Future Generation Computer Systems 71 (2017) 32–42

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Partitioning dynamic graph asynchronously with distributed FENNEL

Zhan Shi[a], Junhao Li[a,*], Pengfei Guo[b,1], Shuangshuang Li[a], Dan Feng[a,*], Yi Su[a]

[a] Huazhong University of Science and Technology, Wuhan, Hubei 430074, China
[b] qingting.fm, Inc., Shanghai, China

## HIGHLIGHTS

- Streaming graph partitioning is hard to scale because of its sequential natural.
- An asynchronous streaming graph partitioning model is proposed to improve throughput.
- Network utilization can be maximized by proposed tree-shaped map-reduce network.

## ARTICLE INFO

*Article history:*
Received 5 July 2016
Received in revised form
31 October 2016
Accepted 7 January 2017
Available online 14 January 2017

*Keywords:*
Graph partitioning
Streaming
FENNEL
Asynchronous
Tree-shaped map-reduce network

## ABSTRACT

Graph partitioning is important in distributed graph processing. Classical method such as METIS works well on relatively small graphs, but hard to scale for huge, dynamic graphs. Streaming graph partitioning algorithms overcome this issue by processing those graphs as streams. Among these algorithms, FENNEL achieves better edge cut ratio, even close to METIS, but consumes less memory and is significantly faster. On the other hand, graph partitioning may also benefit from distributed graph processing. However, to deploy FENNEL on a cluster, it is important to avoid quality loss and keep efficiency high. The direct implementation of this idea yields a synchronous model and a star-shaped network, which limits both scalability and efficiency. Targeting these two problems, we propose an asynchronous model, combined with a dedicated tree-shaped map-reduce network which is prevail in systems such as Apache Hadoop and Spark, to form AsyncFENNEL (asynchronous FENNEL). We theoretically prove that, the impact on partition quality brought by asynchronous model can be kept as minimal. We test AsyncFENNEL with various synthetic and real-world graphs, the comparison between synchronous and asynchronous models shows that, for streamed natural graphs, AsyncFENNEL can improve performance significantly (above 300% with 8 workers/partitions) with negligible loss on edge cut ratio. However, more worker nodes will introduce a heavier network traffic and reduce efficiency. The proposed tree-shaped map-reduce network can mitigate that impact and increase the performance in that case.

## 1. Introduction

At present, graph processing is applied in many fields, for example, in social networks, graph processing can either be used for security analysis [1] or finding trending topics [2], and in traditional fields such as SSSP (Single Source Shortest Paths) and paper citations, social network analysis [3], data mining [4], protein interactions [5]. The ever growing complexity and scale of various graphs are now posing a big challenge to graph processing. Currently, the indexed Web contains at least 4 billion interlinked pages [6]. On Facebook, there are over 1.65 billion monthly active users which is a 15% increase year over year [7]. Besides, every 60 s, among those friend links, 510 comments are posted, 293,000 statuses are updated, and 136,000 photos are uploaded [8]. Such an unprecedented data deluge brings us not only new opportunities and benefits, but also challenges in computing infrastructure.

Most graph computing tasks, such as Community Detection [9], Connected Components [10], Triangle Counting [11], PageRank [12], Shortest Path [13] and Graph Diameter [14], process graph data iteratively, which makes those tasks formidable to any stand-alone machine when the graph is very large. A traditional method for dealing with this problem is to divide the large graph into several smaller subgraphs, then processing it using a distributed system. These subgraphs must be balanced, so it can take advantage of parallel computing to accelerate processing.

* Corresponding authors.
*E-mail addresses:* zshi@hust.edu.cn (Z. Shi), allenlee@hust.edu.cn (J. Li), 1010382609@qq.com (P. Guo), doublelee@hust.edu.cn (S. Li), dfeng@hust.edu.cn (D. Feng), suyi@hust.edu.cn (Y. Su).

1 Work done while at Huazhong University of Science and Technology.

http://dx.doi.org/10.1016/j.future.2017.01.014
0167-739X/© 2017 Published by Elsevier B.V.

Although a good partition is important for processing graph efficiently, it is also hard to attain. Classical definition of balanced partition problem is to partition a graph in a way that all partitions have roughly the same vertex set size, and minimizes the edges whose two endpoints are in different partitions (cut edges). This problem was proved to be NP hard [15,16]. For years, there are many approximative algorithms been proposed and we will briefly introduce those algorithms next.

Modern distributed graph processing platforms such as MapReduce [17], Pregel [18], PEGASUS [10] and GraphLab [19] by default use hash partition to randomly partition the graph. This strategy is easy to implement, and can make the vertices of the subgraphs well-balanced, but the edge cut ratio goes up to $1 - 1/k$ (k is the number of partitions). In these systems, graph processing has to exchange messages between different partitions along the inter-partition edges. Those messages will travel through the network, which could be very costly if edge cut ratio is high.

Another major challenge for large-scale graph partitioning is how to handle dynamic graph data. The 60 s statistics of Facebook mentioned previously reveals a classical scenario of modern applications, the underlying graph is changing constantly and rapidly, and many real-world graphs share the same feature. Therefore, the processing on these graphs should be fast enough to catch up the change. Stream processing provides a viable solution to this problem. Combining with the idea of stream processing, graph partitioning becomes streaming graph partitioning, every arrived vertex needs to be immediately determined which partition it belongs to.

Given that the goal of large-scale graph partitioning is to generate better partition faster, with limited resources, we aim to use our distributed system to accelerate the graph partition. It is because a single machine has limited resources such as CPUs and memory, so it is necessary for us to use a distributed system.

The remainder of this paper is organized as follows. Section 2 provides a short study on graph partitioning and recent advances on streaming graph partitioning. In Section 3, we present our method and our analysis of partition quality and performance. The experiments results are presented in Section 4. Section 5 contains general conclusions and directions for future work.

## 2. Related work

For years, many researchers have proposed various graph partitioning methods. Spectral method [20,21] converts the graph into a matrix, then use eigenvectors to partition it, however this requires massive computation. Geometric method [22–24] partitions the graph based on geometric characteristics, but suffers a high edge cut ratio. Kernighan–Lin (KL) algorithm [25] starts from a vertex and add its neighbour level by level to the partition until the added vertices reach the half of the whole vertices, and its improved method FM (Fiduccia–Mattheyses) [26] provides an efficient solution to the problem of separating a network of vertices into 2 separate partitions in an effort to minimize the number of nets which contain vertices in each partition. Based on classical algorithms, modern libraries such as METIS [27] adopts a multilevel approach. The main idea is to iteratively coarsen the initial graph by merging vertices, then uncoarsen the graph iteratively with local improvement algorithms such as the KL and FM applied at each level. A multilevel KL-based algorithm [28] is presented as a fast partitioner which allows realtime deployment calculations. Above algorithms are designed for static graphs.

At present, the most common dynamic graph partitioning algorithms are hash algorithm, deterministic greedy algorithm, minimum non-neighbour algorithm. Compared to static graph partitioning algorithms, these graph partitioning algorithms use less computation and do not need the whole information of a graph to determine the partition that every incoming vertex belongs, so the partition would be faster, but the edge cut ratio is higher than the static graph partitioning algorithms. Recently, streaming graph partitioning algorithms [29–32] are proposed to handle massive graphs as data streams. Balanced edge partition [33] has emerged as a new approach to partition an input graph data for the purpose of scaling out parallel computations, which is of interest for several modern data analytic computation platforms, including platforms for iterative computations, machine learning problems, and graph databases. Furthermore, JA-BE-JA [34] is proposed to run partitioning in a distributed graph-processing system, and achieves high parallelism. PAGE [35] is a partition aware engine for parallel graph computation that equips a new message processor and a dynamic concurrency control model. Leopard [36], cooperates with FENNEL, achieves vertex reassignment and replication, can partition dynamic graphs heuristically.

FENNEL [37] was proposed to partition large-scale streaming graph with less computational complexity, which is $O(\log(k)/k)$, where $k$ is the number of partitions or hosts. FENNEL is significantly faster than METIS, and its edge cut ratio is close to METIS. Although modern graph processing systems usually adopt parallel architecture such as map-reduce to handle big graphs [38], to use FENNEL in the same way is not easy. As a stream partitioner, FENNEL processes incoming vertices as one stream, simply running multiple processes or threads is not enough for improving parallelism. Furthermore, while doing greedy vertex assignment, every partition will calculates the cost if the vertex is assigned to this partition, the central machine node will compare the costs of every partition altogether. This structure of the system will be a star-shaped network, with a central node for making decisions, as for network, aggregated data transfer network flow will become a limitation.

## 3. Distributed partitioning

As we have mentioned in Section 2, among streaming graph partitioning algorithms, FENNEL has a better edge cut ratio, even catches up METIS in many cases. But to deploy in a distributed system, for scaling performance, we need to handle the problems from processing and transferring.

### 3.1. Processing model of FENNEL

In a distributed system, by assigning one partition to one worker node, we get a direct implementation of FENNEL processing model. For every newly arrived vertex $v$, a proxy node will broadcast the vertex's data, including its neighbour list to all $K$ worker nodes. The workers will cache that data firstly, and use a greedy vertex assignment algorithm to calculate the gradient $\delta g(v, S_i) = |N(v) \cap S| - \alpha((|S| + 1)^\gamma - |S|^\gamma)$, which gives the outcome if vertex $v$ is allocated to this worker (partition), then return the value to proxy. After proxy have gathered all the returned values $\delta g(v, S)$, it will choose $\delta_{\max} g(v, S_i)$, and broadcast the decided optimal partition $i$ back to $K$ workers. Then, for every worker, it will check whether it holds the optimal partition $i$ or not. If yes, the worker takes corresponding vertex data from cache and puts in local storage. Otherwise, the worker removes corresponding vertex data from cache and puts a key–value pair $\langle v, i\rangle$ into local table for future reference.

But there are two major problems in the processing of above FENNEL model:

1. Low network efficiency caused by synchronous processing. Obviously, for every worker, the process is comprised of three phases: receiving vertex, calculating gradient and sending gradient back, which are completely synchronous. Then, the