Efficient MapReduce Algorithm for Counting Triangles in a Very Large Graph

Temitope Ajagbe
University of Texas at San Antonio

Introduction
The paper proposed a more efficient method to find number of triangles in a graph whose number of vertices and edges will not fit in internal memory of a compute node. The algorithm proposed improves on existing Graph Partitioning algorithm that reads and allocates graph node into a finite number of partitions. Each partition maps to a compute node . To find the number of triangles in such a large graph , MapReduce programming model provides a divide and conquer approach to the input data and then iterative reduction in the problem size until some definite solution is arrived at in a reduction step.

Related Work
Algorithm to count triads in a graph by partitioning the graph and counting the triads in each partition was the basis for this work. The partition assumes that a triangle within the graph can either be found within a partition (Type-1) , the nodes may span two partitions (Type-2 triangles) or may span 3 partitions (Type-3) . These triangles are collected and then counted to avoid duplicity in counting especially across partitions.

MapReduce uses high-level Map , Shuffle and Reduce steps to solve a typical problem. Data will be assigned to various partitions which will correspond to a task task that mapper will work on.
High network overhead may result as data may be moved across partitions during shuffling. Shuffling is a technique in MapReduce model that allows tasks that are processed by mapper to be combined and sent to the reducer. As the number of reductions progress, the results is expected to converge. The solution provides an implementation of the Map function that uses a partitioning functions called a hash function that ensures each node gets assigned to a unique partition.

Conclusion
The conclusion of the paper is that the algorithm runs in O(m^3/2) which is the best running time for the existing algorithm. Even though this is not better in terms of performance, but it provides a mechanism to scale theoretically arbitrarily as there are number of nodes. The partition size is the number of computing nodes available in the cluster. The assumption is that the ratio of the nodes to the partition will will fit in internal memory of a single computing node in the cluster.

Proposed Idea
I propose improvement to this solution by looking at heterogeneous cluster with uneven memory capacity per node with weighted connection latency between nodes in the network and having an excessively large graph. Excessively large graph (ELG) will be considered to be one that K exceeds the internal memory capacity of largest internal memory of any node in the cluster. K is defined as the Number of nodes in the graph/number of computer nodes in the cluster. The objective of my proposal is to achieve same or better asymptotic performance of execution using the new algorithm or technique.