A revised implementation of Lazy-K partitioning algorithm for Graph/z

Moudgalya, Shreyas  
smoudgal@hawk.iit.edu

The emerging applications in big data science and social increasing demands on large-scale graph-structured processing has led to the development of several graph-parallel abstractions including GRAPH/Z. However, in the case of using ZHT, a scalable distributed key-value store as building block of GRAPH/Z, the distribution is forced to use hash-based (random) partitioning which has potentially impaired their locality. The goal of this proposal would be to develop a graph partitioning scheme that can be used in the loading process of GRAPH/Z, a scalable graph processing system, to overcome several problems of its performance.

2

# Background Information

Graph partition problems fall under the category of NP-hard problems. Solutions to these problems are generally derived using heuristics and approximation algorithms. However, uniform graph partitioning or a balanced graph partition problem can be shown to be NP-complete to approximate within any finite factor.

There have been a lot of work to handle big datasets of both commercial and scientific applications, including work flow systems, data streaming management systems and graph databases. These systems typically save the data in distributed file systems(such as Hadoop HDFS and FusionFS), SQL databases(such as Oracle and DB2), and NoSQL database(such as Cassandra and ZHT).

Graph related query is tremendously slow on the traditional relational database, which makes it even more challenging to fully reveal and utilize the scientific and commercial value from the continuously increasing graph data sets. An ideal solution for this problem is to replace the traditional data infrastructure with a graph-centric model, including storage and computing, thus to better serve graph-based applications in terms of performance and programmability.

However, since natural graphs are difficult to partition, the old GRAPH/Z system used ZHT’s hash function to physically distribute all the vertices. As a result, vertices and edges, which are the basic elements of graphs, are spread around different nodes. Due to the random partitioning scheme, when each vertex needs to get access to its edge list, it will statistically communicate to another node and make the communication cost become tremendously higher. Therefore, using hash function to partition the graph data can be considered as a worst case, which is determined by the characteristic of graph structure and the hash function. The original use case of hashing partition is dispersing the data to avoid hot spot, but it doesn’t help in the case of graph processing system.

# Problem Statement

The Design and Implementation a Partitioning Scheme that works well on the large graph datasets on Graph/Z. The design and implementation of the algorithm must implement a fast, balanced and lightweight graph partition scheme based on graph traversal.

# Related Work

Various Graph Partitioning Algorithms exist. Since graph partitioning is a hard problem, practical solutions are based on heuristics. There are two broad categories of methods used. They are local and global. Well known local methods are the Kernighan–Lin algorithm, and Fiduccia-Mattheyses algorithms, which were the first effective 2-way cuts by local search strategies. Their major drawback is the arbitrary initial partitioning of the vertex set, which can affect the final solution quality. Other Global approaches rely on properties of the entire graph and do not rely on an arbitrary initial partition. The most common example is spectral partitioning, where a partition is derived from the spectrum of the adjacency matrix.

A multilevel graph partitioning algorithm works by applying one or more stages. Each stage reduces the size of the graph by collapsing vertices and edges, partitions the smaller graph, then maps back and refines this partition of the original graph. A wide variety of partitioning and refinement methods can be applied within the overall multi-level scheme. In many cases, this approach can give both fast execution times and very high quality results. One widely used example of such an approach is METIS, a graph partitioner, and hMETIS, the corresponding partitioner for hypergraphs.

# Proposed Solution

The main work will involve the implementation of a graph partitioning algorithm. We would want to implement a multilevel graph partitioning algorithm in conjunction with the breadth first search. We try to control the size of partitions by grouping the vertices into a super vertex thereby collapsing the graph. This grouping as well as condensing the graph approach reduces the time since we don’t need to know the picture of the large graph to begin with. The grouping process starts from the vertex which has the most numbers of neighbors. We shrink the vertex into a super vertex with the minimum edge cut. We partition the graph with breadth first search and ensure that each partition size reaches a configurable threshold. This ensures good load balance. The basic process of the algorithm is summarized as follows:

1. Sort the vertex list based on the numbers of neighbors.
2. Create Super vertex for each of these vertices and ensure minimum edge cuts.
3. Partition the graph into two equally weighted halves using BFS algorithm
4. Go back to 3 to recursively partition to attain load balance across n-nodes.

# Evaluation

Profiling and comprehensive performance evaluation of the different data sets on the newly implemented graph partitioning algorithm in order to compare its edge cuts, load balancing with other partitioning algorithms such as the Dynamic Cut-Cluster Algorithm, EvoPartition Algorithm, and the Kernighan–Lin Algorithm. We would also profile the running time of the different algorithms along with the newly implemented partitioning algorithm based on increase in graph vertices and increase in graph edges.  
We will use a graph partitioning algorithm defined as:  
Given a graph and a a set of partitions, , an output partition such that

1. are disjoint
2. are roughly balanced
3. let   
   Minimize

# Timeline with Weekly Goals

**Week 1:** Learn on Multilevel Graph partitioning and Coarsening Techniques  
**Week 2:** Learn Uncoarsening Techniques  
**Week 3**: Implement Proposed Graph partitioning Algorithm  
**Week 4:** Implement Proposed Graph partitioning Algorithm  
**Week 5:** Implement Proposed Graph partitioning Algorithm  
**Week 6:** Evaluate the Graph Partitioning Algorithms  
**Week 7:** Evaluate the Graph Partitioning Algorithms

# Deliverables

Test the newly proposed partitioning algorithm on a single node on an instance of AWS EC2 using the dataset from Stanford Network Analysis Project.

# Conclusion

By using the newly proposed partitioning algorithm on GRAPH/Z, we hope to achieve better performance of the GRAPH/Z system.