Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 113–120
doi: 10.14794/ICAI.9.2014.2.113

Efficiency issues of computing graph properties of social networks*

Péter Englert, Márton Balassi, Balázs Kósa, Attila Kiss

Eötvös Loránd University
`{enpraai,bamrabi,balhal,kiss}@inf.elte.hu`

Abstract

In this paper we consider three different properties of social networks, namely the number of the different triangles, the sizes of the strongly connected components (SCC) and Linerank, which is offered as a substitute for betweenness and can be efficiently calculated on graphs with more than 10^9 nodes [7]. We implemented the computation of these measures in three different ways: sequentially, using the widely spread MapReduce model – Hadoop in our case – and its newly emerging rival Giraph, which is based on the Pregel system [11]. MapReduce and Pregel use distributed computation enabling high parallelization of the calculations. The algorithms calculating the above measures can be grouped into three different families. In the case of the triangles for a node only its close neighbourhood should be taken into consideration. In the case of the SCC's at least in the sequential approach the graph should be traversed in a depth search manner, which can be prohibitively expensive for the distributive systems. Finally, Linerank can be computed by using an iterative matrix-vector multiplication. We ran our implementations on several generated and real-world networks of different sizes firstly in order to assess the degree of the benefit of parallelization offered by the last two technologies for the algorithm families. Secondly, to see for which magnitude of the size of the networks it is more beneficial to choose one the aforementioned models instead of the sequential approach.

Keywords: Social Networks, MapReduce, Pregel, Graph Measures

MSC: 68Q85

*This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013).
Besides, this work was completed with the support of the Hungarian and Vietnamese TET (grant agreement no. TET 10-1-2011-0645).

1. Introduction

In the last decade the size of the data that is to be analyzed both in scientific experiments and practical applications has become enormously large in many occasions, which posed several challenges to computer science and to the development of information technologies. Besides the new algorithms, new programming models have also emerged that refined the methodologies and systems of distributed computation. The most well-known invention is probably the MapReduce paradigm [3] that has already broken through the boundaries of laboratories of research institutions and has appeared in the praxis of larger companies. In MapReduce the data distribution, replication, fault-tolerance and load balancing is handled automatically. Basically, the user has to write two functions, namely *Map* and *Reduce*. Map takes a set of key/value pairs as input and produces a set of *intermediate* key/value pairs. The system groups together all intermediate values belonging to the same intermediate key and sends them to the Reduce function. For each intermediate key the Reduce function receives all values corresponding to a given key and performs a user-defined computation on them. The resulting key/value pairs are the output of the program or can be passed to a different Map function as input.

The execution of the Map function is distributed across several machines by automatically partitioning the input data. The Reduce function is also run in a distributed way by R machines. Each map worker hashes its output set of intermediate key/value pairs into R groups and a special worker, i.e., the master notifies the reduce workers about the locations of the data belonging to them.

MapReduce is a general purpose programming model which on one hand is a clear advantage, while on the other hand in certain scenarios this generality may have disadvantages as well. For example in the case of graphs algorithms, when the algorithm is implemented as a sequence of Map/Reduce functions, often the whole graph structure has to be passed from one Map/Reduce unit to the next. Since the research of the networks has also become highly intensive in the last decade Malewicz et al. in Google has developed another system, Pregel [11] to overcome this shortcoming of MapReduce.

The input to a Pregel computation is a directed graph in which to each vertex and edge a unique identifier and a modifiable user defined value is assigned. The directed edges are associated with their source vertices and they also store the id of their target vertex. The calculation consists of a sequence of iterations, called *supersteps*. Within each superstep the vertices execute a user-defined function in parallel. A vertex can modify its state as well as the state of its outgoing edges, process the messages received in the previous superstep and send messages to other vertices (not necessarily to their neighbours) that are to be received in the next superstep. In addition it may change the topology of the graph. The computation terminates when all vertices vote to halt. The output is the set of values returned by the vertices [11].

In our experiments we compared the performance of the previous models to each other and with the performance of the sequential approach. In our implementations we used Hadoop [6] and Giraph [5] that are open source implementations of MapReduce and Pregel respectively. We considered three different properties of social networks, namely the number of the different triangles, the sizes of the strongly connected components (SCC) and Linerank [7]. The algorithms calculating these measures represent three different families of algorithms. In the case of the triangles for a node only its close neighbourhood should be taken into consideration. In the case of the SCC's at least in the sequential approach the graph should be traversed in a depth-first search manner, which is difficult to parallelize. Finally, Linerank can be computed by using an iterative matrix-vector multiplication. More details about Linerank, which was offered as a possible substitute for betweenness, can be found in [7].

We ran our implementations on several generated and real-world networks of different sizes ranging from $875,713$ nodes to 20 million. As expected, Giraph outperformed Hadoop in all cases. In most of the time the sequential model also performed better than Hadoop, which in the case of the Linerank is quite surprising considering the fact that matrix-vector multiplication is considered to be effectively implementable in Hadoop. For triangle counting however, Hadoop ran faster almost in all cases. As we shall see in this case the Hadoop implementation accomplishes only two Map/Reduce function pairs. Finally, for the graph with 20 million vertices Giraph outperformed the sequential approach in all cases, which clearly indicates that for larger graphs it is more beneficial to choose this model. Nevertheless, for smaller graphs in most of the cases the sequential model conquered Giraph even for Linerank whose implementation in Giraph seems to be quite effective again.

The paper is organized as follows. In Sections 2.1., 2.2. and 2.3. the algorithms computing the number of triangles, finding the SSC's and calculating Linerank are sketched respectively. Then in Section 3. the experimental results are explained in more detail. Finally, in Section 4. we conclude by summarizing our work.

2. The description of the algorithms

In this section we provide the detailed description of the three chosen algorithms focusing on the differences between the sequential, Hadoop and Giraph implementations. As a result of the distributed frameworks the chosen graph representation is the adjacency list.

2.1. The number of triangles

The high number of closed triangles is a well-known characteristic of a social network thus providing this measure importance.

The naive sequential algorithm simply performs self-search bounded in depth three from each node. This counts each triangle three times, to eliminate this overhead it is a trivial improvement to utilize the ordering on node IDs to count every triangle exactly once.

The algorithm of the Giraph implementation is essentially the same, while during the Hadoop implementation the structure of the graph needs to be passed between the computation phases as shown in the following pseudo-code.

```
1: class BiDirectionMap
2: function MAP(vertex $v$, neighbors $[n_1, n_2, \dots n_k]$)
3:     for all vertex $n \in$ neighbors do
4:         function EMIT($v$, ($n$, out))
5:         if $ID(n) > ID(v)$ then
6:             function EMIT($n$, ($v$, in))
7: end class
```

Figure 1: Pseudo-code of the first Map phase

2.2. The size of the strongly connected component

Finding the strongly connected components (SCCs) of a graph is a well-known problem, and the resulting SCC decomposition has many applications in graph analysis. [2] Additionally, SCCs are defined through reachability, which requires traversal of the graph. Because of this, it requires algorithms that are fundamentally different from, for example, the calculation of matrix-based centralities such as Pagerank[13], or highly local algorithms like triangle counting.

Unfortunately, although multiple efficient sequential algorithms exist for the calculation of the SCC decomposition, they are difficult to parallelize. In our sequential implementation, we calculate strongly connected components using a one-pass path-based algorithm which runs in linear time. [4] Since this algorithm, like other fast algorithms for finding strongly connected components, requires depth-first search, it does not lead to an efficient parallel implementation. For this reason, significantly different algorithms are used for the *Hadoop* and *Giraph* implementations.

A comprehensive survey on distributed algorithms for finding the SCCs of a graph is given by Barnat el al. [2] Our *Hadoop* implementation is based on the *Colouring/Heads-off* algorithm, which was originally introduced by McLendon et al. [12]

Initially, each node is associated with a unique integer ID. Nodes send their ID along their outgoing edges. If a node receives a smaller ID then its current one, it adapts the smaller ID and propagates it. When no messages are sent, the first phase ends, and each node now has the smallest original ID from the set of nodes from which it is reachable. Nodes in the same SCC have the same ID, so edges connecting nodes with different IDs are removed. We concentrate on the nodes which retained their original ID. After reversing the edges, the nodes reachable from such a node form one SCC. These components can be found using a similar label propagation. The found components are output and removed from the graph. This process is iterated until no nodes are left.

The same algorithm was implemented both in Hadoop and in Giraph, with only slight modifications due to the technical differences in the frameworks.

2.3. Linerank

Recently, a node centrality measure called Linerank was introduced as a possible substitute for the computationally expensive betweenness centrality. [7] The score of a node is calculated by aggregating the importance of its incident edges. The importance of an edge is defined as the stationary probability of a random walk with restarts visiting the edge. Besides its utility, this algorithm was chosen to be implemented because it represents a broader class of distributed graph algorithms, namely, matrix-vector multiplication based iterative algorithms.

To understand how Linerank works, let us first define *line graphs*. A line graph $L(G)$ of a graph G describes the adjacencies of its edges. Each node of $L(G)$ represents an edge of G. If G is directed, then $L(G)$ is also directed, and there is a directed edge in $L(G)$ from e_1 to e_2 if and only if in G, the target of the edge represented by e_1 is the same as the source of the edge represented by e_2. Similarly, if G is undirected, then $L(G)$ is also undirected, and e_1 and e_2 in $L(G)$ are adjacent if and only if there exists a node in G which is incident to both of the corresponding edges. In Linerank, the importance of an edge is defined as its Pagerank score in the line graph. This can be calculated using iterative matrix-vector multiplication. [14] Based on the observation that the adjacency matrix of the line graph is the product of the source and target incidence matrices of the original graph, the computation can be sped up by using the two sparse n by e incidence matrices instead of the potentially dense e by e adjacency matrix, where n denotes the number of nodes and e is the number of edges in the original graph.

As the MapReduce implementation of the Pagerank algorithm has been described many times before, [14] and the sequential implementation of the same algorithm is straightforward, we omit their description here. The implementation of the final step, i.e., summing the scores of the incident edges for each node, is also straightforward.

However, the Giraph implementation of the PageRank calculation deserves an explanation. First, we note that the PageRank score of edges with the same starting vertex is equal. Since the random walk visits edges via nodes, the chance of staying at an edge is proportional to the chance of arriving at its starting node. Additionally, we are dealing with unweighted graphs, so each edge starting at a particular node receives an equal share of this chance.

In the Giraph implementation, each node stores the chance of arriving at that particular node. In an iteration step each node receives along its incoming edges the probability that the random walker arrives to this node via the given edge. Then, after summing, normalizing and dampening these values, the resulting probabilities are propagated to the adjacent nodes along the outgoing edges. In the final step, each node knows the stationary probabilities of its outgoing edges and receives the stationary probabilities of its incoming edges, so it simply aggregates these and outputs its own Linerank score. Convergence is kept track of by using Giraph *aggregators*. Each node calculates the change in probability for its outgoing edges. The maximum is aggregated and if it is smaller than an ϵ constant, each node outputs the results and votes to halt.

3. Experiments

	Seq	H-10	H-20	H-40	G-10	G-17
web-Google	9.061	3281.479	4058.652	5194.457	195.307	237.804
wiki-Talk	26.092	1024.194	1118.83	1490.222	102.129	113.963
soc-LiveJ	60.89	2343.39	2321.948	2834.056	218.258	205.136
forest10M	105.607	4568.531	4367.449	5242.808	289.995	280.08
forest20M	560.021	6699.063	5573.117	6302.713	758.745	381.896

(a)

	Seq	H-10	H-20	H-40	G-10	G-17
web-Google	14.51	124.58	162.25	150.27	52.10	51.70
wiki-Talk	913.36	631.83	575.86	563.32	103.05	105.29
soc-LiveJ	3138.28	2194.22	1880.25	1663.37	355.63	281.08
forest10M	336.40	176.13	168.37	151.55	66.56	52.45
forest20M	362.61	277.58	212.26	180.49	107.49	69.80

(b)

	Seq	H-10	H-20	H-40	G-10	G-17
web-Google	36.184	1755.529	2192.617	155.281	176.961	36.184
wiki-Talk	51.434	1463.529	915.685	145.097	155.601	51.434
soc-LiveJ	67.41	2764.598	2973.812	748.551	537.495	67.41
forest10M	315.21	1873.53	2064.873	250.466	235.531	315.21
forest20M	840.701	2618.853	2533.859	566.999	338.821	840.701

(c)

Figure 2: The computation times of the different implementations on different graphs given in seconds. H-i and G-j respectively abbreviates Hadoop with i workers and Giraph with j workers. (a) The strongly connected components. (b) The number of triangles. (c) Linerank.

In our experiments we used three real-world networks, namely web-Google, wiki-Talk and soc-LiveJournal. In web-Google ($875,713$ nodes, $5,105,039$ edges) vertices represent web pages and directed edges represent edges between them [10]. In wiki-Talk ($2,394,385$ nodes, $5,021,410$ edges) vertices represent users and a directed edge is added from user i to user j, if i had edited at least once a talk page of j [9]. Finally, in soc-LiveJ ($4,847,571$ nodes, $68,993,773$ edges) vertices correspond for users and directed edges represent friendships [1]. Besides these networks, applying a slight variation of the forest fire model [8] we generated two random graphs with 10, 20 million nodes and $24,024,725$, 48097267 edges respectively. The generated graphs of the original model do not contain any directed triangles, therefore for each new edge that was to be added we kept its original direction with probability 0.7, reversed its direction with probability 0.1 and added two edges directed in both ways with probability 0.2[1]. In the tests each PC ran with 2Gb internal memory. They either have 2 Dualcore I5 CPU with 3,2 GHz clock speed, or 1 Dualcore I5 CPU with 3 GHz clock speed.

The computation times can be found in Figure 2 measured in seconds. As expected, Giraph outperforms Hadoop in all cases. In the case of the strongly connected components the sequential approach always runs faster than Hadoop. For the largest graph (with 20 million nodes) Giraph with 17 workers outperforms the sequential implementation, otherwise the latter runs faster. For smaller graphs Giraph-10 is faster than Giraph-17 owing to the increased cost of communication, however, on larger graphs, since the level of parallelization is higher, Giraph-17 performs better. A similar statement can be made for Hadoop-10 and Hadoop-20.

In the case of the triangle counting Giraph outperforms the sequential approach almost in all cases. In fact, for large graphs, there is an order of magnitude difference between the running times. Hadoop also performs better than the sequential approach for even a graph with 2.3 million nodes. Remember that here the Hadoop implementation accomplishes only two Map/Reduce function pairs, which explains the relatively fast computation times.

Finally, for Linerank again, the sequential approach always outperforms Hadoop by at least an order of magnitude. However, for graphs of 10 and 20 million nodes Giraph becomes faster than the sequential model.

4. Conclusions

In our paper we considered three measures of social networks whose computing methods belong to three different algorithm families. We implemented these methods in three different programming models, namely sequential, MapReduce and Pregel, and compared their performances. Not surprisingly, Giraph outperformed Hadoop in all cases. However, the sequential model also often ran faster than Hadoop, which is a more striking result for the algorithm based on iterative matrix-vector multiplication, since this operation is known to be effectively implementable in Hadoop. Finally, for the largest graph with 20 million vertices Giraph outperformed the sequential approach in all cases, which clearly indicates that for bigger graphs, it is more beneficial to choose this model in practically all scenarios.

[1]The forward and reverse burning probabilities were set to 0.37 and 0.2 respectively.

References

[1] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 44–54, 2006.

[2] Jiří Barnat, Jakub Chaloupka, and Jaco Van De Pol. Distributed algorithms for scc decomposition. *J. Log. and Comput.*, 21(1):23–44, feb 2011.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, 2004.

[4] Harold N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74:2000, 2000.

[5] Giraph information. `http://giraph.apache.org/`.

[6] Hadoop information. `http://hadoop.apache.org/`.

[7] U Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. Centralities in large networks: Algorithms and observations. In *SDM*, pages 119–130. SIAM / Omnipress, 2011.

[8] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, 2006.

[9] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1361–1370, 2010.

[10] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008. cite arxiv:0810.1355Comment: 66 pages, a much expanded version of our WWW 2008 paper.

[11] 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 *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 135–146, 2010.

[12] William McLendon III, Bruce Hendrickson, Steven J. Plimpton, and Lawrence Rauchwerger. Finding strongly connected components in distributed graphs. *Journal of Parallel and Distributed Computing*, 65(8):901–910, aug 2005.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, 1998.

[14] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.