Scalable Community Detection from Networks by Computing Edge Betweenness on MapReduce

Seunghyeon Moon, Jae-Gil Lee*, Minseo Kang
Department of Knowledge Service Engineering
KAIST
Daejeon, Republic of Korea
Email: { myth624, jaegil, minseo }@kaist.ac.kr

***Abstract*—Community detection from social network data gains much attention from academia and industry since it has many real-world applications. The Girvan-Newman (GN) algorithm is a divisive hierarchical clustering algorithm for community detection, which is regarded as one of the most popular algorithms. It exploits the concept of *edge betweenness* to divide a network into multiple communities. Though it is being widely used, it has limitations in supporting large-scale networks since it needs to calculate the shortest path between *every pair* of nodes in a network. In this paper, we develop a parallel version of the GN algorithm to support large-scale networks. To this end, we propose a new algorithm, which we call *Shortest Path Betweenness MapReduce Algorithm* (SPB-MRA), that utilizes the MapReduce model. This algorithm consists of four major stages, and all operations are executed in parallel. In addition, we suggest an approximation technique to further speed up community detection processes. We implemented SPB-MRA on Hadoop, which is the most popular open-source platform for MapReduce, and then conducted performance tests for SPB-MRA on Amazon EC2 instances. The results showed that elapsed time decreases almost linearly as the number of reducers increases and the approximation technique introduces negligible errors.**

Keywords*—MapReduce, Hadoop, community detection, edge betweenness, Girvan-Newman algorithm, SPB-MRA.

I. Introduction

As social networking services (SNSs) such as Facebook and Twitter are getting more popular, analyzing social network data has become one of the most important issues in various areas. Among those analysis jobs, community detection from social network data gains much attention from academia and industry since it has many real-world applications such as friend recommendation and target marketing.

Community detection is to partition the set of network nodes into multiple groups such that the nodes within a group are connected densely, but connections between groups are sparse. There have been many studies regarding community detection. The Girvan-Newman (GN) algorithm proposed by Girvan and Newman [1] exploits the concept of *edge betweenness*, which is a measure of the centrality and influence of an edge in a network. Though the GN algorithm is being widely used, it has limitations in supporting large-scale networks since it needs to calculate the shortest path between *every pair* of nodes. The number of node pairs in a large-scale network is really prohibitive.

In the era of Big Data, the amount of available data is growing unprecedentedly. Thus, data analysis calls for very scalable methods that can deal with large data sets. MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. MapReduce has been widely used owing to its scalability and ease of use [2], [3], [4], [5], [6]. It is the the driving force behind big data analysis in recent years.

In this paper, we propose a parallel version of the GN algorithm to support large-scale networks and suggest an approximation technique to further speed up community detection processes. The proposed algorithm, which we call *Shortest Path Betweenness MapReduce Algorithm* (SPB-MRA), utilizes the MapReduce model. This algorithm consists of four major stages, and all operations are executed in parallel. In the first stage, all-pair shortest paths on a network are calculated. In the second stage, the edge betweenness for every pair of nodes in the network is calculated. In the third stage, k_{iter} edges are selected by edge beweenness, and they get removed. In the final stage, the network is updated, and this new network is provided to the next iteration. These four stages repeat until the quality of communities does not improve any more.

The major contributions of this paper are as follows.

- We propose an algorithm called SPB-MRA, which is a parallel version of the GN algorithm, and demonstrate the performance improvement in community detection for large-scale data. The results of performance tests showed that elapsed time decreased almost linearly as more reducers were added.
- We suggest an approximation technique to further speed up community detection processes. Instead of removing a *single* edge per iteration, we remove *multiple* edges that have the top-k_{iter} highest edge betweenness at once. The results of accuracy tests showed that a negligible error was introduced by the approximation.

II. Background

A. MapReduce and Hadoop

MapReduce is a programming model for processing large-scale data in a parallel way [2]. Users can easily implement distributed, parallel processing software by writing only two functions: *map* and *reduce*. The map function processes a sub-problem for input data and emits intermediate <*key*, *value*> pairs. The reduce function combines the values associated with the same key and produces the final output. Apache Hadoop [7] is the most popular open-source implementation of MapReduce.

$*$ Jae-Gil Lee is the corresponding author.

B. Girvan-Newman (GN) Algorithm

The GN algorithm is a divisive hierarchical clustering algorithm exploiting the concept of edge betweenness [1]. Three methods were proposed for the calculation of edge betweenness. Among them, the shortest-path method typically shows the best results. The *edge betweenness* of an edge is informally the number of shortest paths between pairs of nodes that pass through it. Since communities are loosely connected by a few "intergroup" edges, all shortest paths between different communities must pass through one of these few edges. Then, those edges connecting communities will have high edge betweenness. Thus, the communities are detected by eliminating such edges repeatedly. However, since the shortest path should be obtained for every pair of nodes, the GN algorithm is very costly.

III. Proposed Algorithm

A. Overview

SPB-MRA goes through *four* stages, as described in Fig. 1. The output of a stage is chained to the input of its next stage. Each stage executes its own map and reduce tasks. An iteration of these four stages produces a community detection result. In each iteration, Stage1 is executed multiple times, and the other stages are executed only once. The four stages repeat until the result quality no longer improves.

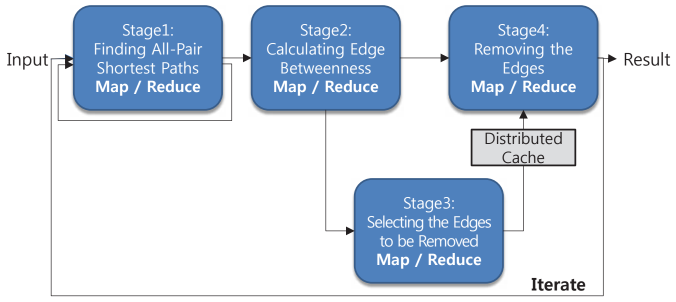
Input
Stage1: Finding All-Pair Shortest Paths Map / Reduce
Stage2: Calculating Edge Betweenness Map / Reduce
Stage4: Removing the Edges Map / Reduce
Result
Distributed Cache
Stage3: Selecting the Edges to be Removed Map / Reduce
Iterate

Fig. 1. An overview of SPB-MRA.

A tuple of 7 elements below is maintained for each pair of nodes in the process of community detection. It holds the network structure (*i.e.*, an adjacency list), the shortest path obtained so far, and so on. The term "tuple" in this paper refers to a tuple of these elements.

- ***targetId*** indicates the destination node of a shortest path and is initially set to be *sourceId*.
- ***sourceId*** indicates the source node of a shortest path and is initially set to be *targetId*.
- ***distance*** indicates the length of a shortest path and is initially set to be 0. The value of *distance* is updated in each iteration of Stage1.
- ***status*** indicates the status of a specific path. a is "active", and i is "inactive" meaning that the shortest path is already detected.
- ***weight*** indicates the number of the shortest paths from *sourceId* to *targetId* and is initially set to be 1.
- ***pathInfo*** indicates the list of the nodes on a shortest path and is initially set to be *null*.

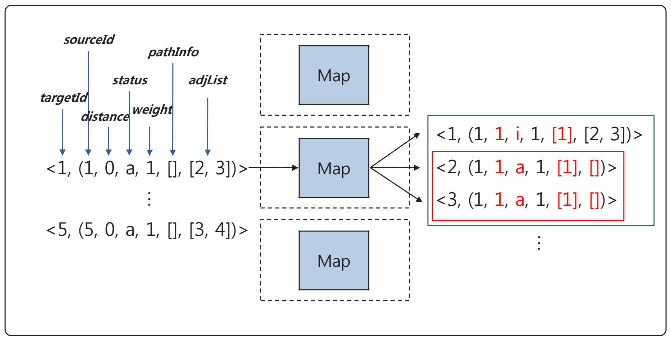
sourceId
pathInfo
status
adjList
targetId
distance
weight
<1, (1, 0, a, 1, [], [2, 3])>
⋮
<5, (5, 0, a, 1, [], [3, 4])>
Map
Map
Map
<1, (1, 1, i, 1, [1], [2, 3])>
<2, (1, 1, a, 1, [1], [])>
<3, (1, 1, a, 1, [1], [])>
⋮

Fig. 2. Map stage of Stage1.

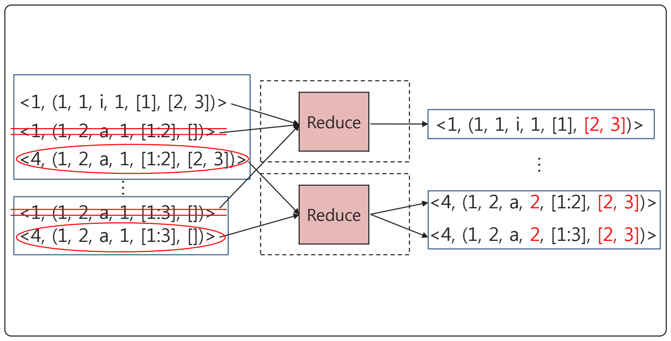
<1, (1, 1, i, 1, [1], [2, 3])>
~~<1, (1, 2, a, 1, [1:2], [])>~~
<4, (1, 2, a, 1, [1:2], [2, 3])>
⋮
~~<1, (1, 2, a, 1, [1:3], [])>~~
<4, (1, 2, a, 1, [1:3], [])>
Reduce
Reduce
<1, (1, 1, i, 1, [1], [2, 3])>
⋮
<4, (1, 2, a, 2, [1:2], [2, 3])>
<4, (1, 2, a, 2, [1:3], [2, 3])>

Fig. 3. Reduce stage of Stage1.

- ***adjList*** indicates the list of the nodes adjacent to *targetId*.

B. Stage1: Finding All-Pair Shortest Paths

In this stage, the shortest path between every pair of nodes in the network is calculated. For this, we adopted a multi-source message passing model proposed by Zeng *et al.*[8].

In the map stage, the frontiers are expanded from every node in the network to its adjacent nodes. For an input tuple, if *status* is i, no operation is needed; if *status* is a, *status* is changed to i, 1 is added to *distance*, and *targetId* is added to *pathInfo*. The tuple is emitted to the reduce stage. In addition, new tuples are generated by assigning each node in *adjList* to *targetId*. For these newly generated tuples, *status* is set to be a, *adjList* is set to be *null*, and the other elements are set to be the same as those of the tuple emitted just before. In this way, all paths between *sourceId* and *targetId* are generated and sent to a reducer. Fig. 2 shows an example of the map stage.

In the reduce stage, among the tuples sharing *sourceId* and *targetId*, only the tuple that has the minimum value of *distance* survives. If two or more tuples have the same minimum, *weight* is changed to the number of such tuples to remember the multiplicity of the shortest path. Fig. 3 shows an example of the reduce stage. These map and reduce stages repeat until all tuples have i for *status*.

C. Stage2: Calculating Edge Betweenness

In this stage, the edge betweenness for every pair of nodes in the network is calculated. In the map stage, unity is divided to each edge on a shortest path according to the total number (*i.e.*, *weight*) of the shortest paths sharing *sourceId* and

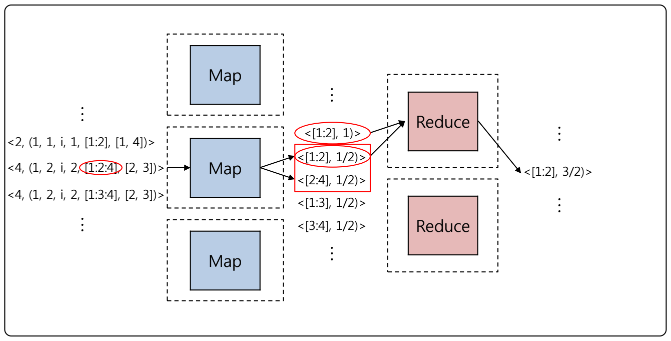
Map
Map
Map
<2, (1, 1, i, 1, [1:2], [1, 4])>
<4, (1, 2, i, 2, [1:2:4], [2, 3])>
<4, (1, 2, i, 2, [1:3:4], [2, 3])>
<[1:2], 1)>
<[1:2], 1/2)>
<[2:4], 1/2)>
<[1:3], 1/2)>
<[3:4], 1/2)>
Reduce
Reduce
<[1:2], 3/2)>

Fig. 4. Map stage and reduce stage of Stage2.

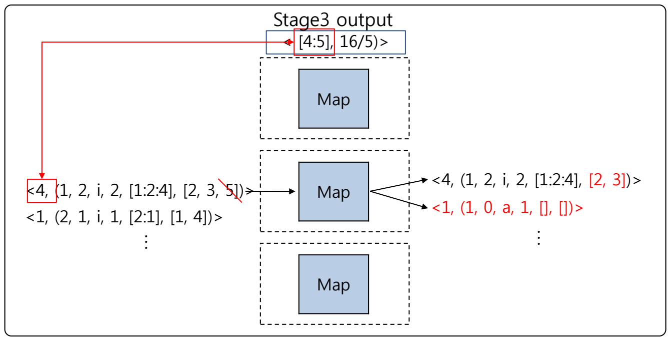
Stage3 output
<[4:5], 16/5)>
Map
Map
Map
<4, (1, 2, i, 2, [1:2:4], [2, 3, 5])>
<1, (2, 1, i, 1, [2:1], [1, 4])>
<4, (1, 2, i, 2, [1:2:4], [2, 3])>
<1, (1, 0, a, 1, [], [])>

Fig. 6. Map stage of Stage4.

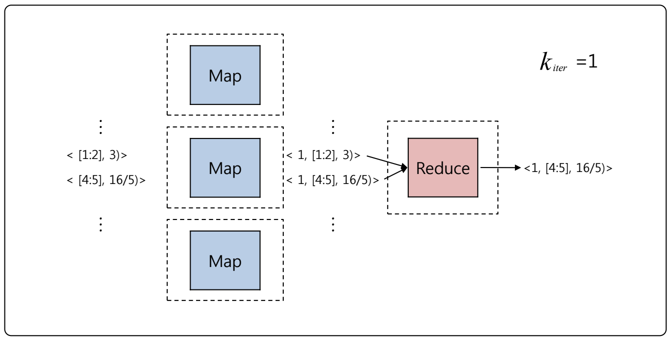

Fig. 5. Map stage and reduce stage of Stage3.

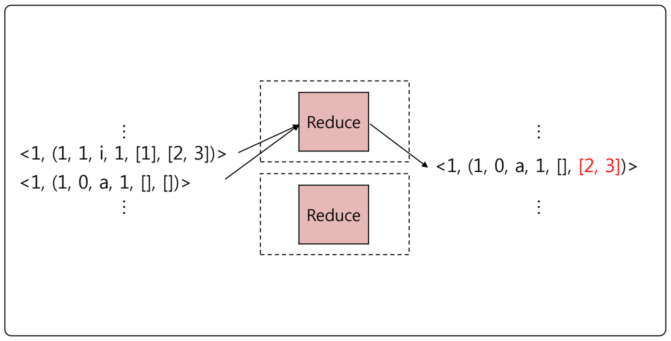
<1, (1, 1, i, 1, [1], [2, 3])>
<1, (1, 0, a, 1, [], [])>
Reduce
Reduce
<1, (1, 0, a, 1, [], [2, 3])>

Fig. 7. Reduce stage of Stage4.

targetId. In the reduce stage, the contribution of each shortest path is summed up for each edge. Fig. 4 shows an example of the map stage and the reduce stage.

D. *Stage3: Selecting the Edges to be Removed*

In this stage, k_{iter} edges are selected according to edge betweenness. k_{iter} is specified by a user as a tuning parameter. In the map stage, no operation is needed. In the reduce stage, edges are sorted in the decreasing order of edge betweenness, and the top-k_{iter} edges are selected. We simply run only one reducer to get a globally sorted result. The selected edges and their value of edge betweenness are stored in the distributed cache such that all mappers of Stage4 can access the information. Fig. 5 shows an example of the map stage and the reduce stage.

Note that SPB-MRA selects multiple edges per iteration whereas the GN algorithm only one edge. That is, we remove k_{iter} edges in one iteration instead of iterating k_{iter} times. Thus, this approximation can speed up community detection by k_{iter} times. Our reasoning is that the edge with the highest value of edge betweenness in the next iteration tends to have a high value in the current iteration, too. The next section proves that our reasoning is indeed correct.

E. *Stage4: Removing the Edges*

In this stage, the edges selected by Stage3 are removed from the network. Then, a new set of tuples are generated to reflect the removed edges since edge betweenness needs to be recalculated in the next iteration. Note that edge betweenness changes if a shortest path ran along one of the removed edges. In the map stage, if *targetId* of a tuple from Stage2 is affected by the edges selected in Stage3, its *adjList* is updated to represent a new network by removing the corresponding node. Other tuples are initialized for the next iteration. In the reduce stage, all tuples are made to have an updated value for *adjList*. These tuples are provided to the input of Stage1 for the next iteration. Fig. 6 and Fig. 7 show an example of the map stage and the reduce stage.

IV. Performance Tests

A. *Data and Environment*

We used two sets of collaboration networks available at Stanford Large Network Dataset Collection [9]. Table I shows the details of the data. We used Java version 1.6.0 and Hadoop version 1.0.4 to implement SPB-MRA. The performance tests were conducted on a cluster that consists of 12 Amazon EC2 m1.xlarge instances.

TABLE I. Data set description.

Data Statistics	ca-GrQc	ca-HepTh
Nodes	5242	9877
Edges	28980	51971
Type	Undirected	Undirected
Average Clustering Coefficient	0.5296	0.4714
Number of Triangles	48260	28339
Fraction of Closed Triangles	0.6298	0.284
Diameter	17	17
90-percentile Effective Diameter	7.6	7.5

B. *Scalability*

In order to show the scalability of SPB-MRA, we measured elapsed time for one iteration while varying the number of reducers from 1 to 32. Fig. 8 and Fig. 9 show that elapsed time

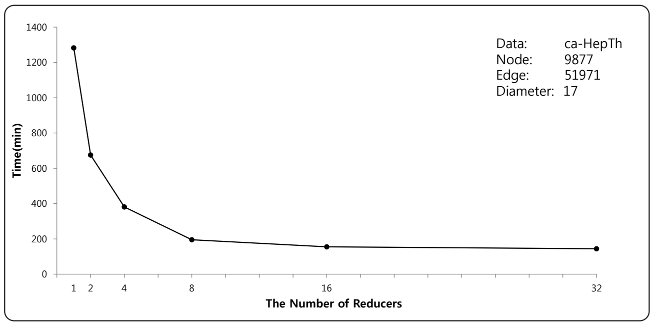

Fig. 8. Elapsed time for the "ca-HepTh" data set.

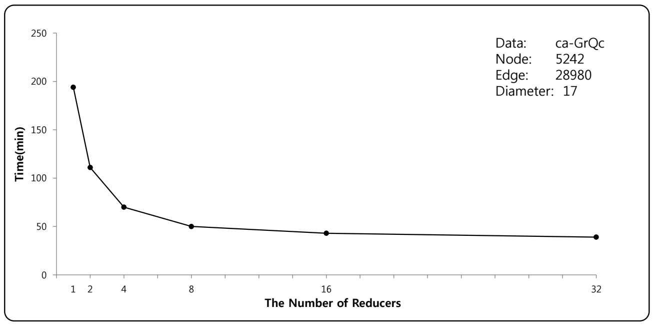

Fig. 9. Elapsed time for the "ca-GrQc" data set.

decreased almost linearly as the number of reducers increased until 8. This number of reducers was sufficient for these data sets, and adding more reducers was not effective. Since the "ca-HepTh" data set is twice larger than the "ca-GrQc" data set, the slope of the improvement became flat earlier in Fig. 9 than in Fig. 8.

C. Approximation Accuracy

In order to show the approximation accuracy of SPB-MRA, we measured the F-score [10] with different k_{iter} values. k_{iter} means the number of edges to be removed for one iteration. The ground truth for the F-score is the set of 40 edges removed by selecting only one edge per iteration, as described in the original GN algorithm. In Table II, as the value of k_{iter} increased, the error also increased because the edges not really having the highest edge betweenness could be removed more likely. However, even though we removed four edges at once, the F-score decreased only by 10%. Thus, it is shown that we can speed up by four times with only 10% error.

TABLE II. ACCURACY OF THE APPROXIMATION TECHNIQUE.

k_{iter}	Iteration	# of edges removed	F-score
1	40	40	1.0
2	20	40	0.875
4	10	40	0.9
5	8	40	0.875
8	5	40	0.9
10	4	40	0.8

V. Conclusion

In this paper, we proposed an algorithm called SPB-MRA, a parallel version of the GN algorithm to support large-scale networks. We utilized the MapReduce model to design the algorithm and implemented it on top of Hadoop. We conducted performance tests for SPB-MRA on Amazon EC2 instances. The results showed that elapsed time decreased almost linearly with the number of reducers increased and the error rate of the approximation technique was negligible. Our future work includes further improving the performance of SPB-MRA by introducing additional approximation techniques.

Acknowledgment

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

References

[1] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, 2004.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "Scope: easy and efficient parallel processing of massive data sets," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, 2008.

[4] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton, "Mad skills: new analysis practices for big data," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1481–1492, 2009.

[5] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414–1425, 2009.

[6] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-Reduce-Merge: simplified relational data processing on large clusters," in *Proceedings of 2007 ACM SIGMOD International Conference on Management of Data*, 2007, pp. 1029–1040.

[7] "Apache Hadoop," http://hadoop.apache.org/, accessed: August 1, 2013.

[8] Z. F. Zeng, B. Wu, and T. T. Zhang, "A multi-source message passing model to improve the parallelism efficiency of graph mining on MapReduce," in *Proceedings of 2012 IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012, pp. 2019–2025.

[9] "Stanford large network dataset collection," http://snap.stanford.edu/data/, accessed: August 1, 2013.

[10] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*, 2nd ed. Morgan Kaufmann, 2006.