Analyzing Patterns in Large-Scale Graphs Using MapReduce in Hadoop

Joshua Schultz, Undergraduate
Dept. of Math. & Computer Science
Salisbury University
Salisbury, USA
jschultz1@gulls.salisbury.edu

Jonathan Vierya, Undergraduate
Dept. of Computer Science
Cal Poly Pomona
Pomona, USA
2010jonathan@sbcglobal.net

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

***Abstract*— Analyzing patterns in large-scale graphs, such as social networks (e.g. Facebook, Linkedin, Twitter) has many applications including community identification, blog analysis, intrusion and spamming detections. Currently, it is impossible to process information in large-scale graphs with millions even billions of edges with a single computer. In this paper, we take advantage of MapReduce, a programming model for processing large datasets, to detect important graph patterns using open source Hadoop on Amazon EC2. The aim of this paper is to show how MapReduce cloud computing with the application of graph pattern detection scales on real world data. We implement Cohen's MapReduce graph algorithms to enumerate patterns including triangles, rectangles, trusses and barycentric clusters using real world data taken from Snap Stanford. In addition, we create a visualization algorithm to visualize the detected graph patterns. The performance of MapReduce graph algorithms has been discussed too.**

***Index Terms*—MapReduce, Cloud Computing, Graph Algorithms, Pattern Detection, Cohesive Components**

I. MOTIVATION

Graph models are frequently used in a wide range of applications to capture the relationships among data entities. In a graph model, data entities are represented as vertices while edges represent relationships among the entities. In order to extract useful information from these relationships, one effective approach is to analyze special patterns such as cohesive components concisely from the graph.

Technological advances have increased the volume of collected data. The corresponding graphs become larger and more complex, making it impractical to process them on a single computer. This has stressed a need to develop methods that recognize patterns and cohesive groups in large-scale graphs efficiently and effectively [1].

II. MAPREDUCE GRAPH ALGORITHMS

Recently, MapReduce has emerged as a reasonable solution to processing large data [2]. It also earned a strong interest to those who want to analyze graphs [1,3]. The programmer defines a map function which processes the input data and generates intermediate key/value pairs. The programmer also defines a reduce function which is applied to the intermediate key/value pairs to output final key/value pairs.

Using the open-source Hadoop implementation of the MapReduce model, programs can be automatically parallelized and executed on a cluster of machines [4]. The tasks of partitioning input data, scheduling the map and reduces tasks across a cluster of machines, handling machine errors, and managing communication between machines are transparent to programmers. The machine running the user's program, referred to as the master node, partitions input data into data segments and assigns map tasks to worker nodes. The map function is run on data segments in parallel distributed across multiple machines. The master node assigns reduce tasks to worker nodes which perform the reduce function on the intermediate map of key/value pairs. Figure 1 shows the execution of the MapReduce programming model in a distributed system.

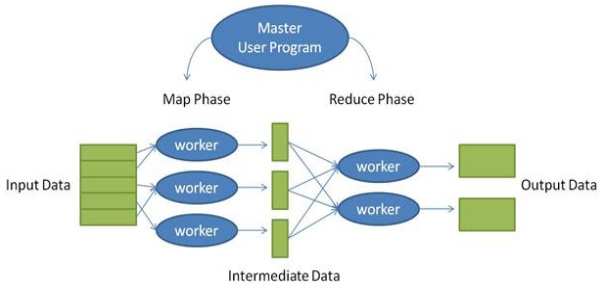


Fig. 1: MapReduce programming model

Cohen has outlined six MapReduce algorithms to analyze graph patterns [1]. These patterns include triangles, rectangles, trusses, barycentric clusters, and components. Some of these patterns such as triangles can be the basis for analyzing other patterns. A triangle is a three-vertex graph where every vertex connects to each other. A rectangle is a four-vertex cycle, which can be found by finding two triangles without the overlapped edge. A k-truss is a relaxation of a k-clique. In a k-truss, every edge is in k-2 triangles, while in a k-clique every vertex is connected to each other. Thus finding trusses can be done by enumerating triangles. Like trusses, barycentric clusters were proposed to find highly connected subgraphs. In barycentric clustering, vertices are given random initial positions that are then updated by multiplying by a matrix a given number of times.

III. Experimental Results and Analysis

We have implemented six MapReduce algorithms to analyze graph patterns including triangles, rectangles, trusses, barycentric clusters, and components. The input data for these algorithms are from Snap Stanford, which contains different types and sizes of real world datasets [5]. In Snap Stanford, the data is formatted as raw edges which fit nicely into Cohen's proposed algorithms. We used three different datasets ranging from 1 MB to 1GB, wiki-Vote (7,115 Vertices, 103,689 Edges, 1MB), soc-Slashdot0811 (77,360 Vertices, 905,468 Edges, 10MB), and soc-LiveJournal1 (4,847,571 Vertices, 68,993,773 Edges, 1GB).

All of the data processed was run on Amazons Elastic MapReduce "small" computers. Each of them was outfitted with 1.7 GB of memory 160 GB of storage and the equivalent of a 1.7GHz Xeon processor [6]. We show and discuss some experimental results below.

A. *Enumerating Triangles and Rectangles*

The MapReduce algorithms for enumerating triangles and rectangles scale well when datasets get large as shown in Figure 2. They were successfully run across an array of cluster sizes from one computer to twenty. We analyzed running times on the 1GB dataset and found a steady decline in the curve as the number of computer increased as shown in Figure 3. There are many other triangle counting algorithms that are very fast. Some of which are less than 100% accurate and do not actually enumerate every triangle [7]. We felt Cohen's MapReduce triangle enumerating algortihm offered an advantage over the others: it counts every triangle accurately and allows the data to be analyzed on each individual triangle. In addition, it uses raw edges instead of a vertex adjacent matrix which are not efficent for sparse graphs with millions of vertices.

B. *Finding Trusses*

Finding trusses is an excellent way to find highly connected subgraphs, and further analysis is required to see how it scales on the cloud. Figure 4 shows different run times as k increaeses. For any dataset, the fastest run time was when k=3. This is obvious because the algorithm enumerates triangles once. However the running time of the program explodes when k is greater than three. The reason is that even if one edge is dropped the entire program needs to be re-run as the finding trusses algorithm states.

C. *Barycentric Clustering*

We found that the barycentric algorithm does not have consistent outputs. By increasing the cut-off length for edges we would get the desired results on very small graphs (less than fourteen vertices). However, the algorithm lost the consistency as we increased the graph size. For the purpose of finding clusters the algorithm did well, but was not always consistent on different datasets. We feel that further work is needed to fully consider the usefulness of the algorithm.

Fig. 2: Enumerating Triangles and Rectangles on Two Computers

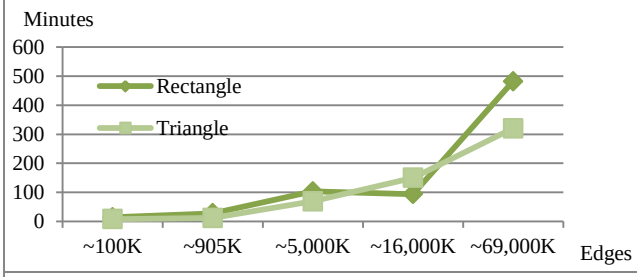


Fig 3: Running Time Enumerating Triangles as the Number of Computers Increases

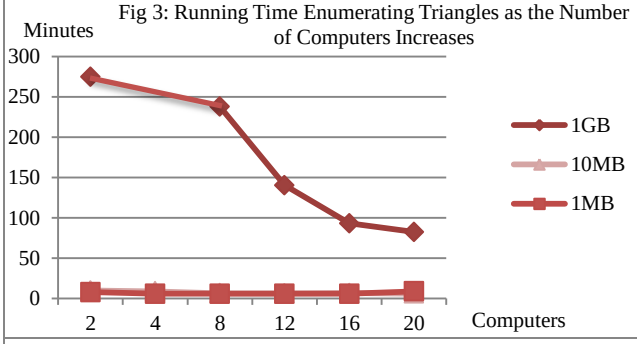
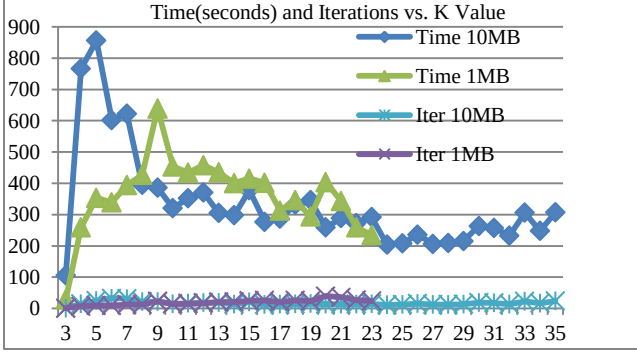


Fig 4: Finding k-trusses as K Increases on Real World Graphs Time(seconds) and Iterations vs. K Value



IV. Acknowledgment

The work is funded by NSF CCF-1156509 under Research Experiences for Undergraduates Program. We would like to thanks Professor Randal Burns at Johns Hopkins University for his valuable inputs on our work.

References

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

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Communications of the ACM, vol.51, no.1, Jan. 2008, pp. 107-113.

[3] J. Lin, and M. Schatz, "Design Patterns for Efficient Graph Algorithms in MapReduce," in Proc. Eighth Workshop on Mining and Learning with Graphs, 2010, pp. 78-85.

[4] Hadoop, http://hadoop.apache.org.

[5] SNAP: Stanford Network Analysis Project, http://snap.stanford.edu.

[6] Amazon Web Services, http://aws.amazon.com.

[7] C. E. Tsourakakis,"Fast counting of triangles in real-world networks: proofs, algorithms and observations," Technical Report CMU-ML-08-103, 2008.