Clustering Algorithms for Large-Scale Graphs Using MapReduce

Chris Joseph
Dept. of Computer Science
Stony Brook University
Stony Brook, USA
chris.joseph.1@stonybrook.edu

Stephen Krucelyak, Enyue Lu
Dept. of Math. & Computer Science
Salisbury University
Salisbury, USA
ealu@salisbury.edu

Jack Slettebak, Matthias Gobbert
Dept. of Math. & Statistics
Univ. of Maryland Baltimore County
Baltimore, USA
gobbert@umbc.edu

***Abstract* — Large-scale graphs representing data from real-world networks are usually non-uniform and contain underlying structures. Graph clustering, a process for identifying these structures, has many applications. Given the massive sizes of modern graph data sets, which consist of millions or billions of vertices and edges, it is difficult or even impossible to process them on a single computer. In this paper, we take advantage of MapReduce, a programming model suitable for processing large data sets, for graph clustering. The aim of our paper is to demonstrate efficient and scalable implementations of clustering algorithms using MapReduce and analyze their performance on the distributed cloud computing platform Amazon EC2. Three different types of graph clustering algorithms are considered: graph theory-based k-trusses algorithm, physics-based Barycentric clustering algorithm, and probability-based Markov clustering algorithm. We also compare the scalability, effectiveness and accuracy of these algorithms for identifying clusters in various data sets acquired from Stanford SNAP.**

Keywords-MapReduce; Clustering Algorithms; Graph Algorithms; Parallel Computing; Cloud Computing

I. INTRODUCTION

Clusters are graph structures that represent similar data entities. By identifying clusters in large data sets, one can get insights about the data and find valuable information. For example, the tendency of data entities to be clustered together in most real-world data sets can be used to identify communities, analyze blogs and detect anomalies.

While many clustering algorithms already exist, they are often inefficient for analysis on modern data sets due to their massive sizes. MapReduce, a distributed parallel processing system using a large number of computers (nodes) on the Internet (cloud), can be applied to solve large-scale computing problems. Using the open-source Hadoop implementation of the MapReduce model, programs can be automatically parallelized and executed on a cluster of computers [5]. In this paper, we use MapReduce to implement clustering algorithms on distributed computing platforms such as Amazon Elastic Cloud Compute (EC2) [1], allowing these algorithms to scale to the sizes of real-world data sets.

II. MAPREDUCE CLUSTERING ALGORITHMS

A universal clustering algorithm for all types of data and applications doesn't currently exist. In this paper, we focus on three different types of graph clustering algorithms: graph theory-based k-trusses algorithm [3], physics-based Barycentric Clustering (BC) algorithm [3], and probability-based Markov Clustering (MCL) algorithm [4]. We implement all clustering algorithms using MapReduce. Every MapReduce algorithm consists of one or more *MapReduce iterations*, in which each iteration has a *Map* phase and a *Reduce* phase. In the Map phase, the master node partitions input data and assigns tasks to worker nodes. In the Reduce phase, the information generated by worker nodes is aggregated to produce the final output.

A. k-Trusses

A truss is a highly connected graph that acts as a relaxation of a clique. Whereas a clique requires all vertices to be connected to each other, a k-truss requires all vertices to belong to k-2 or more triangles. According to [3], trusses capture many of the characteristics of cliques without being too restrictive, since cliques rarely appear in real-world data sets. Our implementation of the k-trusses algorithm is based on [7], which uses five MapReduce iterations.

B. Barycentric Clustering

BC is a physics-based algorithm that uses the ball and spring model, in which a graph's vertices are considered as balls and its edges as springs. According to Hooke's Law, every spring exerts a force in direct proportion to the distance that it is stretched or compressed. If the ball and spring structure is initially stretched out and then released, balls that are mutually connected by many springs will naturally clump together. This notion can be used to identify clusters.

The first step of BC algorithm is to assign a random position to each vertex in space using standard normal samples. In the next step, the "releasing" of the balls (vertices) is simulated for several iterations (five iterations are sufficient, based on [2]); in each iteration, the force felt on each vertex due to its neighbors is computed, and its position in space is then updated using a weighted average of its old position and the computed force. At the last step, a neighborhood average length is calculated for each edge, and any edge exceeding this length is removed, leaving behind components of the graph that are distinct clusters. Because the initial positions of vertices are randomly generated, 30 random initial positions (which is sufficient to minimize statistical variations as suggested in [2]) will be generated for each vertex in the first step and computed

This work is funded by NSF CCF-1460900 under Research Experiences for Undergraduates Program. Joseph and Krucelyak did their work as REU (Research Experiences for Undergraduates) students at Salisbury University during the summer of 2015.

in parallel during the following steps. Our implementation of BC algorithm requires a total of six MapReduce iterations.

C. *Markov Clustering*

MCL is a popular clustering algorithm based on the simulation of stochastic flow in graphs. Intuitively, one can think of a graph's edges as roads and vertices as junctions. Upon arriving at each junction, if the next move is chosen randomly, after many such random walks, one is more likely to keep walking within their cluster than they are to leave it. This tendency can be used to determine the clusters in a graph by simulating random walks.

Initially a matrix *M* is created that contains undirected edges between vertices. Each column of *M* is normalized, so that entry *M(i, j)* represents the probability of randomly walking from vertex *j* to vertex *i*. Next, two stages called *expansion* and *inflation* are performed repeatedly until the matrix converges. Expansion refers to multiplying *M* by itself, which simulates the next time step of the random walk. Inflation refers to raising every value in *M* to a power *p* (known as the inflation parameter), and then normalizing each column. The value of *p* controls the granularity of the resulting clusters, according to [4]. Once the matrix converges, two vertices *j* and *k* are in the same cluster if and only if, within a row *i*, the value *M(i, j)* = *M(i, k)*. The implementation of matrix multiplication in the expansion stage is based on [6]. Our implementation of MCL algorithm requires six MapReduce iterations.

III. EXPERIMENTAL RESULTS AND ANALYSIS

In order to test our algorithms, we utilized two different testing environments. Initial testing was completed on UMBC's High Performance Computing Facility, so that we could ensure the correctness of our programs. We then tested the scalability of our algorithms on virtual machines created using Amazon Elastic MapReduce and managed within Amazon EC2. Each virtual machine was fitted with the equivalent of eight 64-bit virtual CPU cores, 7GB of memory and a minimum of 100Mbps network bandwidth.

Figure 1 shows the scalability of our BC algorithm on various sized Hadoop clusters on the Google+ graph obtained from Stanford SNAP [8], which contains 107,614 vertices and 13,673,453 edges. We found that small sized clusters are ill-fitted for overcoming the bottlenecks associated with Hadoop. Once we increased the cluster size, we were able to reduce runtime from over 100 minutes (on one and two computers) to roughly 15 minutes (on 16 computers). Given this trend we believe that larger clusters will reduce clustering algorithms' run time further.

Figure 1: Scalability on Multiple Computers

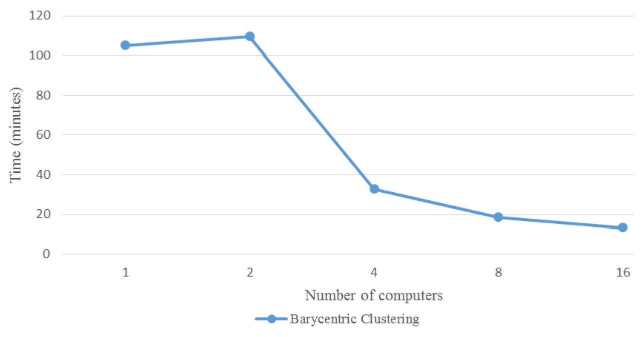
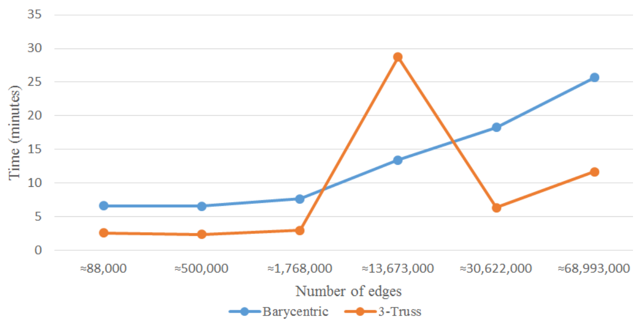


Figure 2 shows the scalability of our implementation of BC and k-truss clustering algorithms on six various sized datasets (ego-Facebook, soc-Epinions1, ego-Twitter, ego-Gplus, soc-Pokec and soc-LiveJournal1 from [8]) using 16 computers on Amazon EC2. From the figure, we learned that both BC and k-trusses algorithms scaled well for larger datasets, since the overall run time was barely affected when we increased the size of the data by millions of edges. We also found that the BC algorithm handled data more consistently, as 3-truss experienced a spike when running on the Google+ (ego-Gplus) graph. The spike was caused by the ratio of vertices to edges being much higher in that particular dataset, resulting in a relatively large number of triangles.

Figure 2: Scalability on Various Data Sizes



IV. ONGOING WORK & FUTURE RESEARCH

Our current MapReduce implementation of MCL converges slowly on large data sets due to large-size matrix multiplication. We plan to investigate the optimal parameters for the matrix block size and the inflation parameter in matrix multiplication to make MCL algorithm practical for large-size data sets. We are also interested in analyzing the effectiveness of the clustering algorithms. We plan to measure the quality of the generated clusters using methods such as comparing modularity, maximum distance (length of the shortest path), and integrated classification likelihood of clusters. Furthermore, we plan to investigate efficient implementations of these algorithms on newer big data processing platforms such as Apache Giraph, Mahout, and Spark.

V. REFERENCES

[1] Amazon Web Services: http://aws.amazon.com [Last Accessed: 7-31-2015]

[2] J. D. Cohen, "Barycentric Graph Clustering," 2008 http://csee.ogi.edu/~zak/cs506-pslc/barycentric.pdf [Last Accessed: 7-31-2015]

[3] J. D. Cohen, "Graph Twiddling in a MapReduce World," Computing in Science & Engineering, vol. 11, no. 4, pp. 29-41, July-Aug. 2009

[4] S. V. Dongen, "MCL - A Cluster Algorithm for Graphs" http://micans.org/mcl/index.html [Last Accessed: 7-31-2015]

[5] Hadoop: http://hadoop.apache.org/ [Last Accessed: 7-31-2015]

[6] J. Norstad, A MapReduce Algorithm for Matrix Multiplication, http://www.norstad.org/matrix-multiply/, 2009 [Last Accessed: 7-31-2015]

[7] J. Schultz, J. Vierya, E. Lu, "Analyzing Patterns in Large-Scale Graphs Using MapReduce in Hadoop", Extended Abstract, Companion of IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing), Salt Lake City, UT, pp.1457-1458 , Nov. 2012

[8] SNAP: Stanford Network Analysis Project, http://snap.stanford.edu [Last Accessed: 7-31-2015]