Research Article

International Journal of Advanced Computer Research, Vol 10(48)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.2020.1048023

Implementation and performance analysis of dynamic partitioning of graphs in Apache Spark

Geetha J[*], Jayalakshmi D S and Harshit N G
Department of Computer Science and Engineering, MSRIT, Bangalore, India

Received: 11-February-2020; Revised: 10-May-2020; Accepted: 20-May-2020

Abstract
In recent years data is growing continuously at an exponential rate. Even in its processed form, large data is difficult to understand and analyze. One of the best approaches to handle large data is to represent it in the graphic format. These graphs can be used for data analysis, processing and decision making. Because of the volume of data is huge, the graphs generated by them are also huge making it difficult to process. Thus, the modular approach of studying them by partitioning them is much more effective. There are several inbuilt partitioning techniques available in Apache Spark which can be extended to graph partitioning according to the user needs. Graph partitioning is an active area of research with considerable activity with an aim towards increasing the accuracy and speed of the algorithms. This research work aims to build a high- performance graph partitioning technique for Apache Spark which makes it faster, scalable and efficient. In this paper, a custom algorithm that can divide the graph into an optimal number of partitions dynamically in Apache Spark is proposed. A novel method to calculate the distance between the nodes and similarity indexes to partition the data is introduced. The optimal number of partitions is decided to use similarity indexes, which are calculated using the concept of Laplacian matrix and Eigenvalues. The proposed algorithm is implemented and its performance is compared to the existing algorithms in Apache Spark. The results indicate that the algorithm partitions graphs which have a huge number of vertices with considerable efficiency and computing cost.

Keywords
Partition, Graphs, RDD, Clustering, Spark.

1.Introduction
In recent years data collection, management and usage have grown to an extremely large extent. Almost all the major fields like information technology, medical science, transport, retail are dependent on data analysis for growth. All necessary technologies needed for data gathering are getting developed exponentially. All the devices from high-end aeroplanes to household equipment are becoming smarter and collecting huge data every second. Now the biggest challenge is to analyze this huge data and retrieving useful information which will help in the major decision-making process.

One of the most discussed problems related to the processing of graph data is partitioned. Balanced partitioning of the graph is a popularly known NP-complete problem that has a large set of applications.

One of these applications is the solution to a major problem in cloud infrastructure i.e., optimal and efficient storage of large sets of data which are structured as a graph. At the root level, graph partitioning is nothing but distributing all the nodes in a partition in such a way that maximum of them are adjacent (with a maximum number of shared edges). Hence the definition of graph partitioning is – dividing the graph into a well-defined number of partitions, in such a way that excludes partitions have a very minimal number of edges. One of its variations is the uniform or balanced graph partitioning, where it is also a priority to place an equal number of nodes in each partitioned component. A proper partitioning can be used to reduce the cost of communication, load balancing or to recognize densely formed clusters [1, 2].

Apache Spark is the trending technology, which has gained immense popularity due to its high-performance computing in case of big data. In Spark, data is stored in persistent data structures called Resilient Distributed Datasets (RDDs). These RDDs

[*]Author for correspondence

can be partitioned and stored in different computational nodes of the system. The RDDs in these nodes are processed in parallel and later the results are aggregated. This partitioning of RDDs increases the performance and speed of Spark to a great extent. There are several inbuilt data partitioning techniques already available in Spark. These methods are used for data partitioning, the same methods can be extended to graph partitioning as well. But the problem is that these methods give good accuracy with only a certain kind and a certain amount of the data. A small variation in the amount of data may lead to a large variation in the accuracy of these algorithms. Also, these techniques divide the data statically into a predefined number of partitions. However, if the data are dynamically changing, then these algorithms fail to adapt [3, 4].

In this project along with the implementation of existing methods, a new custom dynamic partitioning method is developed for the Apache Spark. This dynamic technique exploits the relationship between the nodes to decide the partition. Closely related nodes are placed together. Nodes set with very little interaction (number of edges) are separated into different partitions. These partitions are placed in different RDDs. These individual RDDs are processed concurrently in different nodes. This parallel data processing increases the efficiency of Spark leading to its high-performance computing [5, 6].

1.1Problem statement

In any of the distributed processing systems, partitioning determines the degree of parallelism and impacts the performance of distributed applications to a large extent. The amount of data handled by each node should be balanced. Most of the cases, data will be divided blindly without any proper logic. This further disturbs the load-balance and performance of the application. Along with this, partitioning should have the capacity to group closely related nodes. Also, the partitioning technique should be dynamic to intelligently decide the number of partitions in the given data [7].

Existing algorithms in Apache Spark provide good performance to only a certain amount of data. However, this scale is not uniform as the data grows in size. Hence there is a need to develop a custom dynamic partitioning method for the Apache Spark which is consistent, fault-tolerant and scalable. The custom technique should also give stable accuracy and speed compared to existing partitioning methods. Partitions generated by this algorithm are stored as RDDs which further processed at different nodes of the system in parallel. This will increase the efficiency and speed of data processing in Apache Spark.

1.2Objectives

The objectives are as follows:

- Implement the existing partitioning methods in Apache Spark by extending it to graph data and analyze their performance.
- Implement the custom dynamic partitioning method for Apache Spark to overcome the problems in existing methods.
- Custom partitioning should have the capability to work on a huge amount of data with consistent accuracy and speed.

2.Literature survey

Balanced partitioning of the graph is a popularly known NP-complete problem that has a large set of applications. One of these applications is the solution to a major problem in cloud infrastructure i.e., optimal and efficient storage of large sets of data which are structured as a graph. There are several ways to partition the data (graphs in this case). Commonly used are static partitioning methods. However, if the data is continually changing/updating, then these algorithms fail to adapt. Extensive research has gone on to dynamically partition the data into the required number of partitions. Dynamic techniques exploit the relationship between the nodes to decide the partition [7]. Closely related nodes are placed together. Node sets with very little interaction (number of edges) are separated into different partitions. There are several existing graph partitioning techniques. Some of these techniques are explained in the following sections.

2.1BFS Partitioning

Breadth first search (BFS) is a commonly known method which can also be used in graph partitioning. BFS algorithm traverses the graph breadth-wise (level-by-level) and marks each vertex with the level in which it was visited. After the complete traversal of the input graph, the set of vertices of the graph is divided into two partitions V_1 and V_2 based on a predetermined threshold L. Then all the vertices falling under level which is less than or equal to L are categorized in the set V_1. All the remaining vertices which are having a level greater than L are placed in the set V_2. L is chosen in such a way that $|V_1|$ is always close to $|V_2|$. If there is a need to balance the graph, then $|V_1|$ should be equal to $|V2_{12}|$. *Figure 1* shows the working of BFS partitioning. First all the nodes will be red representing unpartitioned graph. After the first iteration of BFS, nodes falling in the lower level of threshold are grouped in V_1 represented by green color, remaining under V_2 as red nodes. In the third step partitions are balanced by placing few more nodes in V_1 which are comparatively close to lower threshold [8].

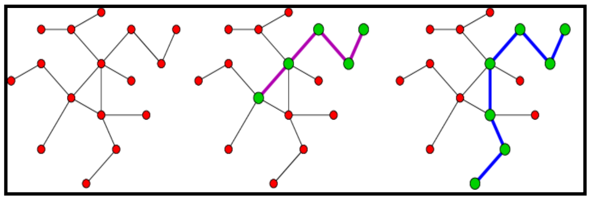
Figure 1 BFS partitioning

2.2Kernighan-Lin algorithm

The Kernighan-Lin algorithm (also popularly known as KL algorithm) is one of the well-known graph partitioning algorithms which is takes a heuristic approach. Consider simple graph represented as *G = (V, E, edge-weight function c).* In the simplest setting, the KL algorithm takes edge-weight function *c* of all edges and creates initial bi-partition (V_1, V_2) of the set *V*. Further, it produces a new partition ($V_1^{'}$, $V_2^{'}$) in such a way that | $V_1^{'}$ | = | $V_2^{'}$| = n by rearranging the vertices in sets. This re-arranged partition will be such that the total cost of the obtained partition is lower than the cost of the original partition. In this algorithm cost function is the cut count which is number of the nodes from which input node has to disconnected to move into new partition. *Figure 2* displays the working of the Kernighan-Lin algorithm. First graph is the initial bipartition with nodes D, E, F and C one set. After few iteration nodes are rearranged and final result has B, C, F and H in one partition and remaining in others [9].

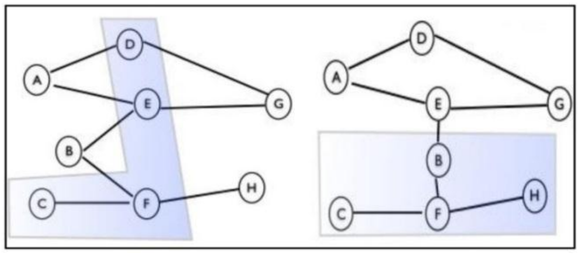

Figure 2 Kernighan-lin graph partitioning

2.3METIS

One of the popular software packages for irregular graph partitioning is METIS. METIS is also used for partitioning large meshes and also in computing fill-reducing orderings of sparse matrices. The algorithms in this package are based on multi-level graphs partitioning. Graph partitioning algorithms which are discussed so far computes partition of a graph by carrying out operations directly on the original graph. These kinds of algorithms are very slow and produce partitions which are of poor quality. On the other hand, multi-level graph partitioning algorithms follow a completely different approach. These algorithms; first reduce the size of the graph by collapsing vertices and edges, thus obtaining smaller graphs. Then algorithm reduce the size and simplify it to construct a partition for the actual large graph. METIS uses the divide and conquer approach to successively reduce the size of the graph. METIS utilizes in-built algorithms that make it easier to find a high-quality partition. During refinement, METIS focuses mainly on the partitions of the graph that is really close to the boundary. These well-tuned algorithms help METIS package to quickly produce high quality partitions for large graphs.

METIS supplies two programs PMetis and KMetis for partitioning the graph which is complex and large.
1. PMetis - based on multilevel recursive bisection
2. KMetis - based on multilevel *k*-way partitioning.

Both of these programs are able to produce partitions of high quality. However, based on the application, one programmer might prefer one over the other. Generally, KMetis is preferred when it is necessary to partition graphs into more than 8 partitions. In such cases, KMetis is notably faster than PMetis. On the other hand, PMetis is preferred for partitioning a graph into a smaller number of partitions [10].

2.4JA-BE-JA algorithm

JA-BE-JA is a distributed algorithmic solution for reconciling the balanced k-way problem. In JA-BE-JA algorithm each node of the graph is practically a virtual processing unit, containing the information about its neighborhood. Each node of the graph acquires knowledge about the group of nodes by local interaction. In the beginning, each node selects a random partition. Gradually nodes swap their partitions with each other to increase the size of the group which they belong to. This algorithm aims at dealing with large distributed graphs. It makes good use of principle of locality. In some cases, it outperforms the results achieved by METIS. Given a graph which is colored, the idea behind this algorithm is to drive the system into a lower energy state by applying the local heuristics. All the nodes in execute this local heuristic in parallel. Each node tries to

swap the color with the neighbour which is having the most dominant color, by the following strategy:
1. Select another node from its neighbours or from a random sample
2. Consider the utility of the color swapping. The two nodes swap their colors if it decreases the energy else, they keep their colors. Thus, the distribution of colors is maintained during the whole process as nodes just exchange colors. Therefore, if the color uniformly assigned at the start, then final result at the end is expected to have balanced partitions [11].

3. Proposed methodology

The various graph partitioning methods like BFS partitioning, Kernighan-Lin Algorithm, Balanced k-way partitioning, METIS, and the JA-BE-JA algorithms provide the required number of partitions and hence play a major role in increasing the parallelism. Finally, based on the study of these existing algorithms, a new custom partitioning method is developed.

3.1Deciding number of partitions on graph RDDs

The data stored in excel files need to be converted to graph format. First, this data is to be converted to Spark RDDs. The Jupyter notebook has an in-built Spark context. RDDs can be created using Parse function available in python. This RDDs have to be mapped and remapped to form well-formed RDDs. Data is split using split () and map () functions. Once RDDs are well formed with nodes and edges, Graph RDDs needs to be generated. The layout is specified using draw_networkx_nodes function. This function has parameters like network graph, node list, node color, node size, etc. Based on the user specification 2D graph will be generated. This graph acts as an initial, un-partitioned, input graph showing all nodes in a single color.

The graph RDDs generated need to be partitioned. Deciding the optimal number of partitions is the core functionality of the system. Earlier data-partitioning methods statically get the value from the user and divide accordingly. This method is pretty straight forward in the case of small datasets. However, in the case of data set having the range in millions of nodes, static methods are of not much use. The function described here makes use of mathematical concepts, statistical analysis to identify the optimal number of groups and display the same.

3.1.1Finding the laplacian matrix

Laplacian matrix which is also known as the admittance matrix is the graph which represented in a matrix format. This is one of the most useful properties of the graph. It provides functionalities like finding the number of spanning trees, obtaining central measure, etc. It is also used in several machine learning applications. However, the main use of Laplacian matrix is to identify the Eigen value and Eigen vectors.

The following is the mathematical formula for finding the Laplacian matrix.

$$L = D - A \tag{1}$$

Where A is the adjacency matrix and D is the degree matrix.

3.1.2Eigen value and eigen vector

Eigen value and Eigen vectors are the most used concepts when it comes to graph theory. They signify the most essential properties of the graph. The large values yielded by calculation of Eigen values signifies which set of nodes are strongly connected. Based on these similarity indexes, the number of partitions is decided.

Based on the Eigen values a new Spark graph RDD is created (RG_mbd) which is used as an input graph to the algorithms. These graph RDDs are useful because of their fault tolerance nature. In the case of the crash, these graph RDDs can be easily recreated using linages. This improves the reliability of the code.

Algorithm: Dynamically deciding number of partitions

```
1.Create RDDs using the dataset stored in excel
    format.
2.Map the RDDs accordingly by split and lambda
    functions.
3.Form node and edge list.
4.Create graph RDDs using node and edge list.
5.Display the un-partitioned graph.
6.Convert graph RDDs to matrix format.
7.Find the Laplacian matrix for input graph RDD.
8.Compute Eigen values and Eigen vector using the
    Laplacian matrix.
9.Create a new graph RDD using Eigen values and
    Eigen vector.
10.Based on Eigen value buckets, decide the number
    of partitions.
11.Output the number of optimal partitions.
```

3.2Existing algorithm implementation

There are several existing graph partitioning methods in place. Among them, 3 most important algorithms are K-Means Clustering, Spectral Clustering and Agglomerative Clustering.

3.2.1K-means clustering

K-Means Clustering is one of the most commonly used unsupervised learning techniques that can be utilized to resolves the partitioning problem. This technique follows a nominal and simple way to divide a data collection into a specified number of partitions (assume k partitions). The idea is to identify K centroids, one for every partition. The result mostly depends on the identification of these centroids. Changing these centroids may result in drastic changes in the final result. Hence it is always better to choose them in such a way that they are far away from one another. For finding out the proximity, Euclidian distance is used. Each point belonging to the input data collection is connected to its nearest centroid. Hence all the data points near to a particular centroid are grouped. When all the data points are grouped, the first iteration is over. In the next iteration, centroids are recalculated and grouping is done again. This process is repeated until there are no more changes in the centroid and they have converged to a defined number of clusters [12]. The algorithm is as follows:

Algorithm: K-Means Clustering
1. Import sklearn libraries.
2. Initialize K indexes, called centroid, randomly or decisively (as per the code needs).
3. Classify every item to its proximate centroid based on the Euclidian distance and update the association resulting in groups.
4. This step is repeated until the convergence is achieved and a given number of clusters are formed.
5. Fit this method with graph RDD and decided the number of partitions. This will return the labels.
6. Use the labels to partition and color the graph nodes.
7. Output the partitioned graph.

If precomputed centroids are provided then steps 2 and 3 can be circumvented and reduce the convergence time to a single iteration [8].

3.2.2Spectral clustering

The biggest problem with the K-Means Clustering is that it depends on the selection of centroid. The algorithm’s accuracy and convergence depend on the metrics used for the selection of centroids. This mostly results in local optimal problem. Spectral Clustering solves this problem for us. Spectral Clustering either uses the Euclidian distance of the K-nearest neighbour strategy to form the cluster. In this case, Eigen values are used as distance metric and indexes are pushed to low dimension space for forming partitions. The algorithm is as follows:

Algorithm: Spectral Clustering
1. Import sklearn libraries.
2. Find out the Eigen values and Eigen vector. Pass it down to the algorithm.
3. Least Eigen value indicates the most proximity. It is used to associate a node with a chosen index.
4. This is applied to each data point and the partitions are formed.
5. Fit this method with graph RDD and number of partitions. Labels will be returned.
6. Use the labels to partition and color the graph nodes.
7. Output the partitioned graph.

Precomputed Eigen values, Eigen vector and indexes are provided to Spectral Clustering as the inputs. The graph is partitioned with a given number of groups with nodes belonging to each cluster designated with a particular color.

3.2.3Agglomerative clustering

Hierarchical clustering is one of the most prominent techniques which gives promising results. Groups tend to maintain an inherent hierarchy. In hierarchical clustering according to the level, splitting or combing takes place. There are 2 types of hierarchical clustering- Agglomerative and Divisive Clustering. In this project, Agglomerative Clustering is used. The Agglomerative Clustering is a bottom-up approach wherein the Divisive Clustering technique is a top-down approach. In Agglomerative Clustering, in the beginning, each data node is considered as a partition. Nearest neighbouring nodes start connecting and form an alliance. Gradually connection merges and partitioning continue. This iterative approach continues until the required number of partitions are created.

Algorithm: Agglomerative Clustering
1. Begin with the disjoint clusters having level 0 and sequence number 0.
2. Find the least distance pair of clusters in the current clustering.
3. Increment the sequence number.
4. Update the distance matrix by deleting the rows and columns corresponding to clusters and adding a row and column corresponding to the newly formed cluster.
5. If all the data points are in one cluster then stop, else repeat from step 2.
6. Fit this method with graph RDD and decided number of partitions. This will return the labels.

7. Use the labels to partition and color the graph nodes.
8. Output the Partitioned graph.

3.3Custom dynamic partitioning

Though the result yielded is quite accurate in case of existing partitioning techniques, they are suitable for only certain kinds of graphs. All these algorithms are dependent majorly on the relationship metric or distance metric used. More accurate results are obtained if the distance metric is stronger. For example, consider the K-Means algorithm, which uses Euclidian distance as the inbuilt distance metric. Spectral Clustering uses Eigen values itself. Agglomerative makes use of squared Euclidian distance. Thus, writing an appropriate distance metric would allow building an effective custom partitioning method. The same principle is used to build this partitioning method.

The custom partitioning technique is built on the basis of scipy package. It uses linkage and fcluster functionalities. First, the custom partitioning function is written. The following distance metric is used.

$$Distance\_metric=(a-b)*a + (b - a)* b \qquad (2)$$

In this formula a stands for the similarity index and b stands for the node which is used to find the distance between the similarity index and given node. If this distance is lesser then the relation is stronger. This formula is derived in iterative greedy approach. The formula is started with a + b and slowly disturbances are removed. For each combination, the result is evaluated and the conclusion is derived as equation (2) gives optimal results. The sum of the distance metric for the nodes is returned.

Further linkage program is used to partition the data. This makes use of 3 parameters. The first one is input data, the second is the method of calculation and the third parameter is the name of the custom function for distance metric. This will generate the partitioned data which can be stored in a variable.

The partitioned data from the above function is passed to fcluster function which is responsible for dividing the graph into the mentioned number of partitions. This function also takes 3 parameters. The first one is partitioned data returned by linkage function, second is the number of the clusters to be formed and the third one is the criterion for classifying. Maxclust option is used to form the flat partitions of the graph.

Following is the algorithm for Custom dynamic partition:

```
Algorithm: Custom Dynamic Partitioning of graphs
1. Input the Graph RDD.
2.Initially, each node is considered as a single
  partition.
3.Compute the distance between a node and index
  using the custom distance formula:
  $Distance= ( a - b ) * a + ( b - a ) * b$
4. Store nodes with least distance to particular index
  into one partition.
5.Repeat steps 3 and 4 until all items are grouped into
  a given number of partitions.
6.Fit this method with graph RDD and decided
  number of partitions. This will return the labels.
7.Use the labels to partition and color the graph
  nodes.
8. Output the partitioned graph.
```

4. Result

This section describes the results of the partitioning algorithms. In this project, all the four algorithms are executed for Twitter data set with different amounts of data. First, the tests are run for the Twitter account of Tim O'Reilly with 1.8 million followers. Data set boils down to 500 nodes after processing for depth 4. The next set of data is executed for the Twitter account of Narendra Modi with 47.6 million followers. Data processing selects 1067 nodes for a depth of 6.

4.1Results of the algorithms for the graph with 500 nodes

Figure 3 represents the input graph with 500 nodes. Algorithms are executed for this input graph. This graph does not have any partitions. Hence all the nodes are represented by blue color.

Based on the similarity index, it is inferred that in this dataset there are 2 partitions. Further, after the application of algorithms graph is divided into 2 partitions. Partitions are recognized by violet and yellow color. This signifies that nodes assigned with a particular color have more similarity among them. In this case, those nodes represent Twitter followers with maximum interaction between each other. Thus, the degree to which most similar nodes are put together in one partition, decides the accuracy of the algorithm. Also, this increases the performance of the system as logically related nodes are together.

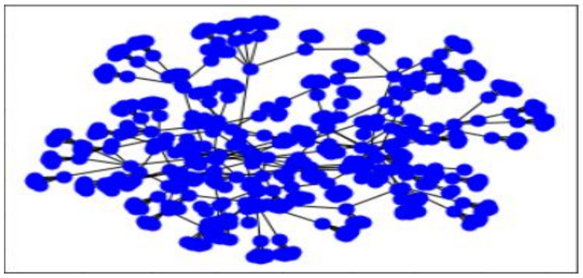

Figure 3 Input graph with 500 nodes

4.1.1K-means clustering

Figure 4 shows the output of the K-Means Clustering algorithm for 500 nodes. K-Means Clustering is done based on Euclidian distance. The nodes having the least Euclidian distance to the centroid are grouped in one partition. This algorithm divides the data in such a way that 419 nodes are put in one partition (represented by violet) which are close to one centroid. Remaining 81 nodes are placed in the second partition (represented by yellow) which have the least Euclidian distance to the second centroid. This indicates 419 nodes in violet partition interact more with each other. This result is considered most accurate as K-Means clustering is the standard algorithm used for the partitioning of data. This also gives consistent high performance with pretty good speed and accuracy. The speed and accuracy are because of the exhaustive grouping of nodes with the standard distance metric (Euclidian distance). Further comparison between the existing algorithms is made by keeping K-Means Clustering as standard. If the other algorithms classify the same 419 nodes in the violet partition and 81 nodes in the yellow partition, then those algorithms are considered 100% accurate.

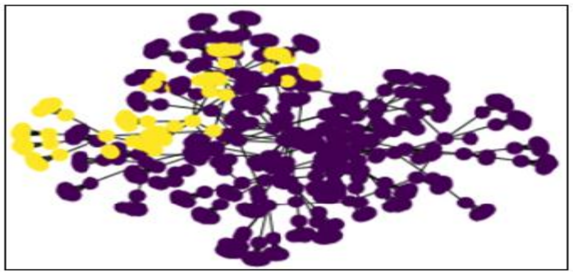

Figure 4 K-means clustering output for 500 nodes

4.1.2Spectral clustering

Figure 5 displays the output of Spectral Clustering for 500 nodes. This is the fastest algorithm when compared to other algorithms. This is because Spectral clustering uses Eigen value as the distance metric which is already calculated during the earlier stages. For executing the 500 nodes dataset it takes 7.03 seconds. It gives 100% accurate results as this also classifies the same 419 nodes in the violet partition which have the least Eigen value difference to the first index and 81 nodes in the yellow partition which have the least Eigen value difference to the second index. However, accuracy is unstable as the data grows in size.

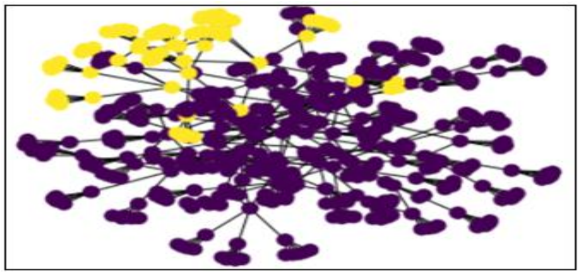

Figure 5 Spectral clustering output for 500 nodes

4.1.3Agglomerative clustering

Figure 6 displays the output of Agglomerative Clustering when executed for 500 nodes. This algorithm is slower compared to others. As the algorithm follows the bottom-up hierarchical method and elaborately groups node by node, execution time is affected. However, this detailed grouping increases the accuracy. It takes around 7.76 seconds to execute 500 nodes. As this algorithm is based on squared Euclidian distance, nodes having the least squared Euclidian distance to the first index are put in one partition (represented by violet color) and remaining in the second partition (represented by yellow color). The accuracy of this algorithm is 98% as it classifies 414 nodes in the violet partition and 86 nodes in the yellow partition.

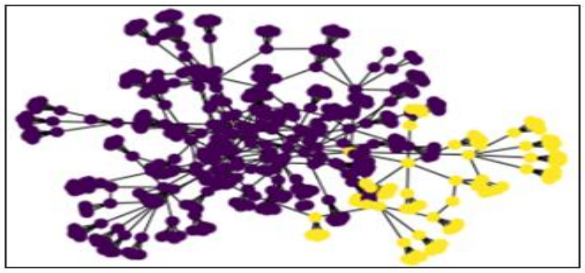

Figure 6 Agglomerative clustering output for 500 nodes

4.1.4Custom dynamic partitioning

Figure 7 displays the output of the custom partitioning method for 500 nodes. Considering other algorithms, custom developed partitioning method is stable. It takes 7.72 seconds to execute which is nearly equal to the execution time of other algorithms. However, accuracy is better than other algorithms and stable with respect to the size of the data. This algorithm classifies the data based on the custom distance metric. Hence the nodes having the least distance metric value to the first index are stored in the violet partition and remaining in the yellow partition. This algorithm gives 99.8 % accuracy as it classifies 417 nodes in the violet partition and 83 nodes in the yellow partition.

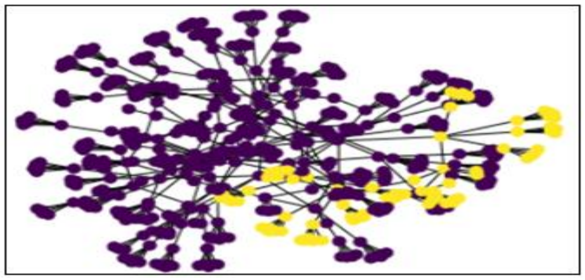
Figure 7 Custom dynamic partitioning output for 500 nodes

4.2Results of the algorithms for the graph with 1067 nodes

Figure 8 represents the input graph with 1067 nodes. Algorithms are executed for this graph. This graph does not have any partitions. Hence all the nodes have blue color.

Based on the similarity index, it is inferred that in this dataset there are 3 partitions. Partitions are recognized by violet, yellow and blue color. Thus all the nodes which have maximum interaction with each other goes inside a particular color partition. As the nodes which frequently interact with each other are together, the performance of the application using this partitioned data increases, eliminating the communication latency.

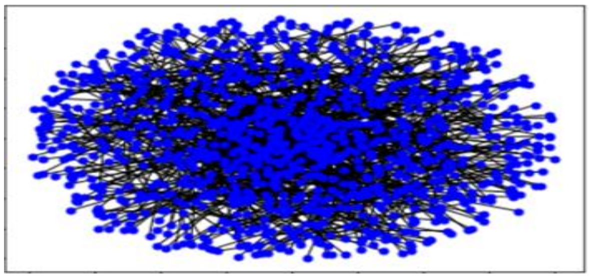
Figure 8 Input graph with 1067 nodes

4.2.1K-means clustering

Figure 9 represents the output of the K-Means Clustering algorithm for 1067 nodes. K-Means being the standard, it gives good result in case of increased data size. For the increase in data size to double, there is only an increase of 0.54 seconds. It partitions data into 3 groups with 100% accuracy as it is the standard algorithm. It partitions in such a way that 993 nodes are in the violet partition which has the least Euclidian distance to the first centroid, 55 nodes are in the yellow partition with the least distance to the second centroid and 19 nodes are in the blue partition which has the least distance to the third index. Any algorithm that partitions data in the same way is considered 100% accurate.

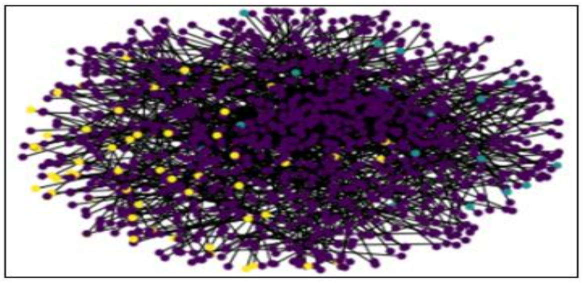
Figure 9 K-Means clustering output for 1067 nodes

4.2.2Spectral clustering

Figure 10 displays the output of Spectral Clustering when executed for 1067 nodes. This algorithm shows less accuracy in the case of 1067 nodes. Accuracy drops to 89% as it puts 914 nodes in the violet partition which has the least Eigen value difference to the first index. 55 nodes which has the least Eigen value difference with the second index are stored in the yellow partition. Remaining 98 nodes are saved in the blue partition. This shows the inconsistency of this method in the case of growing data. Still, it is the fastest with a runtime of 8.03 seconds.

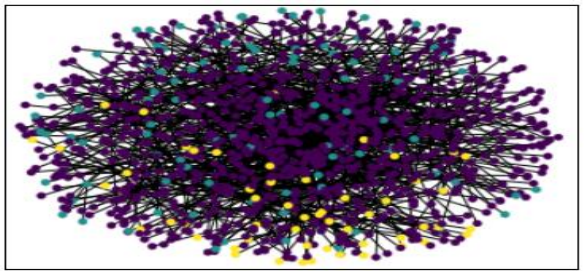
Figure 10 Spectral clustering output for 1067 nodes

4.2.3Agglomerative clustering

Figure 11 represents the output of Agglomerative Clustering when executed for 1067 nodes. Agglomerative Clustering shows an increase in accuracy when data size grows. It gives 99.06% accuracy as it divides 1003 nodes in the violet partition which has the least squared Euclidian distance with the first index, 45 nodes in the yellow partition having the least squared Euclidian distance with the second index and remaining 19 nodes in the blue partition. Though the execution time (8.51 seconds) is high, it pays off with better accuracy. This is because of the hierarchical approach that it follows.

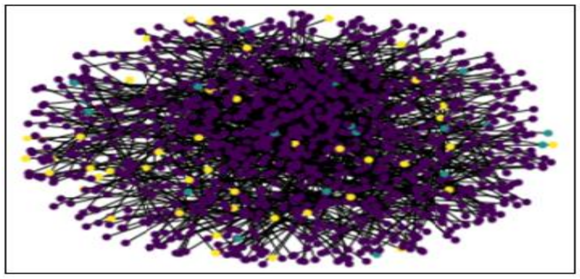

Figure 11 Agglomerative clustering output for 1067 nodes

4.2.4Custom dynamic partitioning

Figure 12 displays the output of custom dynamic partitioning for 1067 nodes. Newly implemented custom dynamic partitioning technique maintains both speed and accuracy when the data grows in size. Additionally, accuracy is high and consistent. It takes of 8.32 seconds execution time which is nearly equal to other algorithms. It gives an accuracy of 99.81. This accuracy is because of the custom distance metric used. Accuracy is 99.81% because it classifies 991 nodes in the violet partition which have the least custom distance value with the first index, 57 nodes in the yellow partition with the least custom distance value to the second index and remaining 19 nodes in the blue partition. This classification is almost same as the output of standard K-Means clustering.

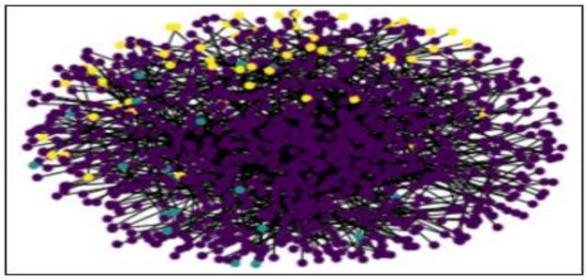

Figure 12 Custom dynamic partitioning output for 1067 nodes

5. Performance comparison

The performance of the system is measured based on 2 parameters. First is based on the execution time of the algorithm. Second is the accuracy of the algorithms with respect to first.

5.1Analysis based on the execution time

The result of the execution time analysis is as shown in *Table 1*. The execution time of the algorithm can be obtained by using magic function available in python. This function needs to be specified at the top of the cell or block. On the execution of the cell, time taken by the cell to run is displayed as wall time. Execution time is measured for each algorithm.

Table 1 Analysis based on the execution time of the techniques

Partitioning technique	**500 nodes (in Seconds)**	**1067 nodes(in Seconds)**
K-Means Clustering	7.81	8.35
Spectral Clustering	7.3	8.03
Agglomerative Clustering	7.76	8.54
Custom Partitioning	7.78	8.32

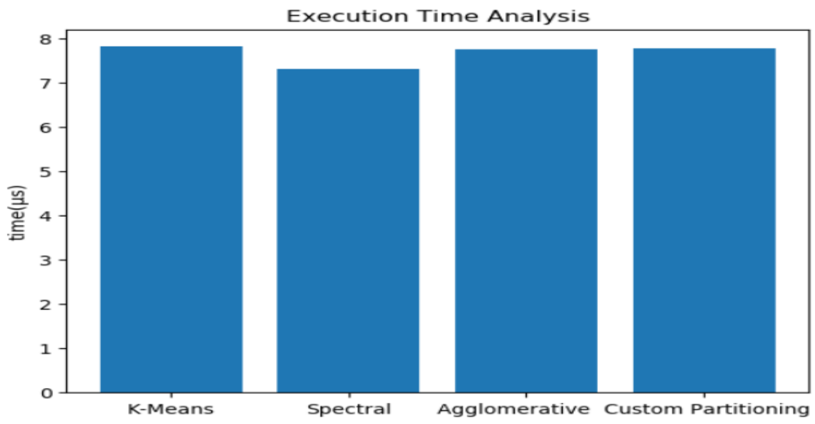

Figure 13 Execution time analysis for 500 nodes

As displayed in *Figure 13* existing algorithms takes around 7 seconds to run the data for 500 nodes. Among the existing algorithms, Spectral Clustering is the fastest with only 7.3 seconds. However, the custom dynamic partition takes 7.78 seconds which is nearly equal to the execution time of other algorithms.

Figure 14 explains the execution time analysis for 1067 nodes. As the data size increases, there is an increase in the running time. However, in this case, though the data size is double from 500 nodes to 1000 nodes, the increase in the running time of algorithms is not much higher. The algorithms take around 8 seconds to execute. Still the Spectral Clustering is fastest with 8.03 seconds, while custom dynamic partitioning takes 8.32 seconds which is better than other 2 algorithms (K-Means Clustering and Agglomerative Clustering). Thus, the custom partitioning technique maintains consistency with the execution time.

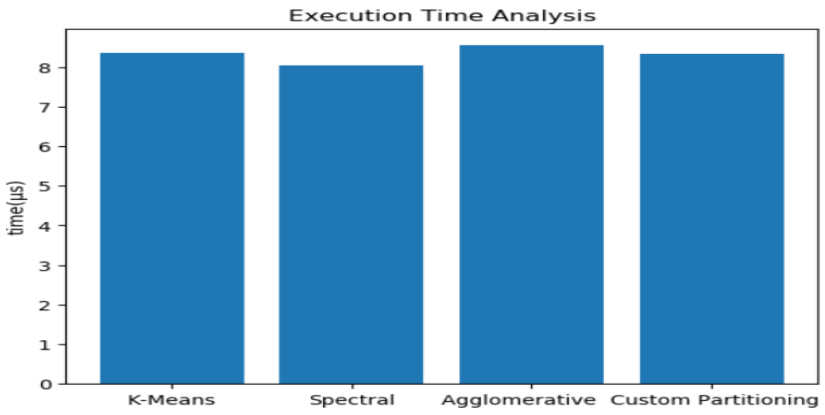

Figure 14 Execution time analysis for 1067 nodes

5.2Analysis based on the accuracy of the techniques

Accuracy is measured based on the number of nodes classified. In this project, accuracy is measured with respect to K-Means algorithm's results. This is because K-Means Clustering is the most widely used and accurate partitioning algorithm. K-Means classifies 419 nodes in one partition and 81 in remaining. If the other algorithm classifies same number of nodes in the given partition, then that algorithms accuracy is considered 100%. This is measured using the accuracy_score function in the sklearn module. This function takes two values. First one is the prediction value (or benchmark algorithms partitioning) and the second is the current partitioning result. The results obtained are printed in percentage. From this, the percentage of accuracy is obtained.

Table 2 gives the accuracy value of algorithms for 500 and 1067 nodes. AS K-Means Clustering is considered as the standard, its accuracy is always 100%. Based on the number of the nodes classified by the other algorithms respective accuracy is computed. For example, Spectral Clustering classifies 419 nodes in one partition and 81 nodes in other. As this is same as the K-Means Clustering, accuracy is considered 100%. But Agglomerative Clustering classifies 414 in one partition and 86 in other. So, its accuracy is 98%. In the same way accuracy is measured for all the algorithms.

As shown in *Figure 15* Spectral Clustering gives 100% accuracy in the case of 500 nodes. This shows that the distance metric used (Eigen values) gives absolute accuracy in case of a small amount of the data. In the same way, Agglomerative Clustering gives 98%. However, custom dynamic partition gives 99.8% of accuracy in this case, which is pretty good when compared to other algorithms for 500 nodes. The following chart shows the accuracy comparison in the case of 500 nodes. *Figure 16* explains the accuracy analysis for 1067 nodes. Accuracy of Spectral Clustering goes down to 89% when running for 1067 nodes. Thus, for a large number of nodes, Spectral Clustering gives lower accuracy. Agglomerative Clustering displays 99.06% accuracy. There is a little improvement in the case of Agglomerative Clustering. However, these algorithms' accuracy is not consistent with the size of the data. Custom partitioning gives the same amount (99.81%) accuracy even when the data size is increased. Thus, custom partitioning method gives both better and consistent results.

Table 2 Execution time analysis

Partitioning Technique	500 nodes (in Percentage)	1067 nodes (in Percentage)
K-Means Clustering	100 (Benchmark)	100(Benchmark)
Spectral Clustering	100	89.03
Agglomerative Clustering	98	99.06
Custom Partitioning	99.8	99.81

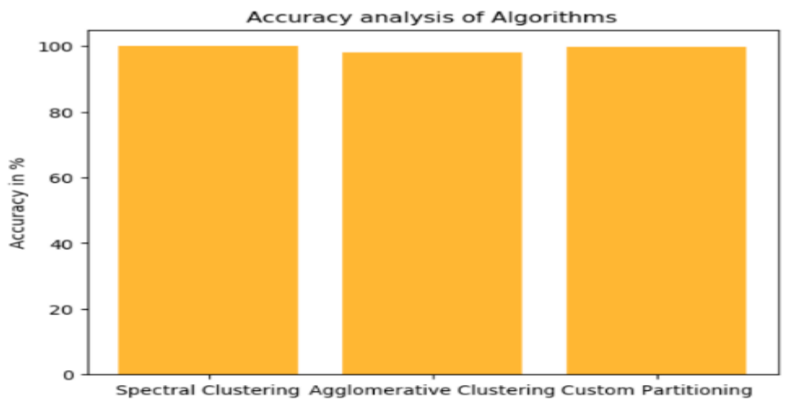

Figure 15 Accuracy analysis for 500 nodes

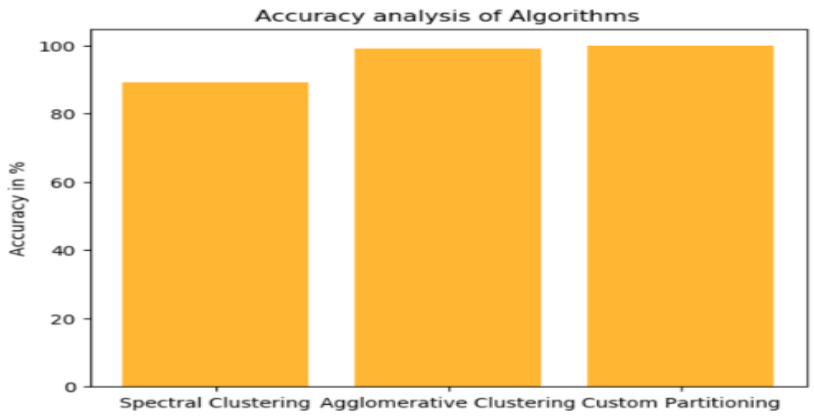

Figure 16 Accuracy analysis for 1067 nodes

6. Conclusion and future work

Graph partitioning is one of the highly discussed research topics in the Big Data environment. Tools like Apache Spark are providing a stable platform for carrying out experiments on the same. There are several existing graph partitioning methods already available in Spark. Each of these algorithms exhibits both strength and weakness in certain measures. Hence there was a need to develop a custom algorithm to incorporate the strengths of these existing algorithms and eliminate the weakness.

In this project, various graph partition methods in Apache Spark are explored. Various existing algorithms like K-Means Clustering, Spectral Clustering and Agglomerative Clustering are implemented. Based on the study of these algorithms, a new custom dynamic partitioning method is developed. The capacity to dynamically decide the optimal number of partitions is integrated into the application. The new method uses an external custom distance metric and does the partition based on those values. All the algorithms are executed on the same datasets. The size of the data is varied and the results are accumulated.

Detailed performance analysis is conducted on all the algorithms based on the gathered results. Execution time of the custom algorithms is almost same as other methods, even better in some cases. However, when it comes to accuracy, the custom partitioning algorithm provides better results in all the cases.

Also, the custom partitioning technique displays consistency in accuracy with all the volume of the data compared to existing algorithms. As future work, we need to enhance the custom dynamic partitioning technique to take less time for execution. Also, there is a need to do an exhaustive research on the integration of real-time graph partitioning with Artificial Intelligence techniques.

1. The performance of this code has to be tuned to process more data in unit time.
2. There is a need to do an exhaustive research on the integration of real-time graph partitioning with artificial intelligence techniques.

Acknowledgment
None.

Conflicts of interest
The authors have no conflicts of interest to declare.

References

[1] Mokashi VS, Kulkarni DB. A review: scalable parallel graph partitioning for complex networks. In second international conference on intelligent computing and control systems 2018 (pp. 1869-71). IEEE.

[2] Hassan M, Bansal SK. Data partitioning scheme for efficient distributed RDF querying using apache spark. In international conference on semantic computing 2019 (pp. 24-31). IEEE.

[3] Gounaris A, Kougka G, Tous R, Montes CT, Torres J. Dynamic configuration of partitioning in spark applications. IEEE Transactions on Parallel and Distributed Systems. 2017; 28(7):1891-904.

[4] Bertolucci M, Carlini E, Dazzi P, Lulli A, Ricci L. Static and dynamic big data partitioning on apache spark. In PARCO 2015 (pp. 489-98).

[5] Abughofa T, Zulkernine F. Towards online graph processing with spark streaming. In international conference on big data 2017 (pp. 2787-94). IEEE.

[6] Taloba AI, Riad MR, Soliman TH. Developing an efficient spectral clustering algorithm on large scale graphs in spark. In international conference on intelligent computing and information systems 2017 (pp. 292-8). IEEE.

[7] Atashkar AH, Ghadiri N, Joodaki M. Linked data partitioning for RDF processing on Apache Spark. In international conference on web research 2017 (pp. 73-7). IEEE.

[8] Tian X, Guo Y, Zhan J, Wang L. Towards memory and computation efficient graph processing on spark. In international conference on big data 2017 (pp. 375-82). IEEE.

[9] Rajan AK, Bhaiya D. Accelerated kerninghan lin algorithm for graph partitioning. In international conference on advances in computing, communications and informatics 2017 (pp. 174-8). IEEE.

[10] Wang M, Yang W, Li H, Lin Y, Chen J. Metis-CIC: a new mesh partitioning heuristic for parallel preconditioned iterative methods in CFD. In international conference on high performance computing & simulation 2016 (pp. 188-95). IEEE.

[11] Kyong J, Jeon J, Lim SS. Improving scalability of apache spark-based scale-up server through docker container-based partitioning. In proceedings of the international conference on software and computer applications 2017 (pp. 176-80).

[12] Olukanmi PO, Twala B. K-means-sharp: modified centroid update for outlier-robust k-means clustering. In pattern recognition association of south Africa and robotics and mechatronics 2017 (pp. 14-9). IEEE.

Dr. Geetha. J. is working as an Associate Professor in Computer Science and Engineering Department of Ramaiah Institute of Technology, Bangalore. Her areas of interest include Cloud Computing, Big Data, Semantic Web, Graph Theory and Web Design.

Email: geetha@msrit.edu

Dr. Jayalakshmi D S is working as an Associate Professor in Computer Science and Engineering Department of Ramaiah Institute of Technology, Bangalore. Her areas of interest include Cloud Computing, Big Data and Computer Graphics.

Email: jayalakshmids@msrit.edu

Harshit. N. G. is a graduate student currently pursuing M.Tech. in Computer Science and Engineering in Ramaiah Institute of Technology, Bangalore. His area of interest includes Big Data, Computer Architecture and Web Technologies.

Email: ngharshit@gmail.com