GRAPHS SIMULATED BY POWERLYRA AND OUR PROPOSED ALGORITHM.

Name	$\varepsilon_{PowerLyra}$	ε_0
soc-Epinions1	2.10	6.99
soc-Slashdot0811	2.94	2.56
soc-Slashdot0922	2.94	2.44
wiki-Vote	4.71	7.93

VI. Conclusion

This paper points out that current existing graph partitioning algorithms overlook the seriousness of creating replicas of high-degree vertices due to the random property of hashing, which may incur high communication cost. Our algorithm tackles the problem by using a greedy approach to minimize the number of replicas for high-degree vertices so that as many neighbours of high-degree vertices stay together with their source vertices as possible. We proved that our algorithm gives lower traffic cost than PowerLyra's hybrid-cut in practical situations. Experiment results show that our algorithm gives lower replication factors than PowerLyra's hybrid-cut for random graphs, power-law graphs and real-life graphs.

Acknowledgment

We thank Rong Chen for providing helpful information on his project of PowerLyra. We also appreciate the constructive and invaluable comments provided by the reviewers on improving the paper.

Proc. of the 10th Intl. Conf. on Advances in Computing, Control and Networking - ACCN 2020
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN No. 978-1-63248-184-9 DOI : 10.15224/978-1-63248-184-9-02

References

[1] Joseph Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In OSDI'12, 2012.

[2] Yuanyuan Tiany, Andrey Balminx, Severin Andreas Corsten, Shirish Tatikonday, and John McPherson. From 'think like a vertex' to 'think like a graph'. In VLDB'13, 2013.

[3] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In SIGMOD'10, 2010.

[4] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel algorithms for graph connectivity problems with performance guarantees. In Proceedings of the VLDB Endowment, 2014.

[5] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning. In UAI2010, 2010.

[6] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In GRADES'13, 2013.

[7] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In HotCloud'10, 2010.

[8] Mijung Kim and K. Selc¸uk Candan. Sbv-cut: Vertex-cut based graph partitioning using structural balance vertices. Data Knowl. Eng., 72:285–303, 2012.

[9] Josep M. Pujol, Vijay Erramill, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The little engine(s) that could: Scaling online social networks. In SIGCOMM'10, 2010.

[10] Duc A. Tran, Khanh Nguyen, and Cuong Pham. S-clone: Socially-aware data replication for social networks. Computer Networks, 56:2001–2013, 2012.

[11] Lei Jiao, Jun Li, Tianyan Xu, Wei Du, and Xaoming Fu. Optimizing cost for online social networks on geo-distributed clouds. IEEE/ACM Transactions on Networking, 24:99–112, 2014.

[12] Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, Mark Jelasity, and Seif Haridi. A distributed algorithm for large-scale graph partitioning. ACM Transactions on Autonomous and Adaptive Systems, 10, 2015.

[13] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 20:359–392, 1998.

[14] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In ESA'11, 2011.

[15] George Karypis and Vipin Kumar. Parallel multilevel series k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48:278–300, 1999.

[16] Peter Sanders, Christian Schulz, Darren Strash, and Robert Williger. Distributed evolutionary graph partitioning. In GECCO '17, 2017.

[17] A. J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimisation approach to graph partitioning. Journal of Global Optimization, 29:225–241, 2004.

[18] Pierre Chardaire, Musbah Barake, and Geoff P. McKeown. A probe-based heuristic for graph partitioning. IEEE Transactions on Computers, 56:1707–1720, 2007.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[19] Una Benlic and Jin-Kao Hao. An eff ective multilevel tabu search approach for balanced graph partitioning. Computers and Operations Research, 38:1066–1075, 2011.

[20] E.-G. Talbi and P. Bessiere. A parallel genetic algorithm for the graph partitioning problem. In ICS '91, 1991.

[21] Alessio Guerrieri and Alberto Montresor. Distributed edge partitioning for graph processing. arXiv preprint arXiv:1403.6270, 2014.

[22] Daniel Margo and Margo Seltzer. A scalable distributed graph partitioner. In Proceedings of the VLDB Endowment, 2015.

[23] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. Powerlyra: Diff erentiated graph computation and partitioning on skewed graphs. In EuroSys'15, 2015.

[24] Jeremy Siek, Lie-Quan Lee, and Andrew Lumsdaine. The boost graph library (bgl), 2016.

[25] Jure Leskovec. Stanford large network dataset collection, 2016.

Appendix

Table 2, 3 and 4 list the details of the large network datasets used for evaluation.

TABLE II. SOCIAL NETWORK GRAPHS USED FOR EVALUATION.

Name	| *V* |	| *E* |	Description
soc-Epinions1	75k	508k	Who-trusts-whom network of Epinions.com
soc-Slashdot0811	77k	905k	Slashdot social network from November 2008
soc-Slashdot0922	82k	948k	Slashdot social network from February 2009
wiki-Vote	7k	103k	Wikipedia who-votes-on-whom network

TABLE III. PEER-TO-PEER NETWORK GRAPHS USED FOR EVALUATION.

Name	| *V* |	| *E* |	Description
p2p-Gnutella04	10k	39k	Gnutella peer to peer network from August 4 2002
p2p-Gnutella05	8k	31k	Gnutella peer to peer network from August 5 2002
p2p-Gnutella06	8k	31k	Gnutella peer to peer network from August 6 2002
p2p-Gnutella08	6k	20k	Gnutella peer to peer network from August 8 2002
p2p-Gnutella09	8k	26k	Gnutella peer to peer network from August 9 2002
p2p-Gnutella24	26k	65k	Gnutella peer to peer network from August 24 2002
p2p-Gnutella25	22k	54k	Gnutella peer to peer network from August 25 2002
p2p-Gnutella30	36k	88k	Gnutella peer to peer network from August 30 2002
p2p-Gnutella31	62k	147k	Gnutella peer to peer network from August 31 2002

TABLE IV. SIGNED NETWORK GRAPHS USED FOR EVALUATION.

Name	| *V* |	| *E* |	Description
soc-sign-epinions	131k	841k	Epinions signed social network
soc-sign-Slashdot081106	77k	516k	Slashdot Zoo signed social network from November 6 2008
soc-sign-Slashdot090216	81k	545k	Slashdot Zoo signed social network from February 16 2009
soc-sign-Slashdot090221	82k	549k	Slashdot Zoo signed social network from February 21 2009

About Author (s):

Shing Ki Wong is a PhD candidate at The University of Hong Kong focusing on researches about game cheat detection, analyzing and identifying cheating behaviors in mobile games.

Siu Ming Yiu is currently a professor in the Department of Computer Science of the University of Hong Kong. His research interests include cyber security, cryptography, and FinTech.