Incremental Graph Partitioning Based on MapReduce

An-ming JI, Feng-lan LI, Song-chang JIN and Shu-qiang YANG
College of Computer, National University of Defense Technology, China

Keywords: Hadoop, Incremental graph, Partitioning.

Abstract. Graph partitioning is widely used in many diverse domains. Incremental graph is a member of Graph which evolves with time. Traditional static graph partitioning methods can not meet the requirement with the incremental graph partitioning due to expensive computational cost and long off-line processing time. In this paper, we propose a distributed incremental graph partitioning algorithm to partition the incremental Graphs into smaller components with Hadoop. Our method considers four different modification events of incremental graph partitioning. Moreover, we evaluate our method in Metis and the experimental results show our method is more effective in terms of different measurements than baselines.

Introduction

Graph is an abstract data type which consists of a finite set of vertices and edges. Graph partitioning is applied widely in the field of online social network analysis, transportation networks analysis and biological network research. With the development of web applications, the size of graphs is growing rapidly. Graph data may no longer be stored or processed on a single server. It is necessary to apply distributed clusters to compute various large scale graph. However, using distributed clusters may lead two major problems. The first one is the communication cost problem between clusters, and the other is the balance load (How nodes can be distributed to ensure the load balance of clusters). So some offline static graph partitioning tools are used to partition the graph into smaller components to improve the performance. Metis is one of the mature and excellent tools show strong ability on Graph partitioning [1] [2]. Nowadays, in many cases, the graph structure is no longer static, it will evolves constantly with the time.

Figure 1. Vertex and Edge types of graph partitioned among two partitions.

Related Work

Balanced graph partitioning is an NP-hard problem [5]. In [3], they propose a scalable incremental algorithm for partitioning various TEGs when different modification events are applied to the graph. Firstly, they use the static graph partitioning algorithm to partition the graph. Then, modification will be divided into four kinds of situation, employ different processing method for each kind of case. The results of this method is very good, but they are restricted by the size of the graph, when the scale of the graph becomes larger, the partitioning process will be difficult to carry on. Maini et al. [6] employ Genetic Algorithm to partition graph. Nie et al. [7] [8] using graph analysis to solve social networks identification problem. In [9] [10], the author proposed the parallel incremental graph partitioning,
they use a linear programming-based method to solve the incremental graph partitioning problem. Their calculations are carried out in a single server, and they are still limited by the size of the graph. Kao et al. [11] aims to develop a distributed processing system for solving pattern matching queries on streaming graphs where graphs evolve over time upon the arrives of streaming graph update events. Many popular graph processing platforms such as Pregel [12] that bases on MapReduce [12], and its open source cousin Apache Giraph, PEGASUS [13] and GraphLab [14] [15] use as a default partitioner Hash Partition of vertices, which corresponds to assigning each vertex to one of the k partitions uniformly at random. This platform ignores entirely the graph structure and unsupports dynamic incremental graph partitioning, but these platforms also provide some thoughts. Salihoglu et al. [16] describes the GPS (Graph Processing System) and presents techniques for graph partitioning in distributed graph-processing systems.

**Parallel Incremental Algorithms**

Before introducing this algorithm, i need to reiterate several preconditions and introduce a few concepts and structure. Our graph is connected by undirected edges. The modifications in the graph contain only the changes we mentioned above. Cut edge-the two endpoints of the edge are in the same partition Interior edge. The two endpoints of the edge are in the same partition Interior vertex-Vertex only connected by interior edges Boundary vertex-Vertex which has one or more cut edges. Boundary vertex has two extra attributes: exterior degree and an interior degree. In the Fig.1, the graph is partitioned into two parts. Vertex A is the interior vertex which has no cut edges. Vertex B is the boundary vertex with one cut edges while B has one exterior degree and three interior degrees. Before the start of dynamic partitioning, we partition the original graph into a specified partition by static graph partitioning algorithm, and then transform the partitioned graph into the following structure (we choose Fig.1 as an example). In our MapReduce program, the vertex number is the key. Interior degree, exterior degree, partition number constitute the value. We can calculate the cut edges by the exterior degree and calculate balance load by the partition number.

<table>
<thead>
<tr>
<th>Vertex Number</th>
<th>Interior Degree</th>
<th>Exterior Degree</th>
<th>Partition Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Cut edges:

\[ E_c = \sum_{i}^{n} ED_v / 2 \]

Balance Load:

\[ B_{pi} = \sum_{i}^{n} EW \]

where \( E_c \) is cut-edge rate and \( ED_v \) represents the exterior degree. \( B_{pi} \) is balance load of graph and \( EW \) represents the expected weight.
**Edge Addition**

The edge addition is divided into two cases, when two nodes of the edge are in the same partition, that is, the edge is an inner edge. Because it has no impact on the edge-cut and the balance, so we just do a simple increase operation. As the Figure 1, we increase an inner edge Egh, we should change the related data of Vg and Vh, the interior degree of Vg is changed from 1 to 2. Vh should change too (1 to 2). The other is a more complex situation, we increase the cut-edge. In the Figure 2, we increase an outer edge Ebg, the two nodes of this edge are in different partitions, we increased the edge-cut. In order to reduce the edge-cut, we need to make some adjustments. We should move vertex B from partition1 to the other partition. After moving, the inner degree of the vertex B is greater than the outer degree. In the process of moving, we need to take into account that when the destination partition does not have enough space for holding more nodes, the node will not be moved. Pseudo code as follows:

![Figure 2. Adding edge to graph and migrating node B to partition 2.](image)

**Edge Deletion**

Edge-deletion event can also be divided into two cases. If the edge to be deleted is the cut edge (two nodes of the edge are in different partitions), removing the outer edge will reduce the cutting edge rate. Therefore, we simply remove the edge, not to move the nodes of the relevant operations. We delete the Ebe, the edge-cut is reduced by 1, and we only need to adjust the parameters values of the outer degree of vertex B and vertex E. In the rest of this section, we would discuss the deletion of the inner edge. Reducing the internal edge can not reduce the edge-cut rate, when the composition nodes are not boundary node, we will delete the edge simply, and modify related parameters of the node, without further operation. If one of the nodes of the deleted edge is a boundary node (It is only required to determine whether the outer degree of the vertex is 0). After reducing the edge, if the outer degree (the outer is aimed at a partition) is greater than the inner degree, then this node would require to be moved to reduce the edge-cut rate.

If the destination partition has no room for node moving, then we take no further action. Otherwise, we move the node to the destination and modify outer degree, inner degree, partition number of the node.

**Node Addition**

We suppose we know the relative information about the outer degree and inner degree and the edges’ connection when we are adding the nodes. At the same time, we have to consider the 2 factors which affect the division quality, the balance load and the cut-edge rat. The cut-edge rat is the first to be considered. When we are merging the vertex into one partition, we need to make sure there adds the fewest cutting edges under this circumstance (better adds nothing, which means this point and its contiguous point are in the same partition). After finding out this point, what we are supposed to is to judge whether the destination partition has places for new node. If there has no place, we have to find the worse one and so on, until we find the bigger one. The newly-add node is connecting to the nodes in both partition 1 and partition 2. Just like what we mentioned before, we have known the condition of the newly-add node’s connection to each partition. There will exist 3 cut-edges when the newly-add node joins in the partition 1. And there will be 4 when it joins in the partition 2 as well. We may think of adding node I to the partition1 in order to making lower cut-edge rat. According to the information
of the node’s connection to each partition (the information of the node’s adjacent vertex), we choose the partition which has the most adjacent node as the destination partition. If the cutting edges fail to meet with the balance degree’s requirements, we choose the worse and so on, until we find the destination partition.

**Node Deletion**

As reducing the nodes, we also know the relevant information of the nodes. When a node is removed from the partition, it is also required to remove the associated edges simultaneously. We do not need to consider that deleting points would cause effects on edge-cut because deleting any vertex does not increase the cutting edge rate. It is possible to result in an imbalance load when deleting a point. In other words, the reduction of the nodes makes the partition of the nodes to reduce the imbalance. So we need to move some nodes from high capacity partition to this partition. Now what we need to consider is the chosen of nodes. To solve this problem, we can refer to the process of adding nodes. We still consider to move boundary nodes (in our program, boundary nodes can be determined by the exterior degree parameters of value). Moving a boundary node randomly is not going to work, which may lead to the increase of the cutting edge. As long as the exterior degree of a node is larger than the interior degree, the movement of the node will not increase the numbers of cutting edge. So we give priority to find this kind of points to move, Compare Interior Degree with Exterior Degree.

In our program, when a node is removed, the internal degree and external degrees of the relevant nodes need to be modified at the same time. Because the connection information of the nodes is known when a node is deleted. In the program MapReduce, key is the node number, so we can modify these nodes according to known information. When the modification is completed, we calculate the balance of each partition according to partition number within value, which determines the movement of the vertex.

**Experiments**

**Experiment Environment and Data Set**

In this paper, all the experiments are running on a server (Intel(R) Xeon(R) CPU E5-2403 0 @ 1.80 GHz, memory 64GB) with 64-bit CentOS. And the Metis is running on one server. The dataset is from the real social network. The dataset used in our experiments is Foursquare [17], which has 1063989 nodes and 28199331 edges.

**Experiment Result**

In this section, experiments results are shown to evaluate the performance of Parallel Distribution Graph Partition (PDGP). The results of Metis is the addition of results of incremental networks so that the experiments results can be compared with PDGP. In this experiment, we divide the graph into two partitions with the static graph partitioning algorithm. In order to achieve the purpose of the division of dynamic graph. We first take out the original 50% as the base graph, adding some nodes to the base graph once in a while. Making a record when it reaches 5 percent point (mainly record time, cut-edge and balance load).

![Figure 3. The Result of Adding Node.](image-url)
Extensive empirical study shows our algorithm works well. From Figure 3(a) and Figure 3(c), we can see that Balance Load of the Metis and PDGP changes are similar. However, the edge-cut ratio of PDGP is higher than Metis during partition process. The reasons are as follows. We do not know the global information of the graph, we can only take the local optimal solution. From Figure 3(b) we can find that the time consumption of our algorithm is lower than that of Metis. Because when the motivation event happens, we do not need to re-partition the graph, and only need to adjust the local of graph, so the running time is less than Metis. And our algorithm supports a larger graph, while Metis not.

![Figure 4. The Result of Deleting Node.](image)

Similar to the experimental results above. The edge-cut ratio of PDGP is higher than Metis. While Balance Load of the Metis and PDGP changes are similar. From Figure 4(b) we can find that the time consumption of our algorithm is also lower than that of Metis.

**Conclusion and Future Work**

The division of the increment graph has very important realistic significance. The partitioning algorithm of the distributed incremental graph which presented by us, not only basically solved the problem of dynamic graph division, but also made the division process out of the scale limit. According to the experiment, the result of our method is good. In the future work, we’ll keep improving the quality of partitioning and think about the butterfly effect caused by increasing or decreasing the nodes (like it needs or not to divide the zone into two if we increase more nodes).

**Acknowledgments**

The work described in this paper is partially supported by this work is supported by national key fundamental research and development program (973 program) No.2013CB329600, 2013CB329606 and National Natural Science Foundation of China No.61372191, 61402511. The author is grateful to the anonymous referee for a careful checking of the details and helpful comments that improved this paper.

**References**


