PEGASUS: MINING BILLION-SCALE GRAPHS IN THE CLOUD

U Kang, Duen Horng "Polo" Chau, and Christos Faloutsos

School of Computer Science, Carnegie Mellon University

ABSTRACT

We have entered in an era of big data. Graphs are now measured in terabytes or even petabytes; analyzing them has become increasingly challenging. How do we find patterns and anomalies in these graphs that no longer fit in memory? How should we exploit parallel computation to boost our analysis capabilities? We present PEGASUS, the first open-source, peta-scale graph mining library, for the HADOOP platform (open-source implementation of MAPREDUCE).

By observing that many graph mining operations can be described by repeated matrix-vector multiplications, we devised an important primitive called GIM-V for PEGASUS that applies to all such operations. GIM-V (Generalized Iterative Matrix-Vector multiplication) is highly optimized, achieving (1) good scale-up with the number of machines, (2) linear run time on the number of edges, and (3) more than 9 times faster performance over the non-optimized version. We ran experiments for PEGASUS on M45, one of the largest HADOOP clusters in the world. We report our findings on several real graphs with billions of nodes and edges. Selected findings include (a) the discovery of adult advertisers in the who-follows-whom on Twitter, and (b) the 7-degrees of separation in the Web graph.

Index Terms— PEGASUS, graph mining, HADOOP

1. INTRODUCTION

Graphs are ubiquitous: computer networks, social networks, mobile call networks, and the World Wide Web, to name a few. Spurred by the lower cost of storage, the success of social networking websites and Web 2.0 applications, and the high availability of data sources, graph data are being generated at unprecedented size. They are now measured in terabytes or even petabytes, with billions of nodes and edges. Historically, however, most graph mining algorithms were designed under the assumption that the graphs would fit in the main memory of a workstation, or a single disk at its largest. The above graphs violate these assumptions. They require us to confront our long-held assumption, and to redesign the algorithms so they can work with these new breed of massive graphs. We surveyed promising frameworks that supported parallel computation, on which we could develop such massively-scalable algorithms. We selected HADOOP, an open-source implementation of MAPREDUCE.

We first address the research question: how to design efficient MAPREDUCE algorithms which can handle such massive graphs? There are several challenges. First, can we formulate graph mining algorithms using simple operations that can be efficiently implemented on MAPREDUCE? Second, how to store the graphs efficiently to minimize storage space and to enable fast graph queries?

The second question we investigate: what patterns and anomalies can we discover in huge, real-world graphs with billions of nodes and edges? Huge graphs have interesting patterns or regularities, such as those in their connected components, radii, triangles, etc. Discovering these patterns helps us spot anomalies, a capability useful in a wide spectrum of applications, such as cyber-security (computer networks), phone companies (fraud detection), and social networks (spammer detection).

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 describes the algorithms for large graph mining. In Section 4 we present the performance results and our findings in real world, large scale graphs. We conclude in Section 5.

2. RELATED WORKS

In this section, we review related work on MAPREDUCE, HADOOP, and large scale graph mining with HADOOP.

MAPREDUCE is a programming framework [1] for processing huge amounts of unstructured data in a massively parallel way. MAPREDUCE has two major advantages: (a) the programmer is oblivious of the details of the data distribution, replication, load balancing etc. and furthermore (b) the programming concept is familiar, i.e., the concept of functional programming. Briefly, the programmer needs to provide two functions, a *map* and a *reduce*. The typical framework is as follows [2]: (a) the *map* stage sequentially passes over the in-

Research was sponsored by the Defense Threat Reduction Agency under contract No. HDTRA1-10-1-0120, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. This work is also partially supported by an IBM Faculty Award, a Google Focused Research Award, and a Yahoo Research Alliance Gift. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, the U.S. Government, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

put files and outputs (key, value) pairs; (b) the *shuffling* stage groups of all values by key, and (c) the *reduce* stage processes the values with the same key and outputs the final result.

HADOOP is the open source implementation of MAPREDUCE. HADOOP provides the Distributed File System (HDFS) and PIG, a high level language for data analysis [3]. Large scale graph mining using HADOOP has attracted significant interests [4, 5, 6] due to its simplicity, fault tolerance, and low maintenance costs, compared to graph mining based on MPI [7] and Bulk Synchronous Parallel model [8].

3. ALGORITHMS FOR LARGE GRAPH MINING

Our proposed PEGASUS package[1] comprises large scale graph mining algorithms which we describe in this section. For each of the algorithm, we provide the motivating questions and our answers.

3.1. Structure Analysis

Problem 1 *How can we find connected components, diameter, PageRank, and node proximities of very large graphs quickly? Furthermore, how can we design a general primitive which can be applied to many different algorithms?*

We observe that many graph mining algorithms, like connected components, diameter, PageRank, and node proximities, can be unified via the `GIM-V` primitive, standing for `G`eneralized `I`terative `M`atrix-`V`ector multiplication [9]. In the `GIM-V`, we generalize the three internal operations(multiply, sum, and assign) in the standard matrix-vector multiplications to define many different algorithms.

Having defined `GIM-V`, the next question is to design efficient methods for the generalized matrix-vector multiplication in MAPREDUCE. Our first main idea is to put together several nonzero elements into square blocks, and perform the block-wise matrix-vector multiplication instead of element-wise multiplication. Our second main idea is to cluster the graph so that nonzero elements in the adjacency matrix are closely located, and then compress the nonzero bit strings of each block by standard compression algorithms like gzip. This compression greatly saves space, which leads to faster running time of block-wise matrix-vector multiplication.

3.2. Eigensolver

Problem 2 *How can we design a scalable eigensolver? How can we handle skewed matrix-matrix multiplication where one matrix is much larger than the other?*

Given a billion-scale graph, how can we find near-cliques, the count of triangles, and related graph properties? All of them can be found quickly if we have the first several eigenvalues and eigenvectors of the adjacency matrix of the graph [10]. Despite their importance, existing eigensolvers do not scale well. We developed HEIGEN [11], an eigensolver for billion-scale, sparse symmetric matrices.

A challenge in HEIGEN is to design an efficient method for skewed matrix-matrix multiplication, where the first matrix is much larger than the second matrix. Our main idea is to broadcast the smaller matrix to all the mappers, so that the second matrix can be joined with the elements of the first matrix in the mapper. This greatly reduces the network traffic and decreases the running time. Experiments show that our proposed method outperforms naive methods by 76× [11].

3.3. Inference

Problem 3 *How to scale-up the inference, or "guilt by association" algorithm for very large graphs with billions of nodes and edges?*

Inference in graphs is an important problem, which often corresponds, intuitively, to "guilt by association" scenarios. For example, if a person is a drug-abuser, probably his/her friends are so, too; if a node in a social network is of male gender, his dates are probably females. The typical way to handle this is belief propagation [12], and we tackle the scalability issue of the belief propagation.

We observe belief propagation cannot be formulated by a generalized matrix-vector multiplication on the original adjacency matrix and a vector. Instead, we formulate the belief propagation by a generalized matrix-vector multiplication on the line graph matrix and the message vector [13]. Our key contribution is to compute the multiplication without explicitly constructing the line graph: instead, we use the original adjacency matrix to compute the multiplication on the line graph.

3.4. Storage and Indexing

Problem 4 *How to store and index graph edge files so that graph mining queries can be answered quickly?*

We consider targeted graph mining queries whose answers require the access to only parts of the graph. Examples of targeted queries include k-step in/out-neighbors, and egonet queries. Our main idea is as follows. In the indexing stage, we make rectangular blocks of adjacency matrix, and store several blocks into grids where each grid corresponds to a square-shaped area in the adjacency matrix. In the query stage, only relevant grids are selected based on the queries. Experiments show that this 'grid selection' reduces the running time up to 4× than the naive methods [14].

4. EXPERIMENTS

In this section, we present experimental results including the performance of our proposed method, and the discoveries on

[1] available at `http://www.cs.cmu.edu/~pegasus`

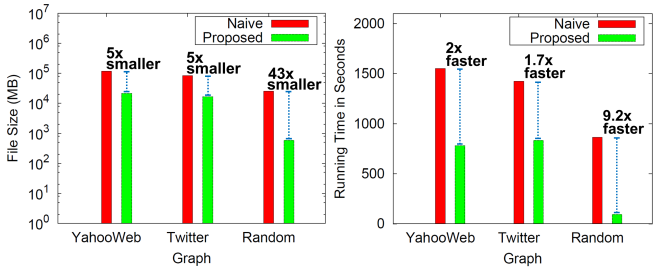
large, real world graphs. Table 1 lists the graphs used. The experiments were performed in Yahoo!'s M45 HADOOP cluster, one of the largest HADOOP clusters available to academia with 480 machines, 1.5 petabyte storage and 3.5 Terabyte memory in total.

Name	Nodes	Edges	Description
YahooWeb	1,413 M	6,636 M	Web links in 2002
Twitter	63 M	1,838 M	Who follows whom in Nov. 2009
Random	177 K	1,977 M	Synthetic graphs

Table 1. Order and size of networks.

4.1. Performance

Figure 1 shows the disk space and the running time comparisons of GIM-V variants. Note that the 'Proposed' method, which combines the clustering and the compression, provides up to 43× smaller storage and 9.2× faster running time compared to the 'Naive' method which does not have the clustering and the compression.



(a) File size (b) Running time

Fig. 1. (a) File size comparison after clustering and compression. The Y-axis is in log scale. Note our proposed method reduces the data size up to 43× smaller than the original('Naive'). The 'Random' graph has better performance gain than real-world graphs since the density is much higher. (b) Running time comparison of PageRank queries. Our proposed method outperforms the baseline by 9.2×.

4.2. Discoveries

We report interesting discoveries in large, real-world graphs. They include the patterns and anomalies in radius plots, connected components, and triangle counting.

4.2.1. Radius Plots

What are the central nodes and outliers in graphs? How closely are nodes in graphs connected? These questions are answered by radius plot, which is the distribution of the radius of nodes. The radius $r(v)$ of node v is the distance between v and a reachable node farthest away from v. The diameter of a graph is the maximum radius of nodes. The effective radius and the effective diameter are defined as the 90% percentile of the radius and the diameter, respectively [15, 16]. We analyze the effective diameter and radii of YahooWeb in Figure 2 (a). We have the following observations.

Small Web. The effective diameter of the YahooWeb graph (year: 2002) is surprisingly small ($\approx 7 \sim 8$).

Multi-modality of Web graph. The radius distribution of the YahooWeb graph has a multi-modal structure, which is possibly due to a mixture of relatively smaller subgraphs which got loosely connected recently.

4.2.2. Connected Components

What are the patterns and anomalies in the connected components of a Web graph? Figure 2 (b) shows the size distribution of connected components in YahooWeb graph. We have the following observation which shows the patterns of anomalous web pages [9].

Anomalous connected components. Figure 2(b) shows two outstanding spikes which deviate significantly from the 'power-law' like size distributions of small disconnected components. In the first spike at size 300, more than half of the components have exactly the same structure and they were made from a domain selling company where each component represents a domain to be sold. The spike happened because the company *replicated* sites using the same template. In the second spike at size 1101, more than 80 % of the components are adult sites disconnected from the giant connected component. Again, the adult sites are generated from a template. To summarize, the distribution plot of connected components reveals interesting communities with special purposes which are disconnected from the rest of the Internet.

4.2.3. Triangle Counting

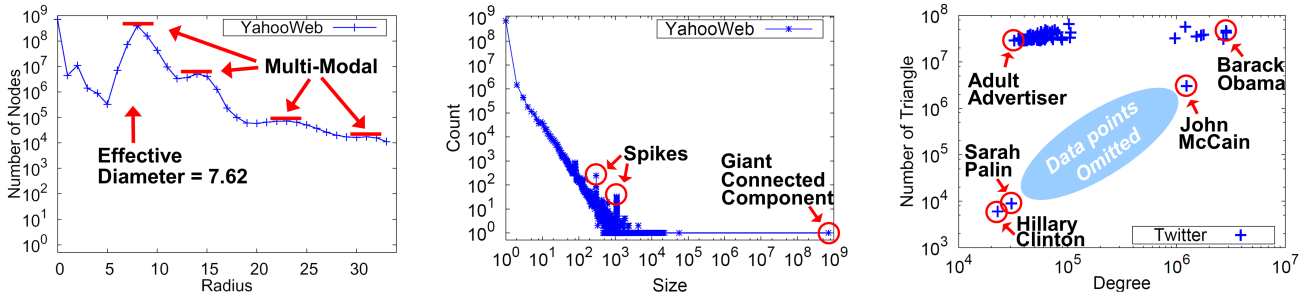
What are the patterns and anomalies in the triangle counts and the degrees in social network graphs? Figure 2 (c) shows the degree and the number of participating triangles in the Twitter 'who follows whom' graph at year 2009 [11]. We have the following observation which can be used to spot and eliminate harmful accounts such as those of adult advertisers and spammers.

Anomalous triangles vs. degree ratio. In Figure 2 (c), celebrities have high degrees and mildly connected followers, while adult accounts have many fewer, but extremely well connected, followers. The reason is that adult accounts are often from the same provider, and they follow each other to possibly boost their rankings or popularities.

5. CONCLUSION

We presented PEGASUS, a graph mining library for finding patterns and anomalies in massive, real-world graphs. Our major contributions include:

- Scalable algorithms for mining billion-scale graphs.



(a) Radius plot of YahooWeb (b) Connected components of YahooWeb (c) Triangle counts in Twitter

Fig. 2. Discoveries in large, real world graphs. (a) Radius plot of the YahooWeb graph. Notice the effective diameter is surprisingly small. Also notice the multi-modality which is possibly due to a mixture of relatively smaller subgraphs. (b) Connected components size distribution of the YahooWeb. Notice the two anomalous spikes which deviate significantly from the constant-slope tail. (c) The degree vs. participating triangles of some 'celebrities' in Twitter accounts. Also shown are accounts of adult sites which have smaller degree, but belong to an abnormally large number of triangles. The reason of the large number of triangles is that adult accounts are often from the same provider, and they follow each other to form a clique, to possibly boost their rankings or popularities.

- Performance analysis of our proposed method, which achieves up to 43× smaller storage and 9.2× faster running time.
- Discovery of patterns and anomalies of structural patterns in huge, real-world graphs. Some of our most impressive findings are (a) the discovery of adult advertisers in the who-follows-whom on Twitter, and (b) the 7-degrees of separation in the Web graph.

As we are only at the dawn of the era of big data, many exciting research directions await us. We have begun working on extending PEGASUS to support massive-scale tensor analysis and unsupervised anomaly detection.

References

[1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *OSDI'04*, Dec. 2004.

[2] Ralf Lämmel, "Google's mapreduce programming model – revisited," *Science of Computer Programming*, vol. 70, pp. 1–30, 2008.

[3] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins, "Pig latin: a not-so-foreign language for data processing," in *SIGMOD '08*, 2008, pp. 1099–1110.

[4] Spiros Papadimitriou and Jimeng Sun, "Disco: Distributed co-clustering with map-reduce," *ICDM*, 2008.

[5] "Mahout," http://lucene.apache.org/mahout/.

[6] Amol Ghoting, Rajasekar Krishnamurthy, Edwin P. D. Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan, "Systemml: Declarative machine learning on mapreduce," in *ICDE*, 2011, pp. 231–242.

[7] Aydin Buluç and John R. Gilbert, "The combinatorial blas: design, implementation, and applications," *IJHPCA*, vol. 25, no. 4, pp. 496–509, 2011.

[8] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD Conference*, 2010, pp. 135–146.

[9] U Kang, C.E Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system - implementation and observations," *ICDM*, 2009.

[10] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *KDD*, 2009, pp. 837–846.

[11] U. Kang, Brendan Meeder, and Christos Faloutsos, "Spectral analysis for billion-scale graphs: Discoveries and implementation," in *PAKDD (2)*, 2011, pp. 13–25.

[12] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the AAAI National Conference on AI*, 1982, pp. 133–136.

[13] U. Kang, Duen Horng Chau, and Christos Faloutsos, "Mining large graphs: Algorithms, inference, and discoveries," in *ICDE*, 2011, pp. 243–254.

[14] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos, "Gbase: a scalable and general graph management system," in *KDD*, 2011, pp. 1091–1099.

[15] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec, "Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations," in *SDM*, 2010, pp. 548–558.

[16] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec, "Hadi: Mining radii of large graphs," *ACM Trans. Knowl. Discov. Data*, vol. 5, pp. 8:1–8:24, February 2011.