

A Streaming Graph Partitioning Approach on Imbalance Cluster

Yang Cao*, Ruonan Rao**

* School of Software, SJTU(Shanghai Jiao Tong University)

800 Dong Chuan Road, Minhang, Shanghai, China, 200240

alex.nantong@gmail.com, rao-ruonan@cs.sjtu.edu.cn

Abstract—Distributed graph computing refers to extract knowledge by performing computations on large graphs. If the data source is continuously input like stream, the system is called streaming graph computing. When computing large graphs, a basic and significant step is to distribute the graph over a cluster of nodes, which is called ‘partition’. If the graph isn’t partitioned properly, the communication will quickly become a limiting factor in scaling the system, especially in streaming graph computing. And inside some cluster, the CPU speed and memory size of different nodes differs from each other. Observing that in this kind of cluster, nodes those has less resource limit the computing speed, we ask if the partition algorithm could be improved. We propose a simple heuristics to do partition in such cluster and compare the performance of some classic algorithms. It makes less cost of communication more efficient, and make better use of nodes those have more resources. Finally, we evaluate the performance gains in imbalance clusters by using our graph partition method to solve standard PageRank computing on a large real-world World-Wide-Web link graph. It shows that in such circumstance, our heuristics are a significant improvement.

Keywords— Graph Partition, Streaming, Imbalance Cluster, Distributed Graphs

I. INTRODUCTION

Recent years, the data process demand is growing and the size of data is more and more huge. The structures of them are various. But many of them are graph-like data. For example, the web graph is constructed with every page in the web, which has at least one 1.01 billion users (vertexes) on Sep. 2015 [1]. Meanwhile, some ordinary graph computation algorithms, such as PageRank [2], minimum spanning tree, shortest path, community detection [3] and radius computations [4] have become challenging tasks while they are applied to large-scale graph data. Some frameworks have been proposed to solve these problems such as Map-Reduce [5] and Storm [6]. But these universal frameworks encounter problems when applied to graph-like data. The performance is low. To solve this question, a computing model named distributed graph computing was proposed. The solution is to distribute a big graph into smaller partitions. Then use a distributed system that runs on every single node to compute every partition. Finally all partitions together give the final result. There are many real systems, such as Pregel [7], GraphLab [8, 9] and GraphX [10].

The key problem of graph computing is how to partition the whole graph. Many distributed graph systems partition the graph by cutting edges. They assign vertices to machines uniquely, so edges will exist in different computation nodes. The overhead mainly lies on communication of cut nodes. It is positive correlated to the number of edges that are cut. So we can minimize the number of cut edges to reduce the overhead.

However, for most real-world graphs, it is hard and expensive to get a balance *edge-cut*. Especially while dealing with the streaming graph, how to get an optimal edge-cut is a NP-Hard problem [11]. As a result, many graph computation systems use random strategy, which randomly distribute vertices across the cluster. This strategy cannot get a good performance however.

PowerGraph [12] uses a new computation model named *vertex-cuts*, which assigns edges to machines and allow vertices to span multiple machines. In Figure 1 we compare two cut models.

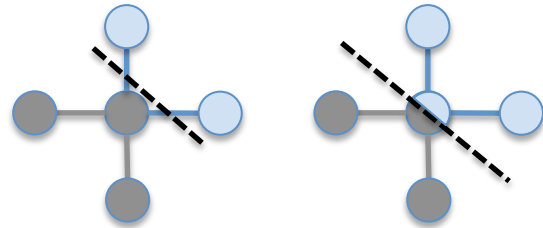


Figure 1. Edge-cut (left) and Node-cut (right). Different colors means they are assigned to different partitions.

The sizes of all partitions need to be balanced to speed up of parallel computing over different partitions. In addition, when computing, if one node is cut, which means the two endpoints are in different partitions, the communication cost will increase because of messages exchanging. So it is critical that the number of edges between distinct partitions needs to be controlled small.

Another challenge is how to process dynamic graphs. Dynamic graphs means the graph is changing as time goes. For example, Google crawls new webpages and updates old webpages every day, then re-calculate the PageRank value of the changed pages. Twitter posts generated by users are also dynamic.. It is important to have an efficient way to partition the dynamic graphs and make the overhead of messages sent across different graph partitions low. The balanced dynamic

graph partition problem setting is known as streaming graph partitioning.

Since that every computation node computes the partition independently, then sends and receive messages to other nodes, every nodes should have similar running speed. If the computation capability differs among the cluster, then the slowest node will limit the overall operating speed. So in such context, the graph partition algorithm is more important.

In this paper, we introduce a greedy algorithm to partition the dynamic large-scale graph with the estimated result of the computation capability and computation speed, which is measured by a dynamic approach we introduced. Then, we use a wide range of datasets and different partitioning algorithms running on imbalance cluster to evaluate our proposed streaming graph partitioning method. The result shows that our method has a better performance and a better trend of time increasing when the imbalance degree increases.

II. RELATED WORK

Streaming graph partition has been discussed for a long time. Before PowerGraph was proposed, most researches are about how to make an edge-balance partition. This problem is that, given a Graph G and a certain number k , how to partition G into k balanced parts, and minimize the edges that are cut. It is a NP-Hard problem [11], but many methods are proposed to get an approximate result within polynomial time. Isabelle and Gabriel summary and compare them in their paper [17]. They showed that *Linear Deterministic Greedy* is the best one-pass algorithm.

Meanwhile, some other papers proposed algorithms to construct spanners, find graph matchings[13, 14] and count triangles in graphs [15, 16]. They stated the one-pass algorithm along with the multi-pass algorithm.

After vertex-cut model was proposed, there are some changes about the problem. Because the researches before this model mainly focused on how to minimize the edge-cuts, not the vertex-cuts. So the algorithm needs to be re-considered.

Also, although the memory size is statically considered in some researches [17, 23], they didn't consider the imbalance condition and the dynamic disturbance of CPU speed. Their algorithms are fit for a ideal cluster, in which the computation nodes has the same calculation speed and the computation process is not disturbed by other processes.

In our paper, we discuss the situation that when the cluster is imbalance, how to make the partition perform better. Here, imbalance not only refers to the capacity, but also the computation speed, mainly related to CPU speed. We proposed a one-pass algorithm. It can be used to help build a framework working on imbalance cluster and solve the reality analysis problems.

III. PROPOSED METHOD

A. Computation Capability Estimation

The capability contains two elements: memory size and CPU speed.

1) Memory size: This element is easy to understand. Every partition stores a partition, which is a sub graph of the total graph. So the size of partition should fit for the size of memory. If one node has a smaller memory size to others, it should not store too many vertexes.

2) CPU speed: In the parallel-graph computation model, the engine always executes some phases in order. For example, PowerGraph has three phrases: gather, apply, and scatter. Each phase is called a minor-step. They run synchronously on all vertices with a barrier at the end. At every barrier, all nodes will sync their status, then do the next step. Therefore, if one node has a faster calculation speed, it will have to wait other nodes finishing their computation. We want the time wasting in waiting shorter.

So we proposed the method that, each node records its situation, including memory size, CPU speed and the time it needs to run one minor-step. These factors will be communicated among all nodes. Each node then calculates a parameter according to these variables.

B. Streaming Partition Model

As we know, to partition a graph to multi ways is a classic NP-hard problem [11]. And the streaming graph partition algorithms just use the information on current status of the cluster, unaware of the future. So the performance depend much on the ordering of coming of edges or nodes. For example, we have a cycle contains 8 edges and two partition. Assuming that our strategy is, to place the edge on the node which has the most intersects the edge, that is to say, counting the number of the two endpoints on all nodes and selecting the most node. If the edges arrive sequentially one by one, the algorithm will lead to 4 adjacent edges on one partition and the other 4 on the other partiton, which cuts 2 nodes. But if the odd edges arrive first, then even ones, the cut of edges would be 4.

Generally, there are three classic streaming orders.

- Random - This is a common order that assumes the edges arrive by random. A more common way is use a hash function to do the partition job. The benefit is we could find the exact partition that one edge belongs quickly.
- BFS - This is an order generated by selecting a starting node from each connected component of the graph, then do a breadth-first search starting at the first node. If there are multiple connected components, the starting nodes are selected by random.
- DFS - This order is similar to the BFS order, but using depth-first search.

Isabelle shows that the three ordering of data arriving does affect the final performance much [17]. So we choose the simplest one, random ordering to do the test.

C. Greedy Partition Algorithm

Our method is to use an objective function, then find the node which make it maximum.

Let P_n^t denotes the n th partition containing nodes at time t . $P^t(e)$ denotes the partition where edge e is contained at time t . $N(e)$ denotes the neighborhood edges of e . S_i^t denotes the computation speed of one minor-step of partition i at time t . $S_i^t = \frac{|P_i^t|}{T_i^t}$, which T_i^t denotes the running time of one minor-step for partition i at time t .

Then $P^t = \{P_1^t, P_2^t, \dots, P_k^t\}$ denotes k -way partition. And our target is to sequentially assign node u to partition i using all information of the current partition. And we propose an objective function:

$$\operatorname{argmax}_{i \in \{1, \dots, k\}} |P_i^t \cap N(e)| \left(1 - \frac{|P_i^t|}{C_i}\right) \left(\frac{S_i^t}{S_{\max}^t}\right)$$

The first consideration is to minimize the node-cut. That is to say, we should find the node that contains more neighborhood edges with edge e . The second is to place the edge to a node that has more capacity. The last consideration is the CPU speed, represented as running speed. Meanwhile, the running situation and capacity is changing every time. So the information should be communicated regularly to make the assignment reasonable.

D. Implement

We implement our method based on GraphX, a distributed graph computing framework based on Spark, which is a universal distributing framework running on Java Virtual Machine.

GraphX has a good architecture. We just need to do three things to implement our method:

- Record the whole capacity, available capacity and running time of every minor-step, store them as a special RDD so that they could be transferred among computation nodes. RDD [18] is called *Resilient Distributed Dataset*, which is a data structure of memory abstraction used in Spark [19]. All frameworks based on Spark must use this data structure.

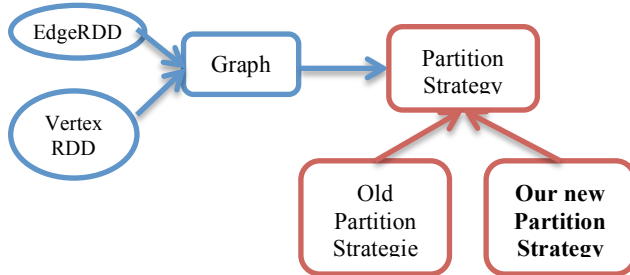


Figure 2. The structure of GraphX. We just need to implement our new PartitionStrategy class.

- Communicating the storage capacity and running time with a certain frequency.
- Implement the *PartitionStrategy* class. This class defines how to determine which partition a new edge should be assigned to. In this class, we calculate the objective function and find the node that makes it max.

IV. EVALUATION

A. Evaluation Setup

We conducted experimental evaluation to discover the performance and trends of our stream partitioning method on a variety of graphs with imbalance clusters. We want to know whether this method could help improve the performance when large-scale graph is computed on imbalance clusters. And as the imbalance degree increases, what's the trend of the node-cuts and computation time.

The imbalance degree should differ from each other and need an index number to standardize. So we proposed an index named *Cluster Imbalance Parameter (CIP)*. This index is defined as:

$$CIP = \sqrt{\sigma_{CPU} \sigma_{Capacity}}$$

Meanwhile, in our evaluation, when we talk about two clusters that have different CIP values or different partition numbers, we make the total CPU speed and the total memory size the same.

We have discussed that the CPU speed is measured by the running time of one minor-step in real system before. However, when calculated statically, so we could use the CPU frequency to express this parameter. Meanwhile, the capacity is measured by the available memory size of every node. The range of CIP is zero or more. The bigger CIP is, the more discrete the cluster is. We use four clusters with different CIP value to examine the performance and trend of our new method.

We used several datasets for our experiments. We pick Wiki-Vote dataset from the SNAP [20] archive. Additionally, we used two classic large social networks (LiveJournal [21] and Twitter [22]) to evaluate our method in real imbalance clusters real system based on GraphX.

We examined all the combinations of datasets, CIP value and partition number. We ran each experiment 3 times on each combination with the random ordering of edges.

B. Evaluation Result

The result is shown in Table 1 and 2.

TABLE 1. EVALUATION RESULT 1

Graph	CIP	Node cuts	Time – random(s)	Time – LDG(s)	Time – Our method(s)
Wiki-Vote	0	0.61	12	12	12
Wiki-Vote	5.4	0.69	12	12	12
Wiki-Vote	15.39	0.73	15	14	13
Wiki-Vote	62.67	0.74	19	17	13
LiveJourney	0	0.31	312	315	320
LiveJourney	5.4	0.41	340	324	325
LiveJourney	15.39	0.45	411	357	336
LiveJourney	62.67	0.49	452	393	348
Twitter	0	0.29	340	346	345
Twitter	5.4	0.38	355	353	350
Twitter	15.39	0.43	384	364	358
Twitter	62.67	0.46	397	379	363

Table I shows the trends of node cuts percentage and the computation time when cluster imbalance parameter increases when the three graphs are calculated on 4 partitions. We compare the classic random partitioning method, the Linear Deterministic Greedy algorithm with our method.

TABLE 2. EVALUATION RESULT-2

Partition number	Time – random (s)	Time LDG (s)	Time – Our method (s)
2	326	310	312
4	340	324	325
8	393	364	360
16	473	402	395

We also evaluated the performance by changing the number of partition. Table 2 shows the result of LiveJourney with the CIP of 5.4 and the partition number of 2,4,8 and 16. It shows that, when partition number increases, the performance will be higher generally. But our method has a lower increasing speed than random method and LDG algorithm.

LDG algorithm considers the memory size difference statically. But when the CPU speed deviation is high, it also has a lower performance. Meanwhile, in our evaluation about different partition numbers, the performance of LDG and our method does not differs much. Because the partition numbers only affects the node cuts probability, but not the CPU speed deviation.

As the result shows, when the all three graphs are computed on imbalance cluster, the normal random partition algorithm lead to a much worse performance when the imbalance parameter increases. However, with our method, the performance keeps relatively stable. It only slows a little, even if the cluster is very imbalance.

V. CONCLUSIONS AND FUTURE WORK

We have presented our simple greedy method to do one-pass streaming partition in imbalance clusters. We formulate an objective function to decide which partition a new edge should be assigned to, which is related to two elements: an element that accounts for the cost of vertexes cost and another element for the computation capability estimated by the CPU speed and memory size. Based on the algorithm, we implemented the streaming distributed graph computing with GraphX, a distributed framework based on RDD.

Then we evaluated the performance of our method when applied on real-world datasets. We showed that with this method, the performance increases by about 10% especially when the imbalance degree is large.

We have identified several possible future research directions. First, the objective function could be improved. It's now a liner function. It may be an interesting problem that how results differs from each other with different objective method. Second, we use GraphX, a distributed framework running on Java Virtual Machine. It is a good framework, but it may exist some deviation on the CPU speed estimated with it. We could try other frameworks to implement it.

ACKNOWLEDGMENT

We thank Huiyi Li, Changyi Yuan and Zhida Zang for helpful comments.

REFERENCES

- [1] <http://newsroom.fb.com/company-info/#statistics>
- [2] Page, Lawrence, et al. "The PageRank citation ranking: bringing order to the Web." (1999).
- [3] Fortunato, Santo. "Community detection in graphs." *Physics Reports* 486.3 (2010): 75-174.
- [4] Qiu, Li, et al. "A formula for computation of the real stability radius." *Automatica* 31.6 (1995): 879-890.
- [5] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.
- [6] Toshniwal, Ankit, et al. "Storm@ twitter." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014.
- [7] Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing.." *Sigmod* (2009):135--146.
- [8] Low Y, Gonzalez J E, Kyrola A, et al. GraphLab: A New Framework For Parallel Machine Learning[J]. Eprint Arxiv, 2014.
- [9] Low, Yucheng, et al. "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud." *Proceedings of the Vldb Endowment* 5.8(2012):716--727.
- [10] Xin, Reynold S., et al. "GraphX: a resilient distributed graph system on Spark." *First International Workshop on Graph Data Management Experiences and Systems* ACM, 2013.
- [11] Andreev K, Racke H. Balanced graph partitioning.[J]. *Theory of Computing Systems*, 2006, 39(6):120-124.
- [12] Gonzalez, Joseph E., et al. "PowerGraph: distributed graph-parallel computation on natural graphs." *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation* USENIX Association, 2012:17-30.
- [13] Ahn, Kook Jin, and S. Guha. "Laminar Families and Metric Embeddings: Non-bipartite Maximum Matching Problem in the Semi-Streaming Model." *CoRR*, abs/1104.4058, 2011. 485 Copyright © SIAM. Unauthorized 2011.
- [14] Ahn, Kook Jin, and S. Guha. "Linear Programming in the Semi-streaming Model with Application to the Maximum Matching Problem." *Lecture Notes in Computer Science* 222.4(2013):59–79.
- [15] Buriol, Luciana S., et al. "Counting triangles in data streams." *PODS '06 Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of databases* (2006):253-262.
- [16] Jowhari, Hossein, and M. Ghodsi. *New Streaming Algorithms for Counting Triangles in Graphs*. Computing and Combinatorics. Springer Berlin Heidelberg, 2005:710-716.
- [17] Stanton, Isabelle, and G. Kliot. "Streaming graph partitioning for large distributed graphs." *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* ACM, 2012:1222-1230.
- [18] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.
- [19] Zaharia, Matei, et al. "Spark: cluster computing with working sets." *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. Vol. 10. 2010.
- [20] <https://snap.stanford.edu/data/>
- [21] <https://snap.stanford.edu/data/soc-LiveJournal1.html>
- [22] <https://snap.stanford.edu/data/egonets-Twitter.html>
- [23] Tsourakakis, Charalambos, et al. "FENNEL: Streaming Graph Partitioning for Massive Scale Graphs." *Proceedings of the 7th ACM international conference on Web search and data mining* ACM, 2014:333-342.