SCALABLE LOCAL COMMUNITY DETECTION WITH MAPREDUCE FOR LARGE NETWORKS

Ren Wang, Andong Wang, Talat Iqbal Syed and Osmar R. Zaïane

Department of Computing Science, University of Alberta, Canada

ABSTRACT

Community detection from complex information networks draws much attention from both academia and industry since it has many real-world applications. However, scalability of community detection algorithms over very large networks has been a major challenge. Real-world graph structures are often complicated accompanied with extremely large sizes. In this paper, we propose a MapReduce version called 3MA that parallelizes a local community identification method which uses the M metric. Then we adopt an iterative expansion approach to find all the communities in the graph. Empirical results show that for large networks in the order of millions of nodes, the parallel version of the algorithm outperforms the traditional sequential approach to detect communities using the M-measure. The result shows that for local community detection, when the data is too big for the original M metric-based sequential iterative expension approach to handle, our MapReduce version 3MA can finish in a reasonable time.

KEYWORDS

Social Network Analysis, Community Mining, MapReduce

1. INTRODUCTION

Huge amounts of data is represented in the form of Information Networks. An "*Information Network*" consists of vertices and edges to model large complex real-world systems. Each vertex (node) is an entity, e.g., a person, a web page, a paper, a protein, or other objects, and each edge indicates a relationship between two entities, e.g., friendship, hyperlinks, protein interaction, etc. Network Analysis unravels the hidden structure in the complex network data. Analyzing and extracting information from information networks can have myriad applications in academic and industrial domains, especially with the large amount of data generated by the Social Network. Besides Social Networks, Network Analysis can also be applied on other domains such as the World Wide Web, biological networks, co-citation networks, crime networks, etc.

Many social networks exhibit the property of containing community structures [1] and based on this, new products and services are recommended [2]. Although there is no universally accepted definition of *community*, intuitively, a community can be considered as group of nodes that are densely connected while being sparsely connected to nodes in the other communities. The metrics to define a community have been discussed in [3], [4], [5]. Approaches to identify a community can be broadly classified into two categories, *partitioning* and *hierarchical clustering*. The partitioning method divides the network into sub-networks and identifies the communities based on their topological structure. A wide variety of heuristic algorithms have been developed based on this idea. For example, multilevel partitioning [6], k-partite graph partitioning [7], relational clustering [8].

The hierarchical clustering approach finds the structure of networks as a hierarchy of communities in the form of a dendrogram. The communities are formed based on various similarity metrics, such as modularity and betweenness [9], [10], [11].

Traditional community detection algorithms suffer from a drawback of having quadratic time complexities and memory requirements. Moreover, most existing algorithms for community mining detect communities using global information, that is they require access to the whole network in memory. The most common and widely used such approaches are Newman's Q-Modularity metric [10] which considers the number of edges within a community minus the expected number of such edges in a random network with the same nodes and the same number of edges globally. Given the intractability of global information for many of the large scale networks in real applications, the idea of exploring the network through local expansion was introduced [12]. These approaches are known as local community mining, as opposed to global community mining, and they are less sensitive to the network size. However, they only find communities in a targeted subset of the total network.

With recent emergence of big data, analysis of information networks using conventional algorithms is almost impossible to be processed in a single machine. Also, parallelizing the existing community detection algorithms is not straight-forward. We are interested in parallelizing a local community mining approach, as it would not require global knowledge of the network, yet we want our solution to be tractable and tackle the whole large network in a reasonable time.

All the solutions proposed in the literature such as [12], [3], [4], [5] are designed for sequential execution on a single system. To overcome this limitation, in this paper, we propose a parallel version of community detection algorithm using the M-metric in a data parallel way, scalable to analyze very large information networks in the order of millions of nodes. The parallel algorithm is implemented in a distributed environment using the MapReduce framework. We call our proposed algorithm *3MA* for M Metric MapReduce Algorithm.

The rest of the paper is organized as follows. Section 2 defines the problem and reviews existing algorithms. We give a description of our approach 3MA in Section 3 and report the experimental results in Section 4, followed by our conclusion in Section 5.

2. PRELIMINARIES

Here we give the definition of the algorithm and review the existing related work.

2.1. Problem Definition

An information network can be defined as an undirected and unweighted graph, G=(V,E), where V is the set of Vertices (Nodes) and E is the set of edges. Given an information network, the task is to extract a set of non-overlapping communities (with each node in a graph belonging to only one community) using multiple distributed systems in a data parallel way, while using local information. We are only focusing on local community mining as it is more accessible for parallelization. Global approaches would demand shared memory to access global information and thus could be illusive for this goal.

2.2. Community Discovery Metrics

The principle of local community mining is to start with a seed of well-connected nodes, possibly one unique initial node, and expand the reachability of these nodes to include other nodes, nodes that are connected to the nodes already in the community. The community continues to expand while a given metric is improving and stops when this metric cannot be improved. The expansion consists of bringing into the community new nodes directly connected to the group. At each given time in this iterative process, there is a group of identified nodes for which we know their connections. The nodes in the community that are only connected to other nodes in the community, are called core nodes. Nodes that are also connected to outside nodes, are known as border nodes. Border nodes should have more connections inside the community than outside. The expansion of the community consists of assessing outside nodes connected by boarder nodes and including them in the community if a certain metric is improved. When no new outside node can be integrated without deteriorating the metric, the process terminates and a community is declared.

There are three major metrics, *R-metric* [12], *L-metric* [5] and *M-metric* [3], that have been proposed in the literature for community evaluation.

Modularity *R* [12] is proposed for the community evaluation problem. The metric is based on the observation that a community would have a sharp boundary which has fewer outward connections and more inward connections. Boundary nodes are defined as nodes with at least one neighbour outside the community. It can be seen that Local Modularity *R* measure focuses on the boundary sharpness to evaluate the quality of the discovered community. The *R* measure is given as $R=B_{in\_edge}/(B_{out\_edge}+B_{in\_edge})$ where B_{out_edge} is the number of edges connecting boundary nodes and nodes outside the community, and B_{in_edge} is the number of edges connecting boundary nodes and other nodes within the same community.

Chen et al. proposed a local community measure, the *L-metric* along with a two-phase algorithm [5]. The definition for the community is based on the intuition that the nodes within the communities have relatively more connections among each other than with the nodes outside the community. The average number of connections for each node is a better measure than the absolute number to avoid the size factor of the communities while computing the L-metric. The L-metric can be defined as

$L=L_{in}/L_{ex}$ with $L_{in}=(\Sigma_{i\varepsilon D}\ IK_i)/|D|$ and $L_{ex}= =(\Sigma_{j\varepsilon B}\ EK_j)/|B|$ where D is the community consisting of all nodes within the community, and B is the set of boundary nodes in the community. IK_i is the number of edges between node i and nodes in D and EK_j is the number of edges between node j and nodes outside the community. Only boundary nodes are considered when computing L_{ex} since other nodes do not have connections outside the community.

Another metric is the M-metric proposed by Luo et al. [3], which is a ratio of the number of connections within a community to the number of connections from the community to others. M=ind(D)/outd(D) where ind(D) is the number of connections between nodes that belong to the community D and outd(D) is the number of connections between nodes belonging to a community to the nodes not belonging to the same community. A community D with an M-measure greater than 1 is called as a *true community*.

We opted to parallelize an iterative expansion with the M-metric because it appeared less complicated to convert into the MapReduce framework. Using a similar strategy for other measures, such as the L-metric, remains an open problem. We have yet to find a way to transmute the L-metric into MapReduce.

2.3. Local Community Detection using M-metric

In [3], the authors also provide an algorithm for Local Community Detection based on the Local Modularity, M-Metric. Given a graph G with a set of nodes, N, and a set of vertices, V, Algorithm 1 finds a local community of a randomly selected node from N. The community is identified by finding the M-gain (increase of the M-measure) of all nodes adjacent to the currently available nodes in the community (starting with one random node) and adding the node with the largest M-gain to the community that increases the overall M measure of the community. Algorithm 1 provides the sequence of steps to find a local community with start node in a graph. S denotes Shell Nodes, a set of potential nodes that can be added to the community, denoted by D.

Algorithm 1: Local Community Detection Algorithm

```
Input:
        Network G and Start Node $n_0$
Output:
        A local community D for $n_0$

1: Add $n_0$ to D, add all $n_0$'s neighbors to S
2: Set M for $n_0$
3: repeat
4:      for each node $n_i$ in S
5:          Compute $M'_i$
6:      end for
7:      Find $n_i$ with the maximum $M'_i$, breaking ties randomly
8:      Add $n_i$ to D and remove it from S
9:      Update S, M
10: until $M' < M$
11: Return D as $n_0$'s local community
```

In Algorithm 1, in each step, a new neighbor node with the largest M is added to the community, breaking ties randomly. The process is continued until there are no candidate nodes that can increase the overall M of the current community. The local detection algorithm can be extended to find multiple communities in a graph by iteratively repeating the local community detection algorithm on the set of nodes that have not been included in any community yet.

By applying Algorithm 1 iteratively to cover the whole graph, all the communities in the graph can be found. Algorithm 2 illustrates the steps required to extract all the communities.

Algorithm 2:Local Community Detection Algorithm to extract all the communities

```
Input:
        Network G and a randomly selected Start Node $n_0$
Output:
        A set of communities

1: Apply Algorithm 1 to find a local community $l_0$ for $n_0$
2: Insert neighbors of $l_0$ into the shell node set S
3: while |S|!= 0 do
4:      Randomly pick one node $n_i$  S
5:      Apply algorithm 1 to find a local community $l_i$ for $n_i$
6:      Remove nodes in S that are covered by $l_i$
7:      Update S by neighbors of $l_i$ that do not belong to any found community yet
8: end while
9: return a set of communities
```

2.4. Existing Parallel algorithms using MapReduce frameworks

The literature contains some work on parallelizing community detection algorithms using the MapReduce framework [2], [13], [14]. Chen et al. [13] and Moon et al. [2] developed a MapReduce version on existing algorithms. Both papers adapt approximation techniques by adding the top-k nodes or removing the top-k edges from their rank list. These methods suffer from a drawback of a randomly selected k value and expanding the communities iteratively based on the selected k. Shi et al. [14] proposed a community detection solution for massive-scale networks with Mapreduce using a similarity metric called Q-metric to detect communities using the hierarchical clustering approach. The authors propose a set of degree-based pre-processing and post-processing techniques that would improve both accuracy and performance. Our approach to detect communities using the MapReduce framework is along the same lines as explained in [14] but using the M-Metric and without approximations.

3. PARALLELIZATION OF LOCAL COMMUNITY DETECTION - THE 3MA ALGORITHM

This section explains the algorithm to run the local community detection in a data parallel way and how the algorithm could be implemented using the MapReduce framework.

3.1. Parallel Algorithm

Sequentially identifying the out-degree of all the nodes from the graph and the computation of the M-measure for the community with each potential node included in the community, consume most of the time in the local community detection algorithm. These tasks are data independent, i.e., it is not required to know about the complete graph while computing the overall M-measure. This allows the parallelization of the data independent tasks, and design a new algorithm that could improve the overall running time of the algorithm. The parallel version of local community detection is explained in Algorithm 3.

Algorithm 3: Parallel Version of Local Community Detection Algorithm

```
Input:
    Network G and a randomly selected Start Node $n_0$
Output:
    Local Community D for Node $n_0$

1: Add $n_0$ to community D
2: Compute the M-measure for $n_0$
3: Add all neighbors of $n_0$ to a set of Nodes, S, called the Shell Nodes
4: Send each potential node in S to different systems to be processed in parallel
5: Find the node, $n_i$, with the largest M-measure of the community and add it to the community D
if adding $n_i$ does not decrease community's overall M-measure
6: If $n_i$ is added, add all neighbours of $n_i$ to the set S and Goto 4
7: Return D as $n_0$'s local community
```

3.2. Implementation in MapReduce Framework

The above parallel version of Local Community Detection algorithm using M-measure can be implemented in a MapReduce Framework. The MapReduce programming paradigm encompasses two separate and distinct tasks performed by a parallel program running on a distributed cluster. The first is the map task, which takes some data and converts it into data with individual elements broken down into key-value pairs. The map step is typically for modeling the data in a form that can be used by a reducer. The Map function is performed by multiple worker nodes (mappers) in parallel.

The reduce task, which is always performed after the map task, takes the output from a map as input and aggregates the data sent by the mapper. The reduce step performs a summary operation, in which worker nodes process in parallel groups of output data per key. Any sequential aspect of an algorithm is executed on the reducer.

There are two different stages in implementing our algorithm using the MapReduce framework. The first stage called the *Initiation stage* identifies the M-measure for the randomly selected initial node by computing the in-degree and out-degree in parallel. This M-measure would be M-Measure of the community, since it contains only one node. Once the community has been initiated, the second stage called *Induction Stage* is invoked. The Induction Stage adds the potential nodes to the community. The *Induction Stage* is iterative, with each iteration adding a node to the community. The iteration stops once the node with the largest M gain for the community reduces the overall M-measure of the community. The overall block diagram is illustrated using Figure 1. Once the first community is identified, the *Induction Stage* is repeated for the expansion of other communities. *Initiation stage* is not repeated as its initial results are saved and exploited for subsequent invocations of the *Induction Stage*.

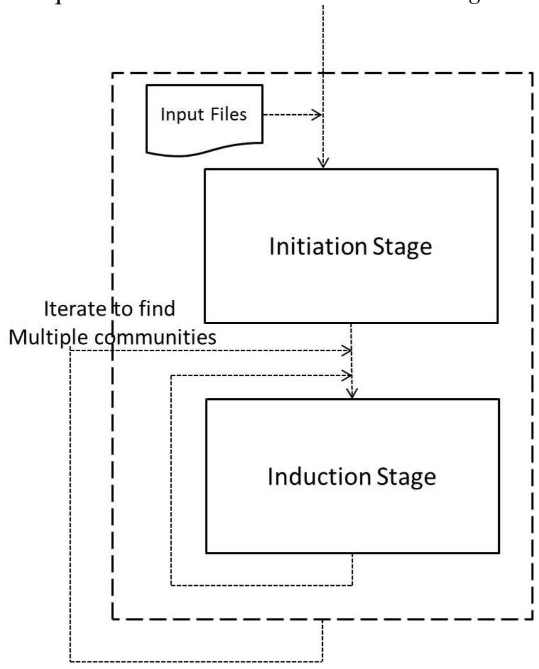

Figure 1. Overall Block Diagram of the parallel version of Local Community Detection using M-Metric

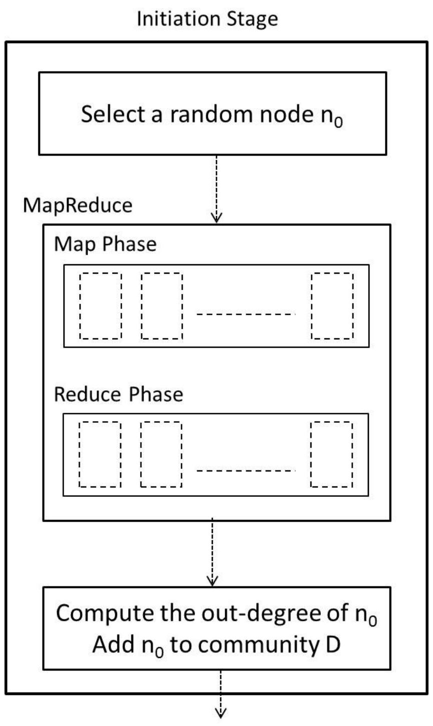

Figure 2. Initiation Stage - Flow through different components of the Initiation Stage including its Mapreduce phase

3.2.1. Initiation Stage

The initiation stage is to find the in-degree and out-degrees of the randomly selected node. This stage consists of a Map Phase and a Reduce Phase. The input to this stage is the graph containing a list of edges connecting various nodes. The flow is shown using Figure 2. Both mapping and reducing are done by parallel worker nodes in the cluster.

Map Phase: The Map Phase takes the input in the form of a set of (key, value) pairs. The key to this mapper is the offset (translating to every new line in the input file) and the value is the Edge itself. For large networks, the input is split into multiple entities called *data split* and sent to various different systems. The map function of the Mapper iterates through all the input values in the data split. Each map function produces an output in the form of a (key, value) pair with key being a node in the edge and the value of 1. The map phase of the initiation stage is explained in Algorithm 4.

Consider an example shown in Figure 3 with 5 vertices and 5 edges. For each pair of (key, value) the mapper would emit two outputs. Each vertex in an edge is emitted along with a value of 1 which denotes the number of occurrences of a vertex in an edge. Therefore for the example in Figure 3, the output pairs of each map phase would be {{(1,1),(2,1)}, {(1,1),(3,1)}, {(3,1),(4,1)}, {(2,1),(4,1)}, {(4,1),(5,1)}}.

Reduce Phase: The output of the Map Phase is sent as an input to the Reduce Phase. Each reducer receives a set of values for the same keys (node labels). Therefore the reducer receives a set of values for the same node. The aggregation of all such values gives the out-degree of the corresponding node. Therefore each node is given as input to a parallel node where the reducer would give the output as the aggregated value of their corresponding out-degrees.

For the above example graph shown in Figure 3, each reducer receives all the pairs mentioned before with the same key. Since there are 5 different nodes, maximum of 5 different reducers can be invoked. Each reducer now aggregates the values and gives the following output: R1:(1,2), R2:(2,2), R3:(3,2), R4:(4,3) and R5(5,1), where R stands for a Reducer. The sequence of steps inside the reducer of the Initiation Stage is explained in Algorithm 5.

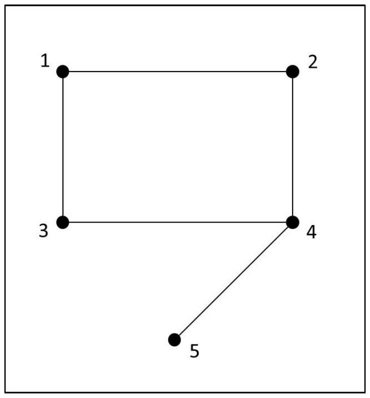

Figure 3. An example of a small information network represented in the form of an undirected and unweighted graph

Algorithm 4: Initiation Stage Mapper

```
Input:
        List of (key,value) pairs
        Key : Offset
        Value: Edge <i-j>
Output:
        List of (key,value) pairs
        Key : NodeID
        Value: 1
1: for all edges <i-j> do
2:      Write <i,1>
3:      Write <j,1>
4: end for
```

Algorithm 5: Initiation Stage Reducer: Compute the out-degree of n_0

```
Input:
        Key : NodeID
        Value: 1
Output:
        Key : NodeID
        Value: OutDegree
1: Initialize OutDegree = 0
2: for all <NodeID,one> do
3:      OutDegree = OutDegree+1
4: end for
5: Write <NodeID,OutDegree>
```

Driver program: After collecting the outputs from all the reducers, the Driver program would then identify the node with the largest out-degree. This out-degree will be used to compare the M-gain at consecutive stages. In the above example, if the initial node, n_0, selected is node with label

1 in Figure 3, then the out-degree of node 1 would be 1 and in-degree would be 0. This node is now part of the community D. Consecutive addition of vertices from the graph would change the out-degree and in-degree of D, and thus the overall M-measure of D.

3.2.2. Induction Stage

The *Induction Stage* adds the nodes to a community. This phase consists of multiple iteration with each iteration adding a node to the community. The parallel component in this phase is to identify the overall M-measure of the community after the addition of a distinct node in parallel. Therefore different systems would add different nodes from a list of potential nodes and the node that increases the overall M-measure by a maximum margin is included in the community. Once there is no node available that would increase the M-measure of the community, the phase is terminated and the community D is provided as the input. Each iteration of this phase consists of a Map and a Reduce Phase. The block diagram and flow of events are explained in Figure 4.

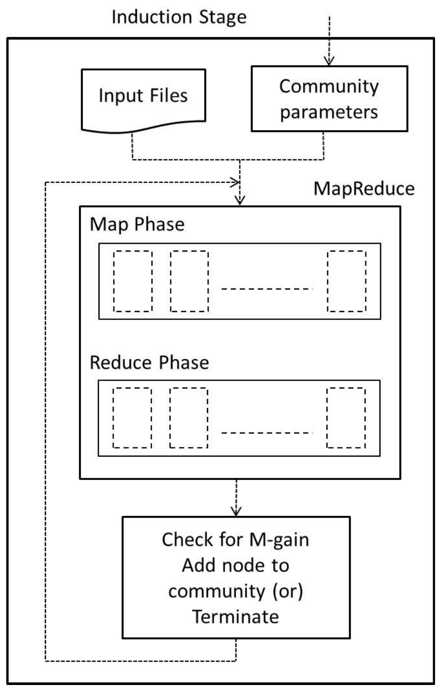

Figure 4. Induction Stage - Flow through different components in the Induction stage including iterative MapReduce Phase

Map Phase: All the mappers receive a data split from the original set of inputs. The input is in the form of a (key, value) pair, with key being the offset and the value being the edge from the input file. Apart from the data split for the map function, all the mappers receive the same input file containing the already existing nodes in the community. Each map function receives an edge with the two connecting nodes. Each node is identified as *belonging to a community* or *belonging outside to community*. If the edge contains both nodes belonging to a community, the map function does not generate any output. If only one of the nodes in the edge belongs to the existing community, an output value of 1 is written by the map function with a key corresponding to the node. If both the nodes in the current edge do not belong to the community, then two outputs are written with the value of -1.

For the same example shown in Figure 3 with the following edge set as input, {(1-2),(1-3),(2-4),(3-4),(4-5)} and assuming that the community D already consists of two nodes in the community {1,2}. With (1-2) as an edge to be processed, the map function does not produce any output. For edges {(1-3), (2-4)}, the output that is generated in the form of (<key>, <value>) is (<3>,<1 1>) and (<4>, <2 1>) respectively. The output contains 1 as part of the value since if the nodes (as keys) are added to the existing community, they would be added to the in-degree of the community. For edges such as (3-4) and (4-5), two outputs {(<3>,<4 -1>), (<4>, <3 -1>)}, {(<5>,<4 -1>), (<4>, <5 -1>)} are generated for each input respectively. The value of -1 indicates that the value would be added to the out-degree if a node is added to the communities.

Algorithm 6: Induction Stage Mapper

```
Input:
        List of (key,value) pairs
        Key : Offset
        Value: Edge <i-j>
        Community D
Output:
        Key : Potential NodeID
        Value : <Corresponding Edge:Flag>

1:  for all Input Edges <i-j> do
2:      if i in D and j not in D then
3:          Set Flag := 1
4:          Write <j, i:Flag>
5:      else if j in D and i not in D then
6:          Set Flag := 1
7:          Write <i, j:1>
8:      else if i not in D and j not in D then
9:          Set Flag := -1
10:         Write <i, j:Flag>
11:         Write <j, i:Flag>
12:     end if
13: end for
```

Reduce Phase: All the values corresponding to the same key are sent to a single reducer for further processing. All reducers receive a common file that contains the information about the current in-degree and out-degree used to compute the M-measure for the community. In a given reducer, the reduce function aggregates the number of 1's and -1's and adds to the in-degree and out-degree of the community respectively to compute the new M-measure, M'. If the in-degree of any node n_r is 0, then the M-measure is not calculated since n_r is not a neighbor of at least one of the nodes in community D. In the above example, it is evident from the outputs that Node 5 has an in-degree of 0 and an out-degree of 1 and thus Node 5 is eliminated from the list of potential nodes. If M' > M, the output is written with the node and the new value of M'.

At the Driver: After the completion of the reduce phase, the Driver identifies the maximum gain of the M-metric and adds the corresponding node to the community D. The process is repeated until there is no more edge that could increase the overall M-measure of the community.

Using the same approach, a set of local communities could also be identified by iteratively calling the Initiation and Induction Stage. Each set of Initiation and Induction Stage would identify a new local community and add it to the list of communities. A random node will be selected during the Initiation Stage among the set of nodes that do not belong to any community.

Algorithm 7: Induction Stage Reducer: Compute M gain of shell nodes

```
Input:
        List of (key,value) pairs
        Key : Potential NodeID
        Value : <Corresponding Edge:Flag>
        InDegree and OutDegree of Community D

Output:
        Key : Node $n_i$
        Value : M' (Overall M-measure after include of Node $n_i$)

1:  Initialize localInDegree := InDegree, localOutDegree := OutDegree
2: for all <nodeID, Flag> do
3:        if Flag = 1 then
4:                localInDegree := localInDegree + 1
5:        else if Flag = -1 then
6:                localOutDegree := localOutDegree+1;
7:        end if
8: end for
9: if indegree != 0 then
10:       Compute $M_{gain}$
11:       Emit <nodeID, $M_{gain}$ >
12:       end if
```

4. Experiment

Experiments were conducted to test the scalability and accuracy of the MapReduce version of Local Community Detection algorithm using the M-Metric.

4.1. Experimental Setup

Experiments to test the scalability were conducted on Amazon's Elastic Map Reduce (EMR) framework (V 3.6.0) with hadoop's V2.4 implementation of the MapReduce framework using Amazon EC2's instances. The master node running the driver program is c3.xlarge with 7.5G of main memory and 4 cores of vCPU. The worker nodes running the mapper and reducer are m1.large instances with 7.5G of main memory and 2 vCPUs per instance. The running time of the sequential algorithm are also run on the same c3.xlarge instance.

4.2. Datasets

Two different groups of datasets were used for the experiments. The first set included a group of 3 synthetic datasets generated using the LFR network generator [15]. Each dataset was a simulated network with known ground truth to compare the accuracy of the parallel version. The number of nodes and edges with the number of communities are shown in Table 1. The accuracies of the synthetic datasets with different network sizes are reported in Section 4.3.

The datasets that are used to report the improvement of execution time using our parallel version of the algorithm are large real datasets [16]. The real datasets consist of the Orkut Social Network with more than 3 million nodes and more than 100 million edges, and Friendster with about 65 million nodes and almost 2 billion edges. The exact size and characteristics of the real networks are given in Table 2.

Table 1. Synthetic Datasets.

Network	Nodes	Edges	Communities
Dataset 1	1,000	15,534	33
Dataset 2	1,200	20,290	39
Dataset 3	900	12,376	31

Table 2. Real Datasets.

Network	Nodes	Edges	Size of Data
Orkut	3,072,441	117,185,083	2.2GB
Friendster	65,608,366	1,806,067,135	32GB

4.3. Results and Analysis

The first set of experiments were conducted to compare the accuracy of the sequential local community detection algorithm with the MapReduce version to detect communities locally using the same M-metric. The accuracy is compared in terms of Adjusted Rand Index [17] for all the communities that have been identified in each dataset. The ARI is a common measure of the similarity between two data clusterings (or community mining results) that is adjusted for the chance grouping of elements, which is equivalent to the adjusted mutual information from information theory also used to compare clusterings. The validation indices comparing the sequential and parallel algorithms are illustrated in Figure 5. The reported results in Figure 5 are averages of 10 runs of the algorithm on each dataset for each algorithm. The accuracies are very good for all three datasets and for both algorithms. It is seen from the graph that the accuracy of the parallel version is very similar to that of the sequential algorithm with only marginal differences. There were a total of 34 communities found by each algorithm for all the runs with random initial nodes. The difference in the accuracy is due to the randomness in selecting a node to find a local community. The variations were minimal and indicated by the standard deviation as error bars on the graph.

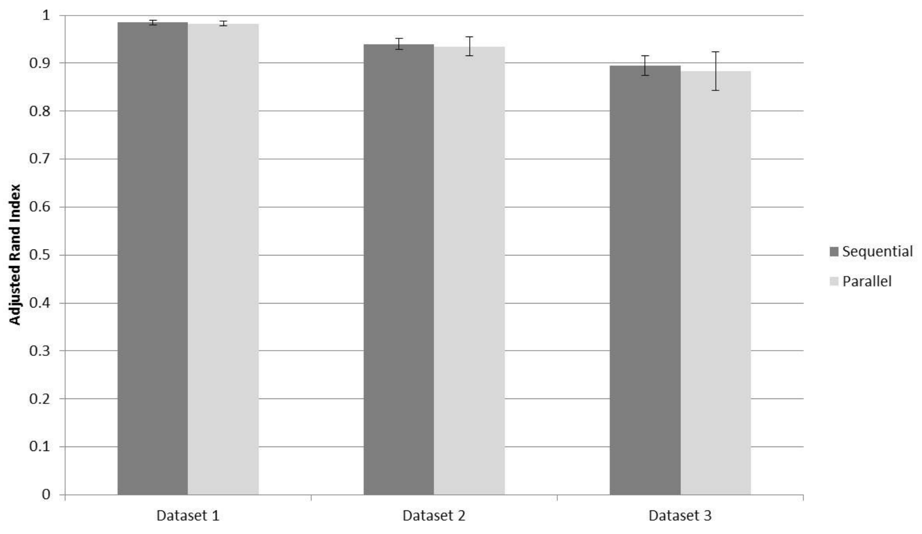

Figure 5. Comparison of Accuracy - Sequential vs Parallel version

The other set of experiments were conducted to study the scalability and the improvements in execution time of the parallel algorithm. The sequential algorithm finds a community after traversing through each of the edges to the neighbors of the current community to compute their in-degrees and out-degrees. This independent computation being sequential consumes significant time resource which has been eliminated in the parallel version. Also, for very large networks, it is not possible to run the community detection algorithm due to the constraints in memory which can be addressed using our parallel algorithm.

Figure 6 shows the improvements in execution time with the increasing number of available parallel systems. As the number of parallel systems increase, the parallel algorithm takes advantage of the available systems and computes the in-degree and out-degree in parallel. Increasing the number of parallel systems would not improve the results linearly due to the fact that the number of parallel systems that can be completely utilized by the MapReduce version of the algorithm depends upon the number of vertices in a graph for which the degree (in and out) measure is computed, i.e., the total number of reducers that can be called in parallel.

Similar experiments were conducted on the Friendster dataset shown in Table 2. Due of the tremendous size of the data, the sequential algorithm was not able to complete the process for lack of memory. Indeed the program crashed due to an *OutOfMemory Error*. Nevertheless, our parallel MapReduce-based algorithm was able to complete the task. However, at the time of writing, we were unable to run our algorithm multiple times to report an average. As expected, one run of our algorithm on the Friendster dataset with more than 65 million nodes and about 2 billion edges takes more than 5 hours.

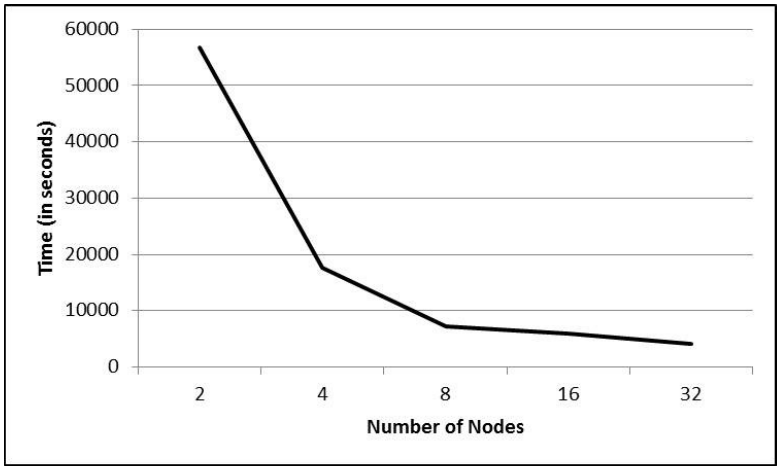

Figure 6. Improvements of execution time with increase in parallel system for the parallel algorithm on Orkut Dataset.

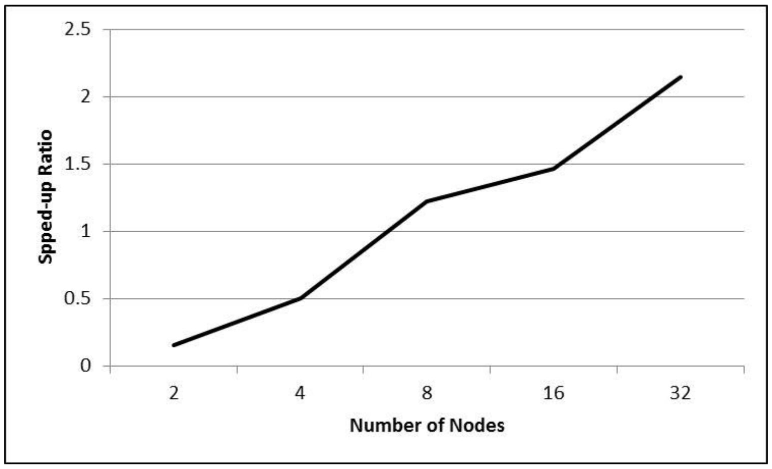

Figure 7. Improvement of parallel algorithm over the sequential one with increasing parallel systems on Orkut Dataset

5. CONCLUSION

To summarize, a parallel version of the local community detection algorithm using the M-metric, 3MA, was introduced and implemented using the MapReduce framework. For very large networks, the parallel version improves the time to extract local communities from the network and the same has been proved empirically with experiments on real very large datasets. The communities that are formed using both the sequential and parallel versions are also similar and were shown using experiments on generated synthetic datasets with known ground truth. One of the aspect that should be investigated in the future is the possibility to develop similar MapReduce strategies to identify local communities in an information network but using other more recent metrics such as the L-metric [4] and the T-metric [18], that are more robust measures to detect a local community and evaluate the relative quality of a community by considering the number of internal and external triads (3-node cliques) the community contains.

REFERENCES

[1] Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC Transactions on ECE, Vol. 10, No. 5, pp120-122.

[2] Gizem, Aksahya & Ayese, Ozcan (2009) Coomunications & Networks, Network Books, ABC Publishers.

[3] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai, (2011) "Adaptive algorithms for detecting community structure in dynamic social networks", 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, Shanghai, China, pp. 2282–2290.

[4] S. Moon, J.-G. Lee, and M. Kang, (2014) "Scalable community detection from networks by computing edge betweenness on mapreduce", International Conference on Big Data and Smart Computing , pp. 145–148.

[5] F. Luo, J. Z. Wang, and E. Promislow, (2006) "Exploring local community structures in large networks," in Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, Washington, DC, USA: IEEE Computer Society, pp. 233–239.

[6] J. Chen, O. Zaiane, and R. Goebel, (2009) "Detecting communities in large networks by iterative local expansion", International Conference on Computational Aspects of Social Networks, CASON '09, pp. 105–112.

[7] J. Chen, O. Zaïane, and R. Goebel, (2009) "Local community identification in social networks", International Conference on Advances in Social Network Analysis and Mining, IEEE ASONAM'09, pp. 237–242.

[8] G. Karypis, V. Kumar, and V. Kumar, (1998) "Multilevel k-way partitioning scheme for irregular graphs," Journal of Parallel and Distributed Computing, vol. 48, pp. 96–129.

[9] B. Long, X. Wu, Z. M. Zhang, and P. S. Yu, (2006) "Unsupervised learning on k-partite graphs," in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06. New York, NY, USA: ACM, pp. 317–326.

[10] B. Long, Z. M. Zhang, and P. S. Yu, (2007) "A probabilistic framework for relational clustering," in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07. New York, NY, USA: ACM, 2007, pp. 470–479.

[11] M. Girvan and M. E. Newman, (2002) "Community structure in social and biological networks," Proceedings of the National Academy of Sciences, vol. 99, no. 12, pp. 7821–7826.

[12] M. E. Newman, "Fast algorithm for detecting community structure in networks," Physical review E, vol. 69, no. 6, p. 066133, 2004.

[13] M. E. Newman and M. Girvan, (2004) "Finding and evaluating community structure in networks," Physical review E, vol. 69, no. 2, p. 026113.

[14] A. Clauset, (2005) "Finding local community structure in networks," Physical review E, vol. 72, no. 2, p. 026132.

[15] Y. Chen, C. Huang, and K. Zhai, (2009) "Scalable community detection algorithm with mapreduce," in Communication ACM, vol. 53, pp. 359–366.

[16] J. Shi, W. Xue, W. Wang, Y. Zhang, B. Yang, and J. Li, (2013) "Scalable community detection in massive social networks using mapreduce", IBM Journal of Research and Development, vol. 57, no. 3/4, pp. 12–1

[17] A. Lancichinetti and S. Fortunato,(2009) "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities", Physical Review E, vol. 80, no. 1, p. 016118, 2009.

[18] J. Yang and J. Leskovec, (2012) "Defining and evaluating network communities based on ground-truth," in Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics, New York, NY, USA: ACM, pp. 3:1–3:8.

[19] L. Hubert and P. Arabie, (1985) "Comparing partitions," Journal of classification, vol. 2, no. 1, pp. 193–218.

[20] J. Fagnan, O. Zaiane, and D. Barbosa, (2014) "Using triads to identify local community structure in social networks," in IEEE/ACM International Conference on Social Networks Analysis and Mining (ASONAM).