Abstract—In this information era, data from different sources (online activities) are in abundance. Social media are increasingly providing activities and data, relations and interactions (audio, video and texting) among social actors (people), due to increasing capabilities of mobile devices and the ease access to the Internet. More than a billion people are now involved in online social media, and analyzing these interactive structures is a huge data-analytic problem. The primary focus of this work is to develop a clustering algorithm for multi-characteristic and dynamic online social networks. This work uses a combination of multi-objective evolutionary algorithms, distributed file systems and nested hybrid-indexing techniques to cluster the multi-characteristic dynamic social networks. Empirical results demonstrate that this adaptive clustering of dynamic social interactions can also provide a reliable distributed framework for BIG data analysis.

Index Terms—Social Network, Clustering, Graph, Evolutionary, Genetic Algorithm, HDFS, Hadoop Distributed File System, Distributed Fuzzy Clustering.

1. INTRODUCTION

Social network data can be stored as graphs, which dynamically change with time either by expanding or shrinking. The topology of the graph also changes along with the relationship among nodes. Several algorithms were proposed for community clustering, but only a few of these deal with multi-characteristic and dynamic networks. In particular, most of the existing algorithms work for static and small networks. Very few algorithms can be extended to work on large and dynamic networks. In this work, we developed an evolutionary clustering framework using MapReduce programming paradigm. The framework runs over an HDFS (Hadoop distributed file system). The combination of evolutionary algorithms and HDFS allows us to reduce the computational cost and efficiently perform parameter-less clustering of big and dynamic social network data.

1.1 Data Clustering

Data clustering is the process where a set of elements is divided into a set of Groups/Classes contains these elements; each Group/Class contains data that shares similar characteristics or properties. Similarity measures that are used to cluster these data differ based on the nature of data to be clustered. These measures control the clustering process and the formation of the Classes/Groups; it can be the distance measure in some problems or connectivity in other problems, etc.

1.2 The Problem Statement

Social media and online social networks have become a massive data source and represent virtual relations among people. These social media and online social networks contain important information, which helps in studies such as social behavior, online marketing, and studies about web usage. Recently, these have attracted more attention among the research field and research groups.

Communities can be defined as a group of individuals who interact within a group more frequently than outside the group. Studies on these communities can not only contribute in the above-mentioned areas, but also in addressing security issues. Understanding how these groups are formed and how they change over time, by classifying nodes in a network based on some characteristics, can help in applying theories and techniques to improve data analytic field.

Graph or network clustering has been proven to be NP-complete problem [1], which means there is no known efficient way to find an optimal solution, also the time required to solve this problem increases relatively with the size of the dataset. However, social networks grow fast, and they are dynamic in nature, this means by the time the network is clustered, the network has already changed, and the newly formed network may be different than the recent clusters.

In addition to the above data issues, social
networks are multi-characteristic in nature. There are multiple ways online social communication can occur. For example, Facebook creates a node (a person/page…etc.) and can add another node as a friend when agreed to share information, which constitutes a connection. A node can also send a message to another node, and this is different type of connection, and it can be combined together to form links with values for each characteristic.

Combining different types of connections with multiple characteristics results in large and complex datasets, making it difficult for clustering algorithms to cluster in a reasonable time.

1.3 Defining Network Clusters

In this work, we referred to graph of vertices V and edges E as G = (V,E), as an undirected graph. Let number of vertices |V|=m, number of edges |E|=n and clustering C = \{C_i, C_2, C_3, ... C_j\} as a partition of V as disjoint sets. We call C, a clustering of G containing j clusters. The number of clusters j has a minimum of j=1, when C contains only one subset \(C_i = V\), and a maximum of j=m when every cluster \(C_i\) contains only one vertex. We identify the cluster \(C_i\) as a sub-graph of G. The graph \(G(C_i) := (C_i, E(C_i))\), where \(E(C_i) = \{(V,Wj) \in E.V,W \in C_j\}\.

Then \(E(C) = \bigcup_{k=1}^{j} E(C_k)\) is the set of intra-cluster edges and \(E(E(C))\) is the set of inter-cluster edges. The number of intra-cluster edges denoted by \(m(C)\) and \(m(C)\) is the number of inter-cluster edges.

As an input, a social network is represented as a set of graphs \(SNCG=\{G_1, G_2, G_3, ..., G_z\}\), and the set of graphs-clustering \(SGC=\{G_1, G_2, G_3, ..., G_z\}\) where each graph \(G_i\) has its own clustering \(C_{Gi}\) satisfying conditions mentioned for G and C respectively. Let Z is the number of characteristics in a given dataset. Now, graphs \([G_1, G_2, G_3, ..., G_z]\) have the same set of V but have a different set of E and each \(C_{Gi}\) is an objective to achieve.

The goal is to find SGCs using a multi-objective optimization and combine them into one clustering \(SNCG=\{SNCG_1, SNCG_2, SNCG_3, ..., SNCG_x\}\) where \(SNCG := \bigcup_{i=1}^{z} C_{Gi}\). The set of clustering for social network SNCG is not necessarily disjointed, but it is a union of sets where each set is a group of disjointed subsets.

A social network representation with all of its characteristics can lead to a dataset of a huge graph, however, we represented the social network as a set of graphs rather than one graph, each graph represents one characteristic The proposed algorithm takes the social network as a multi-characteristic dataset, then partitions it into set of graphs SGC, where each graph contains edges for only one characteristic. Then each graph \(G_i\) is clustered individually by an edge removal algorithm to produce disconnected graph represented by clustering \(C_{Gi}\), then by measuring the strength of these clusters. After clustering each graph \(G_i\), we combine elements of each clustering in SGC into one clustering SNCG, to produce an overlapped clustering where clusters \(SNCG_1\) to \(SNCG_X\) are not necessarily disjointed.

During the clustering process of each graph, an evolutionary algorithm has been used and Hadoop distributed file system (HDFS) has been used to provide improved performance and speed by partitioning the large datasets into smaller logical blocks.

1.4 Related work

Several algorithms have been applied for data clustering problems; we surveyed these approaches and classified them based on their scalability issues in handling large data. The list below shows different clustering and data mining techniques along with their advantages and drawbacks.

- In “basic concept of data mining, clustering and genetic algorithm” [24], Tsai-Yang Jea reviewed basic evolutionary algorithms, whose concepts have been analyzed with the following results:
  - Advantages: Fast results as a clustering algorithm, since it doesn’t search the whole space of solutions.
  - Drawbacks: Final clusters don’t show global optimization, the chromosomes represent the whole space each time, and need parameters as inputs, like a number of clusters to start. Also the search for node’s similarities itself is time consuming process, and it has to be done in C*N2 time, where N is the number of nodes and C the number of chromosomes. The amount of time taken to produce the result makes it inapplicable for huge datasets.
  - Petra Kudová developed (CGA) an evolutionary algorithm for clustering in his paper “Genetic algorithm clustering” [20], published from Academy of Sciences of the Czech Republic, ETID, on 2007. After a deep analysis the advantages and drawbacks for his research have been summarized as follows:
    - Advantages: faster than regular search approach, and looks for global optimization.
    - Drawbacks: Similar to Tsai-Yang Jea [4], the search for node’s similarities itself is a time consuming process, and it has to be done in C*N2 time, where N is number of nodes and C number of chromosomes. Also each chromosome copies the whole search space as a list, and that is C*S where S is the search space size (billions of nodes and connections in real life). That makes the execution time and space for this algorithm impossible for huge dataset processing.

There are many other related works (listed below) which demonstrate different advantages of clustering algorithms; nevertheless, these approaches have almost similar drawbacks as discussed in reference to the algorithms mentioned in above section. The major drawback with these approaches is they needed some parameters to be fed. For example, a number of clusters, size of clusters and/or number of generations are needed by
evolutionary approaches. It is suggested that these parameters should be found from the dataset and not given as an input parameters. Also these inputs aren’t available; hence, it changes the solutions and can result false solutions.

Some other critical drawback is that these approaches copy the search space many times “each solution encoded in a way that represent the whole network”, thus resulting in high demands for processing space, which is an impractical approach in real life. On the other hand, if the results are produced slowly, the network has already changed because of its dynamic behavior. Below is another list of papers that share the same disadvantages:

- Evolutionary Clustering and Analysis of Bibliographic Networks [15]
- Multi-objective Evolutionary Algorithms for Dynamic Social Network Clustering [13]
- A Multi-objective Hybrid Evolutionary Algorithm for Clustering in Social Networks [6]
- A framework for analysis of dynamic social networks. [23]
- An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. [21]
- Genetic algorithm and graph partitioning. [22]
- Multi-objective evolutionary clustering of web user sessions. [18]
- Dynamic algorithm for graph clustering using minimum cut tree. [15]
- Community detection in complex networks using genetic algorithm. [2]
- A new graph-based evolutionary approach to sequence clustering. [16]

None of the previous works show interest in a combination of evolutionary and distributed or parallel approach, thus resulting in clustering and search overhead. The approach in this study uses the Hadoop distributed file system and a combination of hybrid hashing and evolutionary algorithms; hence, resulting in a fast, robust, and practical solution according to the dataset size.

1.5 Large “Social Networks” Datasets Clustering

The major problem in traditional combine and test, for search algorithms (the sub problem of clustering), is that the search space is too large and for larger networks, it is impossible to apply. It was observed that the previous works, including evolutionary clustering algorithms, need to search for connections, and this process takes place several times for each node in the network, either for the clustering process or for the evaluation process. On the other hand, traversing the network at each node and looking for all neighbors in a huge dataset is an overhead itself.

For the above reasons, a new distributed evolutionary algorithm was developed for very fast clustering.

The algorithm takes the social network data as an input and then transforms it into multiple graphs, each graph clustered separately to reduce the clustering overhead for one large dataset, and all clusters combined together as final clusters for the social network, thus can lead to overlapped clusters based on the characteristic.

1.5.1 The Evolutionary Clustering

Since the large datasets clustering problem have a huge search space, and the clustering problem is proved to be an NP-Complete problem, as per Jiri Sima and Satu Elisa Schaeffer proved in their work “On the NP-Completeness of Some Graph Cluster Measures”. We chose evolutionary algorithms to find an approximation or a close to optimal solution and because of the dynamic nature of the social networks, it was decided to develop an evolutionary algorithm that clusters the network in a fast way and uses the metrics above as a fitness function and evaluation for solutions. Since social networks are full of noise in terms of data, the chromosome encoding developed as a list of weak and noisy edges to be removed, “edge removal and cut based algorithm”.

Most traditional evolutionary algorithms were developed in a way that the user would provide some parameters for the network, and the algorithms would process the data based on these parameters. In this study, it is believed that the user shouldn’t provide these parameters, but they should be extracted from the network itself. Hence an algorithm is developed in such a way that the network-edges list and its characteristics- has to be the only input, and noisy data was read from the user. During the execution of the algorithm, the framework receives the changes in the network and reflects on the algorithm inputs-the edges list- then the algorithm produces results and creates output solutions based on the most recent network inputs.

jMetal 4.3 is a powerful object-oriented Java-based framework aimed at multi-objective optimization by using metaheuristics. jMetal provides a rich set of classes which can be used as the building blocks of multi-objective techniques. As per Antonio J. Nebro, Juan J. Durillo “jMetal 4.3 User Manual” [18], jMetal is used to develop the evolutionary algorithm.

1.5.2 The Job Distribution and Parallelism

The clustering problem causes overhead, and there are a lot of searches during the process. To make the process faster and less memory demanding, a parallel-distributed evolutionary algorithm is developed. Such algorithms need a synchronization mechanism so the solutions can be produced. In this study, the algorithm is synchronized on a population level (discussed in architecture section): each population will move to the next one only after a complete evaluation.

Evolutionary algorithms make the adopted approach work faster and distributed evolutionary computing makes it possible to process even faster;
resulting in less clustering overhead and increasing of practicality of clustering huge dynamic datasets.

Hadoop distributed file system (HDFS) provides a robust platform for our algorithm where a dataset is distributed among multiple computers and each computer works on the data it has, based on the job it received from the master computer.

2. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

2.1 HDFS Architecture

Hadoop distributed file system (HDFS) is an open source file system that can combine multiple computers and show them as one system. It also allows users to query and process large datasets in a short time. Additionally the capabilities of the system increase with the number of computers added to HDFS cluster. It can work with unstructured data as well as a collection of structured data. The main idea behind this file system is to split large datasets into smaller ones and spread them over the HDFS cluster with some redundancy mechanism to provide a strong reliability in the cluster.

HDFS cluster provides a collection of services. The primary service among them is MapReduce, where any process (also called job) can be submitted to the master computer (called the master node), and the master computer, in turn, splits the job into tasks for each computer in the HDFS cluster (called data nodes). Each computer performs mapping and reducing for the task assigned to it on the data it has. After mapping and reducing, the result is returned to the master node, and the master node returns result to the HDFS client as files.

2.2 MapReduce

MapReduce is a programming paradigm that can run on HDFS. MapReduce can be used to query from serial files distributed on HDFS cluster. This process made it easier to query huge datasets (petabytes of data) in a faster way than normal indexed databases.

After the file is uploaded to HDFS, it becomes ready to use. MapReduce operations can occur based on RPC (remote procedure calls) for the user to the Job Tracker. The user defines a MapReduce functions and passes them to the Job Tracker. Then the Job Tracker spreads the map function as tasks to Task Tracker. Each Task Tracker works only on the data blocks it has on its DataNode. The map function reads the data and maps it to pair off <Key, value> and passes that to the reducer function. Before the reducer function performs the reduce operation, a shuffle operation exchanges the pairs between TaskTracker, where each reducer takes one Key or more to work on. The reducer then starts reducing the collection of <key, value> pairs that came from the map function into a new single pair of <NewKey, NewValue>. The reduce operation is user defined which writes these results into files and saves it in the HDFS.

3. EVO-DISTRIBUTED CLUSTERING USING HDFS

The proposed framework consists primarily of two main components, the evolutionary component, and the distributed file system. The two components overlap to provide one framework that takes a dynamically changing dataset as an input and splits it into blocks over the distributed file system. The evolutionary part creates chromosomes and is responsible for extracting the parameters, create clustering and evaluating jobs, sending jobs to HDFS cluster, getting the result back to generate new solutions and create new jobs again.

Figure 1 illustrates the proposed framework architecture on a very high level. The components’ purposes are listed after the figure.

![Figure 1: The Proposed framework](image)

The framework composed of two main components:

1. Client Module:
   a. Inputs Preprocessor: processes the social network dataset and transforms it into a multi-dimensional dataset ready for uploading into HDFS.
   b. Output Module: gets evaluation results and sends it to evolutionary module and saves the most recent clusters as clustering results.
   c. Evolutionary Module: processes and executes the evolutionary algorithm and its operator to find best clustering solutions, it also creates clustering and evaluating jobs for HDFS.

2. HDFS module:
   a. Inputs Distribution Module: receives the processed dataset and distributes it over HDFS DataNodes.
   b. Job Manager: receives jobs from evolutionary module and transform it into tasks of Map and Reduce.
   c. Tasks Distribution module: distribute Map and Reduce tasks to TaskTrackers.
d. Results Warehouse Module: saves clustering results of solutions and its evaluations to be read by output manager module on client module.

Every distributed algorithm needs synchronization. The approach adopted in this study, especially the evolutionary algorithm component, needs synchronization because no solutions can be generated if previous solutions (parents) aren’t evaluated. To reduce the overhead in job generation, submission, and on HDFS calls, the synchronization is generalized to the simplest level.

The simplest level to which synchronization can be generalized is new population level. The approach can be preceded with the population itself but cannot be moved before evaluating solutions. The algorithm class in jMetal is modified to evaluate the population at once rather than solve on each level. Each group of solutions is sent at once as a clustering and evaluating job to HDFS. The next generation of solutions can only be created after the previous one is already clustered and evaluated.

The pseudo code of the algorithm after the file is uploaded in the HDFS and the changes are continuously being uploaded in parallel with the execution of the algorithm and can be further described as:

1. The problem class generates the inputs for the evolutionary algorithm.
2. The User program runs the algorithm by issuing the command execute for the modified algorithm class, and sends the problem object with its values to the algorithm object by the execute method.
3. The algorithm class on HDFS client generates the first population chromosomes, and keeps it without fitness values ready for evaluation.
4. The algorithm class running on HDFS client contains the unevauled population into a job along with number of dataset characteristics, and sends the job as a MapReduce job to the JobTracker on the NameNode.
5. The JobTracker distributes the job to TaskTrackers as tasks.
6. A Map and Reduce operation carried out by TaskTrackers writes the solutions as groups with its fitness in results files into HDFS.
7. JobTracker triggers the HDFS client running the user program that the job is done and the solutions with evaluations are ready along with group description for each solution.
8. The HDFS client pulls the results from HDFS results files and writes the groups of the best solutions into network file as the most recent clustering solution, on the other hand the algorithm takes only the fitnesses along with the solutions as an evaluated population. Then, ready to do GA operations like crossover and mutation, it generates a new unevaluated population.

Step 4 is repeated while the program is still under execution.

While the system is running, any changes to the dataset is immediately uploaded and merged with the input dataset on the HDFS. In parallel to the running algorithm, the changes are immediately reflected in new solutions.

3.1 Evo-Distributed Solution Space

The primary idea of the encoding scheme for the chromosome (called solution) is an array of integers that represent the noise edges in the network. We want to find and remove them, to create a network of distinct groups without any noisy edges and then find how strong these groups are as an evaluation for each solution. Each TaskTracker works only on the parts of the chromosome that it has in its block and marks them as removed edges. Each integer is an ID to an edge in the dataset, these IDs are created uniquely for each edge before uploading into HDFS.

3.2 Evo-Distributed Objective Functions

The algorithm is configured to be parameter-less, while previous work required the user to enter parameters such as number of clusters, clusters size gap, clusters modularity etc. It is believed that these inputs should be derived from the dataset, to make solutions realistic. So these parameters are made as objective functions, and they are added to the problem class in jMetal framework. The main objective function is composed of an equation, which contains a number of groups, a number of noise edges removed, and the values of each characteristic of the edges itself. Another objective has been added to represent the groups’ strengths and uses these values as multiple fitnesses to evaluate the solution. The formula below shows the main objective:

\[ f_c = \left( \sum_{groups} \sum_{Vn} \frac{Vn}{Gn} \right) - Er \]

Where:

- \( Fc \): objective for characteristic C.
- \( N \): edge N.
- \( Vn \): value of characteristic C on edge N.
- \( Gn \): the group size.
- \( Er \): number of edges removed.

N number of objective functions were created based on number of characteristics the network have, and each one of them reflects a different Fc and results in a different Fc value. Each one of these values is considered to have the objective to maximize in the problem class in addition to the modularity objective. This formula is developed to remove noisy edges. If any edge other than noise is removed, it will result in a lower fitness as a penalty, which lowers the number of edges removed and keep the network in the same topology, it also prevents groups of one node from being created.
3.3 Tasks at TaskTracker Level

Tasks on the TaskTracker’s level receives a copy of the solutions list (population) and then maps the data read from data blocks into pairs of <Key, Value>, by comparing the data with the solution list received. Each TaskTracker works only on the parts of the solution contained in its data blocks. We call these parts “active parts” and the rest parts of the solutions are called “inactive parts”. During the reduce step, values for each solution are collected from all mappers and each solution is made fully active.

The green dashed line is the active part of the solution, and its data is available in the data block on DataNode where the TaskTracker runs. The red dashed line indicates that this data is available in another data block on another DataNode. The reduce step combines all available solutions for one or more Keys, and the main Keys become active because of the data shuffle process that collects all the data for each solution. After the reduce step is carried out, the result for each solution (NewKey) is a data structure we developed. Its content is the list of solutions with its distinct groups and final fitness ready to be written on HDFS, so that HDFS can read it and proceed to the next generation in the algorithm.

4. Synthetic MapReduce Job Illustration

In this section, a MapReduce job is illustrated on the proposed framework to explain exactly what is happening in each step of the Job on TaskTrackers.

After uploading the dataset to HDFS cluster, it gets divided into data blocks each on DataNode, multiple blocks can be on the same DataNode. For illustration purposes, the file is divided into three data blocks, each on separate DataNode as shown in figure 2.

![Figure 2: (A) File before uploading to HDFS, (B) Data blocks after uploading to HDFS](image)

Each TaskTracker receives a copy of the Keys-solutions- to the mapper as mentioned in section 4.4, and maps the data into a <Key, Value> pair for the active part of the solutions. These <Key, Value> pairs are written into intermediate files using the custom write-ables we have specially designed. Figure 3 shows the map task on a single TaskTracker on one DataNode.

![Figure 3: map operation on single DataNode](image)

Figure 3 shows a data block, which contains edges with IDs {1, 2, 3} and {4}, and the Keys list, which is input for the mapper and consists of three solutions. The first one is responsible for removing noise edges 3 and 6, here edge 3 is considered an active part because the data blocks on this DataNode (only one block available) contains information about it; on the other hand, 6 is inactive since no information about it is in this DataNode. The mapper maps these values after removing the edge 3 for the first solution and writes the values shown in figure 13 as an array list. In figure 14, nodes {1, 2, 3} and {4, 5} are grouped after their fitness is calculated. Other nodes will be combined from other mappers in the reduce phase, and groups can be merged together when there is connections and fitness is recalculated. The same process is continued for solution 2 and 3. Their keys and values list from this mapper is then sent to the reducer.

The reducers shuffle the files so that each Key gets assigned to one reducer along with collection of values as an array, making the whole key active at this stage. The reducer looks for connections between groups, combines groups where there is connection into a single group, then combines multiple array elements into one element and recalculates the fitness by combining subgroups’ fitness values in the same formula. Figure 4 (A) shows the result received by a single reducer for the first solution, and (B) shows the solution after reducing and merging groups that have connections.

![Figure 4: (A) Values list for first solution from mapper collected to reducer. (B) Groups merged after reduce process](image)

In figure 4 (A) each color should be reduced into one group. The reducer merges these groups using the hybrid HashMap we developed.
After all reducers finish their assigned tasks, all keys are finalized with values and written into HDFS ready to be downloaded to the HDFS client for most recent results files. These values are for the algorithm to generate the next generation (population cycle).

5. EXPERIMENTS AND ANALYSIS OF RESULTS

5.1 Large Scale Real World Dataset

A multi-dimensional dataset available online from Youtube servers uses an open source API called Youtube API. The dataset contains 15,088 nodes and 5,574,249 edges; it also contains the following characteristics:

- Number of shared subscribers between two users.
- Number of shared favorite videos
- Number of shared friends between two users excluding the original nodes.
- Number of shared subscriptions between two users.

These datasets were merged together in one dataset and uploaded into HDFS of 4 nodes, three DataNode and one NameNode, and then the following experiments were performed. The first experiment consisted of 100 edges, population size 100 chromosomes. Each generation execution time was around 6000 ms. After ~50 generations the solutions started to form steady groups and 17 groups were found. During the run of the algorithm, the dataset was modified. Five arbitrary groups were added (totally unconnected to any of the previous groups), and the results were immediately reflected. After 3 groups were joined together with some noisy edges, the algorithm took around 10 generations to find those edges and to separate the groups again and go back to steady results, which was expected. The same experiment was done with larger datasets, 200, 400, 800, 1600, 3000 and 10000 edges; table 1 shows the results of these experiments.

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Average Generation execution time (ms)</th>
<th>Number of groups</th>
<th>Number of generations for steady results</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>~6000</td>
<td>17</td>
<td>~50</td>
</tr>
<tr>
<td>200</td>
<td>~9000</td>
<td>26</td>
<td>~130</td>
</tr>
<tr>
<td>400</td>
<td>~12000</td>
<td>50</td>
<td>~280</td>
</tr>
<tr>
<td>800</td>
<td>~24000</td>
<td>90</td>
<td>~740</td>
</tr>
<tr>
<td>1600</td>
<td>~50000</td>
<td>236</td>
<td>~2000</td>
</tr>
<tr>
<td>3000</td>
<td>~110000</td>
<td>479</td>
<td>~3200</td>
</tr>
<tr>
<td>10000</td>
<td>~270000</td>
<td>4932</td>
<td>~7100</td>
</tr>
</tbody>
</table>

5.2 Real World Dataset Experiments Analysis

As results are shown in table 2, there was almost a polynomial relation between the dataset size and the generation execution time. The time difference was caused by the dataset size and the communication latency. More time was needed for shuffling data between the extra cluster components which increased the clustering algorithm time.

Graph 1 illustrates the average generation running time vs. dataset size on the same HDFS cluster.

Graph 1: Average generation running time Vs. dataset size

The small curve at the end of the graph is caused by the difference between the last two datasets in size, and the light curve at the left side of the graph is caused by the communication latency at shuffle step between the maps and reduces. The algorithm doesn’t affect the number of groups found. The numbers of groups are totally related to the dataset.

Another positive impact of the approach is that it extracts parameters like the optimal number of groups and group sizes, considering them as objectives where previous approaches consider these values as input parameters, and gives our approach an advantage of a lower number of runs to get the correct inputs, which differ from dataset to another.

A further analysis of results files produced by the framework showed that after making changes in the dataset, the changes immediately reflected on the dataset of the changes do not result in adding groups, or splitting groups. However, if the changes add groups or split current groups into new groups it takes very little time to cluster new changes. Groups that are not affected do not need to be re-clustered and that is because of the evolutionary part of the framework, which keeps a copy of the best solutions and passes it on to the next generation.

5.3 HDFS Experiments

The HDFS components are tested on the same YouTube dataset of 10000 vertices. The number of DataNodes involved in the HDFS cluster are changed and then tested on a single node cluster, 2 Nodes HDFS cluster, 3nodes HDFS cluster and 4 nodes HDFS cluster. Table 2 illustrates the results on multiple clusters for 10000-edge dataset and 100-solution population size.

<table>
<thead>
<tr>
<th>HDFS cluster size</th>
<th>Average Generation running time (ms)</th>
<th>Number of generations for steady results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand alone one node</td>
<td>~320000</td>
<td>~7100</td>
</tr>
<tr>
<td>Two nodes HDFS cluster</td>
<td>~270000</td>
<td>~7110</td>
</tr>
<tr>
<td>Three nodes HDFS cluster</td>
<td>~180000</td>
<td>~7120</td>
</tr>
<tr>
<td>Four nodes HDFS cluster</td>
<td>~130000</td>
<td>~7110</td>
</tr>
</tbody>
</table>
Table 2 shows that the HDFS cluster size had negligible effect on the number of generations needed for steady results to start being produced, thus only the size of HDFS cluster affected the running time and had no effect on the results of clustering the dataset. Graph 2 illustrates the relationship between the sizes of the HDFS cluster vs. the average running time for each generation.

The average running time does not decrease in polynomial form since increasing the number of nodes in HDFS cluster reduces clustering work. Conversely, the communication and shuffling latency time increases and communication between nodes does take more time.

### 5.4 Comparative Results

For comparison between average running times on different sizes of HDFS clusters and different dataset sizes, graph 3 illustrates the performance of HDFS cluster and its effect on clustering time. Graph 3 also explains the effect of the HDFS size on clustering performance over time variable. Results on one node HDFS cluster illustrate the execution of the evolutionary algorithm without distribution.

By comparing results on graph 3, the results shows that HDFS cluster size have a big effect on the running time, and this effect decreases with the dataset size, on a very small dataset the HDFS size starts to lose it affect, since communication time between HDFS components and the shuffling latency takes more time than clustering on one node HDFS cluster. The almost steady change in average running time for 100,200 and 400 datasets size clearly proves the analysis above.

Graph 4 illustrates the time performance to cluster 3000-edge network, to compare the difference of executing evolutionary clustering algorithm without distribution, with executing the algorithm on HDFS cluster of 4 nodes.

### 6. Concluding Remarks and Future Work

Clustering social network and huge datasets is an NP-complete problem, optimization techniques proved efficient in solving such problems in the past, however combining such algorithms with the new distributed systems, lead to noticeable improvement. Also Defining multi-Characteristics social network dataset as a collection of multiple layers graphs, where each have the same set of nodes but different set of edges based on the characteristics, improves the computation and makes less overhead in computation aspects. On the other hand, considering the social network graph as a single layer represents all links and characteristics, leads to more complex graph with multiple links between same end nodes thus resulting in higher complexity for the same algorithm.

Distributed systems do not affect the solutions as much as the primary algorithm does; however, it has a big influence on the performance of the algorithm used, speeds up the process, and reduce work load and memory usage. Distributed computing opens new directions for algorithms to be expanded, and distributed file systems allow computing power to be able to work on a larger scale.

### REFERENCES


[6]. Babak Amiri, Liaquat Hossain & John Crawford, Multiobjective Hybrid Evolutionary Algorithm for Clustering in Social Networks,


[13]. Keehyung Kim, RI (Bob) McKay& Byung-Ro Moon, Multiobjective Evolutionary Algorithms for Dynamic Social Network Clustering


[15]. Manish Gupta, Charu C. Aggarwal, Jiawei Han & Yizhou Sun, Evolutionary Clustering and Analysis of Bibliographic Networks,


[18]. Pasi Franti & Olli Virmajoki, POLYNOMIAL TIME CLUSTERING ALGORITHMS DERIVED FROM BRANCH-AND-BOUND TECHNIQUE, University of Joensuu, Department of Computer Science, P.O. Box 111, FIN-80101 Joensuu, FINLAND


[24]. Tsai-Yang Jea, Basic concepts of Data Mining, Clustering and Genetic Algorithms, Department of Computer Science and Engineering, SUNY at Buffalo.


