PSCAN: A Parallel Structural Clustering Algorithm for Big Networks in MapReduce

Weizhong Zhao[†][*], VenkataSwamy Martha[*], Xiaowei Xu[*]
[†]*College of Information Engineering, Xiangtan University, Xiangtan, China*
[*]*University of Arkansas at Little Rock Little Rock, AR, USA*
vxmartha@ualr.edu, wxzhao1@ualr.edu, xwxu@ualr.edu

***Abstract*—Big data such as complex networks with over millions of vertices and edges is infeasible to process using conventional computation. MapReduce is a programming model that empowers us to analyze big data in a cluster of computers. In this paper we propose a Parallel Structural Clustering Algorithm for big Networks (PSCAN) in MapReduce for the detection of clusters or community structures in big networks such as Twitter. PSCAN is based on the structural clustering algorithm of SCAN, which not only finds cluster accurately, but also identifies vertices playing special roles such as hubs and outliers. An empirical evaluation of PSCAN using both real and synthetic networks demonstrated an outstanding performance in terms of accuracy and running time. We analyzed a Twitter network with over 40 million users and 1.4 billion follower/following relationships by using PSCAN on a Hadoop cluster with 15 computers. The result shows that PSCAN successfully detected interesting communities of people who share common interests.**

***Keywords*-Network clustering algorithms, community structures, big data, MapReduce, Hadoop.**

I. Introduction

We have been facing explosive growth of data. Recently the term of "big data" is often used to describe the huge amount of data that has been collected in many areas such as sensors used to collect climate information, surveillance cameras for digital videos, signals of cellular phones, webpages in the internet, and user posts in social media to just name a few. The pace of data growth is approaching and in many cases has already outgrown our capability to storage, process, and analyze data, which is a major challenge in almost every business today. Many of the big data are highly complex, which can only be represented as big graphs with millions of vertices and edges. The data or networks require distributed and parallel processing with over thousands of computers. Apache Hadoop cluster is a major distributed system used by many businesses to process the big data due to its outstanding scalability of growing the system with additional computers, flexibility of processing both structured and unstructured data, fault tolerance of computer system failures, and convenience of parallel programming model of MapReduce.

In this paper we propose a parallel clustering algorithm for big networks in MapReduce of Apache Hadoop. Network clustering is an important tool for analysis of large graphs including social and biological networks. Many network clustering algorithms have been successfully used for analysis of large networks. We focus on the clustering algorithm of SCAN, a structural clustering algorithm, because of its accuracy of finding clusters and vertices playing special roles, as well as its linear time complexity, which is scalable to large networks [5].

Specifically, SCAN finds clusters by using breadth-first search of the graph and claims a visited vertex as a new member of the cluster if its structural similarity, which is a normalized measure of shared neighbors with a current cluster member, is larger than or equal to a real parameter ϵ. Therefore, a straightforward parallelization of SCAN algorithm would be using a parallel breadth-first search algorithm in MapReduce as described in [9]. The parallel breadth-first search is implemented using an iterative MapReduce job, where each iteration expands the search frontier by only one hop from the source. Therefore, the number of iterations is the diameter of the graph, or the largest distance between any pair of vertices in the graph. Although this number is small in average according to the well-known six degrees of separation of social networks, it is quite large for some big networks based on our experiments.

Therefore, we propose a fast Parallel Structural Clustering Algorithm for big Networks (PSCAN) in MapReduce as follows. First the structural similarity is calculated for each edge of the graph. Then the graph is pruned by cutting off the edges with the structural similarity less than the threshold ϵ. Finally, the clusters are identified by finding connected components in the pruned graph. The advantage of PSCAN over the straightforward parallelization is significantly reduced MapReduce job iterations, which makes it scalable to big networks as demonstrated in our empirical evaluation.

The paper is organized as follows. We review related work in section II. PSCAN algorithm is presented in section III. An empirical evaluation of PSCAN is performed by using both synthetic and real big networks. The experiment results are presented in section IV. Finally we conclude the paper with future work in section V.

II. Related Work

Finding clusters or community structures is one of the fundamental task for network analysis. There are many algorithms having been proposed and successfully used across many disciplines. A more comprehensive overview can be found in [10]. A brief review of representative algorithms and other related work are presented as follows.

Network clustering has been first applied in biology and social sciences for taxonomy and sociology. Since many networks have hierarchical structures, hierarchical clustering algorithms have been successfully used [11]. Like network clustering graph partitioning algorithms aim to divide vertices into groups such that the number of edges lying between groups is minimized. One of the earliest method is the KernighanLin algorithm [12], which is still used in parallel computing, circuit partitioning and layout. Spectral clustering algorithms have been successfully applied in image processing, which finds clusters by using eigenvectors of the adjacency matrix of the graph. Normalized cut technique is a good example of spectral clustering algorithms [13]. Modularity, an objective or quality function originally defined by Newman and Girvan [8], has been used by many network clustering algorithms for the detection of community structures. Optimization of modularity is an NP complete problem [14]. Therefore, most algorithms find approximate solutions by using greedy search including [15]. In many applications clusters may overlap with each other. Network clustering algorithms have been proposed to find clusters that share some common vertices. One of the popular approach is the Clique Percolation Method (CPM) [16].

All methods discussed above have a time complexity that is not scalable to large networks. Scalable methods are devised to cluster large networks in linear or nearly linear time complexity with the size of the network. A good representative of scalable network clustering algorithms is the Structural Clustering Algorithm for Networks (SCAN) [5], which is an extension of the density based clustering algorithm (DBSCAN) for large networks [17].

Even a scalable algorithm like SCAN alone can't efficiently handle big networks with millions of vertices and edges. Therefore, parallel algorithms using modern computational infrastructures such as Hadoop cluster is required [9]. A remarkable work is [18], which implemented a label propagation algorithm [7] in MapReduce for detecting community structures in a Twitter network containing 40 million users and 1.4 billion edges. However, the implemented label propagation algorithm is not deterministic, which means the clustering result may vary for the same input graph. Moreover, the bipartite or nearly bipartite subgraphs lead to the oscillations of cluster labels, which is especially true in cases where communities take the form of a star graph.

In this paper we focus on SCAN and propose a Parallel SCAN (PSCAN) for detecting community structures in big networks in MapReduce.

III. PSCAN Algorithm in MapReduce

In this section, we present a Parallel Structural Clustering Algorithm for Networks (PSCAN) in MapReduce. Given one real parameter ϵ and input undirected network represented in adjacency list, PSCAN finds clusters and a (possible empty) set of outliers and hubs, in which structural similarity of each edge in each cluster is larger than or equal to ϵ and no edge adjacent to outliers or hubs with structural similarity larger than or equal to ϵ.

The basic idea of PSCAN is as follows. Firstly, it calculates the structural similarity of edges; then, it cuts off the edges with structural similarity less than ϵ; finally, it finds all the connected components in the pruned network, each of which is a cluster, and the vertices that are not part of any cluster are either outliers or hubs. Note that all of the three steps can be executed in parallel in MapReduce.

A. Parallel Calculation of Structural Similarity (PCSS)

In order to calculate the structural similarity of one edge, we just need the adjacency lists of two vertices adjacent to the edge. The calculation of the structural similarity of different edges is independent. Therefore it can be executed simultaneously in MapReduce by using two functions, a mapper and a reducer respectively.

More specifically, the mapper takes a pair of (key, value) as input, where the key is the input vertex and the value is its adjacency list. To illustrate our algorithm, we take the vertex v and its adjacency list $< v : v1, v2, \cdots, v_s >$ as an example. For each neighbor v_i $(i = 1, 2, \cdots, s)$ of v, we know that there is an undirected edge between v and v_i. In order to represent the edge uniquely, it is represented using the vertex pair in an increasing order of the vertices adjacent to the edge. For example, if v precedes v_i, the edge between v and v_i is represented as (v, v_i), or it is represented as (v_i, v). Without loss of generality, we assume v precedes v_i.

For each neighbor v_i of vertex v, the mapper emit a (key, value) pair, in which the key is the edge (v, v_i) and the value is the adjacency list of v. If the input vertex in the mapper is v_i, we know that v is a neighbor in the adjacency list of v_i. When it processes the vertex v_i, the mapper will emit a (key, value) pair, in which the key is also the edge (v, v_i) and the value is the adjacency list of v_i. In the reducer, for the key (v, v_i), the corresponding values will include the adjacency lists of v and v_i. Therefore, we can calculate the structural similarity of (v, v_i) in the reducer. All the edges in the network can be processed in the same way. Thereby, the structural similarity of edges can be calculated in parallel in MapReduce.

The pseudo codes of the mapper and the reducer is summarized in algorithm 1 and algorithm 2 respectively.

Algorithm 1 PCSS_Mapper

```
Input: key is the input vertex, value is the adjacency list of the input vertex
Output: $< key', value' >$, where the $key'$ is the edge connecting the input vertex and its neighbor and $value'$ is the adjacency list of the input vertex
1) Get the input vertex ID v and its adjacency list from $key/value$ pair;
2) For each neighbor $v_i$ in the adjacency list do
       If $v < v_i$
         Take $(v, v_i)$ as $key'$;
       else
         Take $(v_i, v)$ as $key'$;
       Take the adjacency list of vertex $v$ as $value'$;
       Emit($key', value'$)
3) End For
```

Algorithm 2 PCSS_Reducer

```
Input: key is one edge, values includes the adjacency lists of the two vertices adjacent to the edge
Output: $< key', value' >$, where the $key'$ is the edge and $value'$ is structural similarity of the edge
1) Get the adjacency lists of the two vertices from the $values$;
2) Calculate the structural similarity of the edge;
3) Take $key$ namely the edge as the $key'$;
4) Take the structural similarity of the edge as $value'$;
5) Emit$< key', value' >$;
```

B. Cutting off Edges

The output of the previous step is the edge list with structural similarity of each edge. Given the threshold ϵ, we can cut off the edges with structural similarity less than ϵ. Actually, we can do the cutting procedure in the reducer of previous step, in which we just emit the edges whose structural similarity is larger than or equal to ϵ.

After the cutting phase, we get several connected components, each of which is recognized as a cluster. Then we find the connected components to get the final clustering result.

C. Label Propagation for Connected Components (LPCC)

Inspired by the idea of label propagation [7], we can find connected components in parallel from different vertices. Recently the same idea is also proposed by [6]. The basic idea is that we label all the vertices in one connected component as the smallest (or largest) ID of vertices in the connected component.

More specifically, we first prepare the input to the format of (vertex ID, structure information), in which the structure information includes status, label and adjacency list of the vertex. Each vertex has two status: activated and inactivated. The activated vertex needs propagate its current label to its neighbors, while the inactivated vertex doesn't need to do that. At beginning, each vertex is initialized as activated and its label is its own ID.

Then, the label propagation proceeds in an iterative way. Each iteration is a MapReduce job. In the mapper, each of the activated vertex propagates its label to all of its neighbors and passes its structure information to the reducer. For the inactivated vertex, it just passes the structure information to the reducer. In the reducer, for each vertex, if the smallest label propagated from its neighbors is less (or larger) than its current label, we update the label as the new one and set the vertex as activated. If the label need not change, we set the vertex as inactivated. The procedure iterates until there is no label updated in one iteration.

Finally, vertices with the same label are in the same connected component, which is taken as a cluster. The algorithm is summarized in algorithms 3, 4 and 5.

Algorithm 3 LPCC

```
Input: A network in the format of adjacency list
Output: All of the connected components in the network
1) Initialize the label of each node as its own ID and the status is activated;
2) Iterate
       LPCC_Mapper();
       LPCC_Reduce();
3) Until all nodes are inactivated
```

D. Time Complexity

The time complexity of calculating the structural similarity of edges is linear with the number of edges. The step of finding connected components takes a running time, which is in the order of the diameters of the connected components. Since many of the social networks have an average diameter of six, which is known as six degrees of separation, the overall running time of PSCAN is linear with the number of edges in the graph.

IV. Empirical Evaluation

We empirically evaluated the performance of the proposed algorithm for clustering big undirected networks. All of the experiments were conducted on a cluster of 15 computers, each of which includes 8 core 2.6 GHz processors and 16GB of memory. Hadoop version 1.0.0 and Java 1.6.0-31 are used as the MapReduce system for all the experiments.

Algorithm 4 LPCC_Mapper

Input: *key* is the input vertex ID, *value* is the structure information (status, label and adjacency list) of the input vertex

Output: $< key', value' >$, where the key' is a vertex ID and $value'$ is the label or the structure information of the input vertex

```
1) Get the vertex ID, status, label and its adjacency list
   from $key/value$ pair;
2) If the vertex is activated,
       For each neighbor $v_i$ in the adjacency list,
         Take $v_i$ as $key'$;
         Take the label of the input vertex as $value'$;
         Emit$< key', value' >$;
3) End if
4) Take the vertex ID as $key'$;
5) Take the structure information of the vertex as $value'$;
6) Emit$< key', value' >$;
```

Algorithm 5 LPCC_Reducer

Input: *key* is vertex ID, *values* includes the structure information of the vertex and the labels from its neighbors.

Output: $< key', value' >$, where the key' is the vertex ID and $value'$ is the updated structure information of the vertex.

```
1) For each $value$ in $values$,
       If the $value$ is the structure information,
         Store the structure information temporarily;
       else
         Find the smallest label from the neighbors;
2) If the smallest label from its neighbors is less than its
   current label,
       Set the vertex as activated and update the label in
       the structure information as the smallest label;
3) else
       Update the status as inactivated in structure infor-
       mation of the vertex;
4) Take $key$, namely the vertex ID as $key'$;
5) Take the updated structure information of the vertex
   as $value'$;
6) Emit$< key', value' >$;
```

A. Datasets

To evaluate the performance of our algorithm we use both synthetic and real datasets.

1) Synthetic Datasets: The first set of synthetic datasets is generated to evaluate the accuracy of our algorithm by using the benchmark data generator proposed in [1]. The generated synthetic graphs have built-in community structures and can be used as a benchmark to evaluate the clustering accuracy. Moreover, in the synthetic graphs, the distributions of vertex degree and community size are both power laws, which are two important characteristics of many real networks. The statistics of the generated benchmark graphs are presented in Table I.

Table I
BENCHMARK GRAPH

Graph	# nodes	# edges	# clusters
Graph-5k	5,000	35,626	278
Graph-10k	10,000	70,365	598
Graph-20k	20,000	143,521	1008
Graph-40k	40,000	285,275	2030
Graph-80k	80,000	576,642	3876
Graph-160k	160,000	2,293,255	4115

To evaluate the scalability of PSCAN, the second set of synthetic graphs is generated by using Barabasi graph model [2]. The graphs generated by Barabasi model is a scale-free network. However, we do not know the real community structures in the graphs. Therefore we only use them to evaluate the running time of our algorithm. We could use the benchmark graph generator in [1] for the purpose. However, it is too slow and failed to generate a graph with one million vertices after running more than a week. The statistics of the generated graphs using Barabasi model are presented in Table II.

Table II
BARABASI GRAPH

Graph	# nodes	# edges
Barabasi-1M	1,000,000	13,999,895
Barabasi-2M	2,000,000	27,999,895
Barabasi-3M	3,000,000	33,999,847
Barabasi-4M	4,000,000	59,999,880

2) Real Datasets: Twitter dataset (collected by Kwak et al [19]) has follower/following relationship between twitter users. The Twitter dataset has nearly 42 million vertices and 1.4 billion edges. We treat Twitter as an undirected graph by removing the directions of edges in the original graph. Moreover, we remove the vertices whose degrees larger than 20, because they are most likely the hubs according to the theory of structural clustering [5]. After preprocessing, Twitter dataset contains 25,017,493 vertices and 46,417,485 edges.

B. Evaluation Methodology

1) Clustering Accuracy: To this end, we use benchmark graphs to evaluate the accuracy of clustering results. For each graph, we tried five different ϵ in step 0.2 from 0 to 1, and took the clustering result that maximizes the modularity [8] as the final result. We used the Adjust Rand Index (ARI) [3] and the Normalized Mutual Information (NMI) [4] to evaluate the clustering accuracy. For the perfect clustering

Table III
CLUSTERING ACCURACY

Graph	ARI	NMI
Graph-5k	0.997	0.999
Graph-10k	0.998	0.999
Graph-20k	0.969	0.977
Graph-40k	0.981	0.988
Graph-80k	0.971	0.987
Graph-160k	0.999	0.999

result, the ARI and NMI values are both equal to 1. In general, the larger the ARI and NMI values are, the better the clustering quality is.

The clustering accuracy results are presented in Table III. The results show that PSCAN can find the clusters, outliers or hubs in each graph effectively. Especially for Graph-5k, Graph-10k and Graph-160k, PSCAN successfully identified nearly all of the clusters, outliers or hubs in the graphs. The reason is that PSCAN always produces the same result as the original SCAN algorithm [5], which is very accurate for clustering large networks. Therefore, PSCAN performs effectively in clustering big graphs.

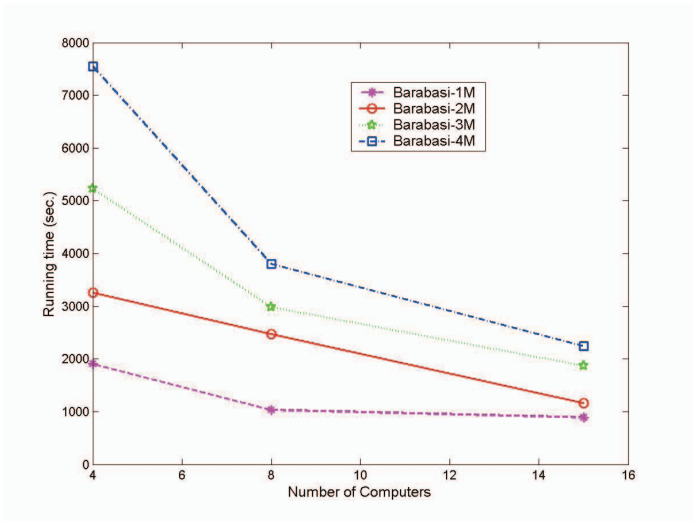
2) Running Time Performance: To this end, we employ Barabasi graphs to evaluate the speedup, scaleup and sizeup performance of our algorithm. Since we will only evaluate the efficiency of PSCAN and not its accuracy, we set the same ϵ when tested on different Barabasi graphs. Moreover, we set the number of iterations for finding connected components as six assuming of six degrees of separation.

To measure the speedup, we kept the graphs constant and increased the number of computers in the system. We have tested the speedup on different Barabasi graphs on different Hadoop clusters with 4 ,8 and 15 computers respectively. In our experiments, the relative speedup given by the larger system with m computers is defined as:

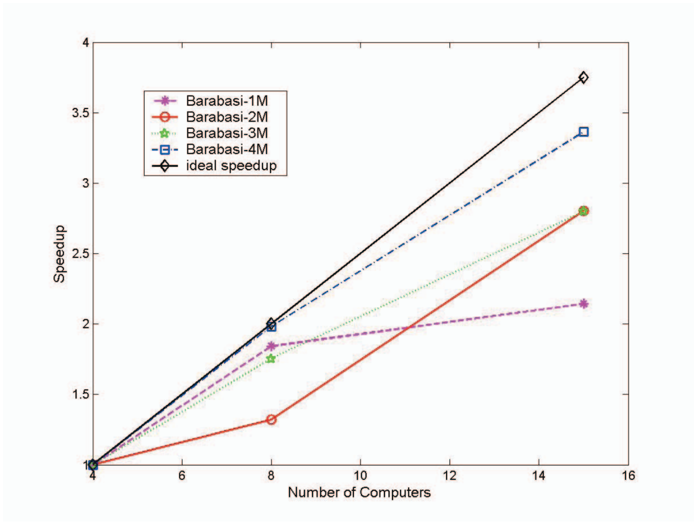
$$Relative\ speedup(m) = \frac{\texttt{running-time on 4 computers}}{\texttt{running-time on } m \texttt{ computers}} \quad (1)$$

The results are presented in Figure 1. As the figure shows, our algorithm has a very good speedup performance. From the relative speedup in Figure 1(b), we noticed that the larger graphs have an improved speedup performance. The reason is that increasing the size of the graphs reduced the percentage of the overall time spent in communication and I/O operations.

Scaleup measures the ability to grow both the system and the graph size. Specifically, we have evaluated running time on Barabasi-1M on a Hadoop cluster with 4 computers, running time on Barabasi-2M on a Hadoop cluster with 8 computers and running time on Barabasi-4M on a Hadoop cluster with 15 computers respectively. Each running time



(a) Speedup



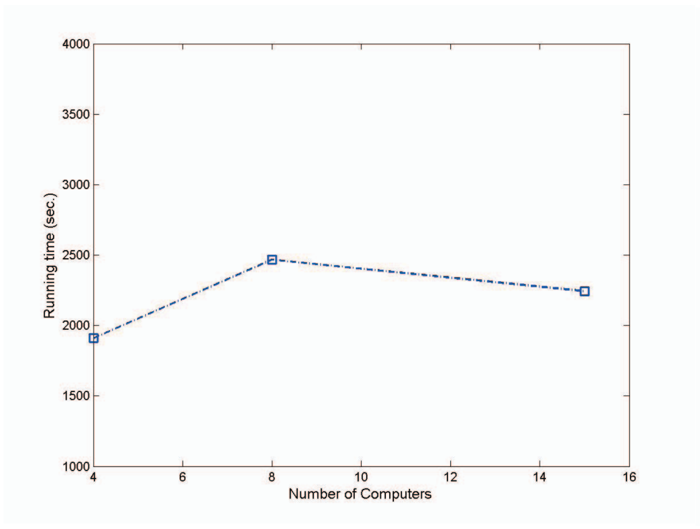
(b) Relative Speedup

Figure 1. Speedup

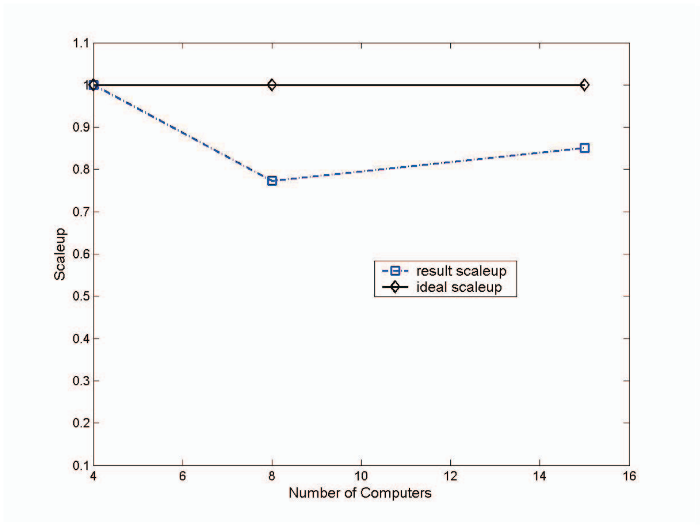
divided by running time on Barabasi-1M on the Hadoop cluster with 4 computers is the relative scaleup result.

The ideal scaleup is a horizontal line with value 1 in the vertical axis in relative scaleup result. However, ideal scaleup is difficult to achieve because of the communication and I/O cost increasing when the scale of Hadoop cluster grows. Figure 2 shows the scaleup results. Clearly, the PSCAN algorithm scales very well. Similarly the larger graphs show a better scaleup performance.

To measure the performance of sizeup, we hold the number of computers in the system constant, and increase the size of graphs. Sizeup measures how much longer it takes on a given system, when the graph size is m-times larger than the original graph. The relative sizeup is defined

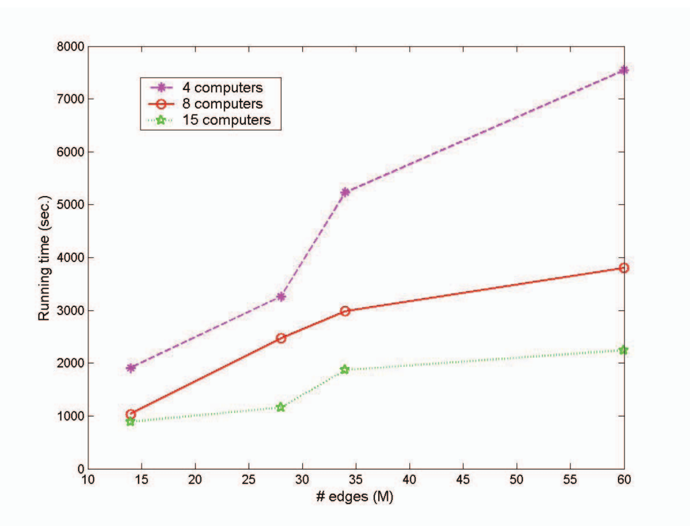


(a) Scaleup

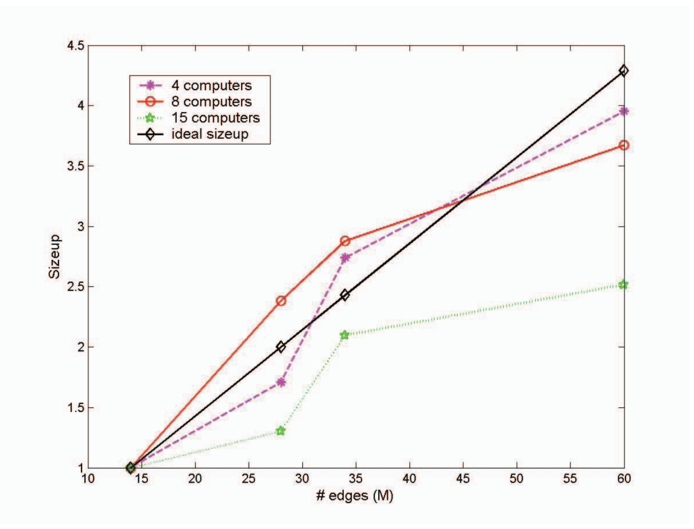


(b) Relative Scaleup

Figure 2. Scaleup



(a) Sizeup



(b) Relative Sizeup

Figure 3. Sizeup

as follows:

$$Relative\ sizeup(DB, m) = \frac{\texttt{running-time for clustering } m{\star}DB}{\texttt{running-time for clustering } DB} \quad (2)$$

In evaluating sizeup of PSCAN, we have fixed the number of computers as 4, 8, and 15 respectively. Figure 3 shows the results on different scale of Hadoop clusters. The graph shows that PSCAN has a very good sizeup performance. Specifically, the results in Figure 3(b) show that the sizeup of PSCAN is better when the scale of cluster and size of graphs become larger.

3) Results on Twitter: Since we run PSCAN on preprocessed Twitter dataset, we apply certain post-processing to guarantee the accuracy of the clustering result. For a large degree (larger than 20) vertex in original Twitter network, if its neighbors in the clustering result are assigned to different communities, the vertex is classified as a hub according to the structural clustering theory [5]. If its neighbors in the clustering result are classified as outliers, the large degree vertex should be the center of a community, and the neighbors are identified as members in the community with the large degree vertex as the core based on the structural clustering theory.

Our experiments on the Twitter network [19] found meaningful clusters. A cluster obtained by PSCAN for Twitter network represents a group of people who share common interests and other features. Manual investigation on some of the clusters revealed that PSCAN identifies two users of a cluster, although there is no direct follower/following relation exists but sharing some common interests. Such capability of PSCAN helped us to find users from a city, an organization, or a country. It is not feasible to discuss all the clusters here, one of the interesting clusters is a cluster representing twitter pages of BBC weather channel and weather alerts. The cluster of 20 members all representing BBC weather related pages is found. There are many such clusters that represents a group of users who share some common interests. There is no base line of communities in Twitter to measure accuracy of the clustering but our manual observations found the accuracy is significant. Moreover, the experiment is designed to prove the feasibility of SCAN in MapReduce framework because SCAN is proved to be accurate enough for clustering. The experiment on Twitter data proved the accuracy and scalability of PSCAN.

V. Conclusions and Future Work

We present a parallel structural clustering algorithm (P-SCAN) for big networks in MapReduce in this paper. PSCAN identifies clusters, as well as vertices playing critical roles such as outliers and hubs in big networks with billions of edges in three steps, namely calculating structural similarity of edges, cutting off edges with low structural similarity and finding connected components. All the steps can be executed in parallel in MapReduce. The time complexity of PSCAN is linear with the number of edges in the graph. Our empirical evaluation demonstrated an accurate clustering result and an excellent running time in terms of scaleup, sizeup and speedup. Moreover, we applied PSCAN for analysis of a Twitter social network with over 40 million users and 1.4 billions of follower/following relationships. The result shows that PSCAN can find interesting communities of people sharing common interests or other features. In the future, we plan to further investigate the performance of PSCAN by applying it for the analysis of some really big networks in real world.

Acknowledgment

This project was funded by Acxiom Corporation. The authors are grateful for invaluable collaboration with Kevin Liles and Derek Leonard throughout the project. This work was supported in part by the National Science Foundation under Grant CRI CNS-0855248, Grant EPS-0701890, Grant EPS-0918970, Grant MRI CNS-0619069, and OISE-0729792. Weizhong Zhao would like to thank the support of the National Natural Science Foundation of China (No. 61105052).

References

[1] A. Lancichinetti, S. Fortunato, and F. Radicchi, *Benchmark graphs for testing community detection algorithms*. Physical Review E 78, 046110, 2008.

[2] S. Yook, H. Jeong, and A. Barabasi, *Modeling the internet's large scale topology*. In PNAS Proceedings of the National Academy of Science, pages 13382-13386, October 2002.

[3] L. Hubert and P. Arabie, *Comparing partitions*. Journal of Classification, 193C 218, 1985.

[4] A. Strehl, J. Ghosh, R. Mooney, *Impact of similarity measures on web-page clustering*. Proceedings of the workshop on artificial intelligence for web search, pp 58-64, 2000.

[5] X.Xu, N.Yuruk, Z. Feng, T. Schweiger. *SCAN: a structural clustering algorithm for networks*,Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 824-833, 2007.

[6] Bin Wu; YaHong Du. *Cloud-based Connected Component Algorithm*, 2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI), vol.3, no., pp.122-126, 23-24 Oct. 2010.

[7] Usha Nandini Raghavan, Rka Albert, and Soundar Kumara, *Near linear time algorithm to detect community structures in large-scale networks*. Phys. Rev. E 76, 036106 (2007)

[8] M. E. J. Newman and M. Girvan, *Finding and evaluating community structure in networks*, Phys. Rev. E 69, 026113 (2004).

[9] Jimmy Lin and Chris Dyer, *Data-Intensive Text Processing with MapReduce*, Morgan and Claypool Publishers, 2010. pp. 94-101.

[10] Santo Fortunato, *Community detection in graphs*, Physics Reports, Volume 486, Issues 35, February 2010, Pages 75-174, ISSN 0370-1573, 10.1016/j.physrep.2009.11.002.

[11] Jiawei Han, Micheline Kamber, and Jian Pei, *Data Mining: Concepts and Techniques*, 3rd edition, Morgan Kaufmann, 2011.

[12] B.W. Kernighan, S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Syst. Tech. J. 49 (1970) 291307.

[13] J. Shi, J. Malik, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell. 22 (8) (2000) 888905.

[14] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikolski, D. Wagner, *On modularity np-completeness and beyond*. URL http://digbib. ubka.uni- karlsruhe.de/volltexte/documents/3255.

[15] A. Clauset, M.E.J. Newman, C. Moore, *Finding community structure in very large networks*, Phys. Rev. E 70 (6) (2004) 066111.

[16] G. Palla, I. Derényi, I. Farkas, T. Vicsek, *Uncovering the overlapping community structure of complex networks in nature and society*, Nature 435 (2005) 814818.

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise*. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226231. ISBN 1-57735-004-9.

[18] Akshay U. Bhat. *Scalable Community Detection using Label Propagation & Map-Reduce*, http://www.akshaybhat.com/LPMR/. 2008

[19] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon, *What is Twitter, a Social Network or a News Media?*, WWW 2010, April 2630, 2010