Leveraging Distributed Computing for Subspace Clustering

Sandhya Harikumar[1] P. Munindra Naidu[2] S. Surendhar[3]

Abstract— **Subspace clustering is an extension of the traditional clustering method that seeks to find clusters in various subspaces within a given dataset. Usually, the data is huge, spread across different machines and it may contain irrelevant or redundant features. Data clustering might not be dependent on all the features in the dataset, it mostly forms clusters based on some important features and we need to check the clusters in those subspaces only. So to get these clusters for the important features, we apply the subspace clustering algorithm to find relevant features within a subset of dimensions to obtain better results. There are two ways in which we can solve this problem either to bring the entire data in one machine (like supercomputer where computation is faster) so that the subspace clustering algorithm can be applied to this. There is another way instead of bringing data on one machine, we can try to run distributed subspace clustering algorithm on all machines which are having data present within them so that one system can be acted as master and remaining works as slaves. In this paper, we are going to do the later way within spark environment. Here master machine assigns the computations to the slave machines that need to be done and the results obtained from the nodes are sent to the master so it decides the best result. This distributed platform can be realized using Spark, Hadoop, Distributed Python (Dispy). Here subspace clustering algorithm is implemented in the Spark platform and results are compared against Subspace Clustering on single machine and Dispy.**

I. INTRODUCTION

A. Problem Statement

The solution which is being addressed here is that to devise a Subspace clustering algorithm on spark platform to the Data distributed across various sites, without breaching the autonomy of the respective systems.

Consider a Company which has multiple numbers of branch spread across different parts of the country. The Management wants to improve the facilities, increase the number of employees and other staff members. Here, as the data is spread across different machines in the same branch and also spread across various locations, it is not distributed uniformly and data consists of millions of records. The data consists of features related to facilities, staff etc. The requirement is to get the insights and analysis from the given dataset. Here dataset has so many features which are irrelevant and redundant. So, it becomes a high dimensional. As per the requirement, only a few features are needed and the rest of these are not important.

This can be realized using a subspace clustering algorithm where it identifies the clusters in different sub spaces. A subspace is a space that wholly contained in other space, or whose points or elements are all in another space. So the features which are necessary for the requirements are chosen by the subspace algorithm. The distributed platform can be achieved using Apache Spark. It is an open-source distributed general-purpose cluster-computing framework. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance[1]. Usually the data is spread across different locations so there are different systems which have distinct computation speeds. So we achieve parallelism by running the subspace clustering algorithm on all those machines which are the part of the organization. Here it is implemented using the multi-node cluster in the spark platform[2].

B. Background Study

Clustering is one of the main techniques used for the unsupervised knowledge of discovering classes out of unlabeled datasets. This technique uses the idea of similarity to group data points in entities known as clusters. Traditional clustering algorithms such as partitioning based approaches takes all the features into examination for clustering. However, as the datasets become larger and higher-dimensional, these algorithms fail to expose the useful clusters due to the existence of irrelevant and redundant features[3].

A new format of clustering algorithms, which has been derived from the frequent pattern mining field, rapidly constituted a novel field named subspace clustering[3]. These algorithms have been designed to find clusters hidden in the subsets of the actual feature space. It also helps in avoiding the curse of dimensionality. Nevertheless, up to our knowledge none of these techniques scales well with respect to the size of datasets.

During our analysis, we found have found that there is a lack of parallel subspace clustering algorithms that can be scalable for high dimensional data on top of current Big Data distributed platforms. Recently, the MapReduce model together with the Hadoop framework has become the most favoured Big Data distributed framework[4]. Nevertheless, there are certain limitations for the MapReduce and it also lacks in suitability for the iterative Machine Learning algorithms which have motivated researchers to propose different alternatives, Spark being one of the most representative. Therefore, it is a big challenge to design a scalable subspace clustering algorithm on top of Apache Spark Framework.

C. Motivation

Distributed systems can improve the entire performance of the computing system and Price to Performance ratio for the system is more favourable for a distributed system. Some applications are intuitively distributed problems and they could easily be solved by the introduction of the distributed platforms. These Distributed platforms allow the machines to share the resources both by hardware or software. Each piece of hardware can be replaced if it fails. It also allows the distributed environment to grow incrementally as computers are added one by one[5].

Bringing the entire data in one machine which is spread across various machines is time-consuming and also costly. Technological advantages in the present are far better than the olden days, So it is easier to find a solution to work on the distributed data. If the data is not moved from the system then there is no need for the cables and also data need not be encrypted. If the data is not moved from the respective machines then it cannot tamper or values of it cannot be changed which means data is secured. One of the prime objective is the privacy of the data and how the distributed environment secures it.

When compared to very large centralized systems, Distributed systems can be much more cost-effective and secure. Their initial cost could be higher than standalone systems, but only up to a certain point later which they are more about economies of scale. A distributed system made up of many minicomputers can be more cost-effective than a mainframe machine. One of the most used distributed platforms in modern days is Apache Spark[6]. Spark can use either memory or disk or both for processing, whereas MapReduce is only for disk-based computation. MapReduce uses persistent storage for processing the data whereas Spark uses its own data structure called Resilient Distributed Datasets (RDDs), which are immutable. These RDDs can achieve Fault Tolerance in the distributed system[7]. It remembers the dependencies between every RDD involved in the operations, through the lineage graph created in the DAG, and if there is any failure, Spark refers back to the lineage graph to apply the same operations to perform the tasks.

II. Related Work

There is a proposed framework in Spark that supports running applications while retaining the scalability and fault tolerance of MapReduce. To achieve this Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark did outperform Hadoop by 10x in iterative machine learning jobs and can be used to interactively query[8].

An interface which has been developed called MLI which is an Application Programming Interface designed to address the challenges of building Machine Learning algorithms in a distributed setting based on data-centric computing[9]. Its primary goal is to simplify the development of high-performance, scalable, distributed algorithms. Their results show that, relative to existing systems, this interface can be used to build distributed implementations of a wide variety of common Machine Learning algorithms with minimal complexity and highly competitive performance and scalability.

The K-Medoids clustering algorithm not only solves the problem of the K-Means algorithm on processing the outlier samples but also it is not able to process big-data because of the time complexity. In order to break the big-data limits, the parallel K-Medoids algorithm HK-Medoids based on Hadoop was used[10]. Every submitted job has many iterative MapReduce procedures: In the map phase, each sample was assigned to one cluster whose centre is the most similar with the sample; in the combined phase, an intermediate centre for each cluster was calculated; and in the reduce phase, the new centre was calculated. The iterator stops when the new centre is similar to the old one. The experimental results showed that HK-Medoids algorithm has a good clustering result and linear speedup for big-data.

There is also an implementation of DBSCAN algorithm on the distributed platform. DBSCAN is a well-known density-based data clustering algorithm that is widely used due to its ability to find arbitrarily shaped clusters in noisy data[11]. However, DBSCAN is hard to scale which limits its utility when working with large data sets. It also presents a new algorithm based on DBSCAN using the Resilient Distributed Datasets approach: RDD-DBSCAN. RDD-DBSCAN overcomes the scalability limitations of the traditional DBSCAN algorithm by operating in a fully distributed fashion. This paper also evaluates the implementation of RDD-DBSCAN using Apache Spark, the official RDD implementation.

In high-dimensional data, clusters of objects often exist in subspaces rather than in the entire space.There is a data sparsity problem faced in clustering high-dimensional data. In the new algorithm,they have extended the k-means clustering process to calculate a weight for each dimension in each cluster and use the weight values to identify the subsets of important dimensions that categorize different clusters[12]. This is achieved by including the weight entropy in the objective function that is minimized in the k-means clustering process.The new algorithm is also scalable to large data sets.

A new algorithm called fast and exact k-means clustering (FEKM)[13], which typically requires only one or a small number of passes on the entire dataset and provably produces the same cluster centres as reported by the original k-means algorithm. The algorithm uses sampling to create

initial cluster centres and then takes one or more passes over the entire dataset to adjust these cluster centres. They also describe and evaluates the distributed version of FEKM, which they refer to as DFEKM. This algorithm is suitable for analysing data that is distributed across loosely coupled machines. Unlike the previous work in this area, DFEKM probably produces the same results as the original k-means algorithm. DFEKM is suitable for analysing data that is distributed across loosely coupled machines.

A method for finding clusters of units in high-dimensional data having the steps of determining dense units in selected subspaces within a data space of the high-dimensional data[14], determining each cluster of dense units that are connected to other dense units in the selected subspaces within the data space, determining maximal regions covering each cluster of connected dense units, determining a minimal cover for each cluster of connected dense units, and identifying the minimal cover for each cluster of connected dense units. CLIQUE scales linearly with the size of input and has good scalability.

III. Clustering

A. Introduction to Clustering

Clustering is the task of dividing the data points into a number of groups such that data points which belong to the same groups are much more similar to other data points in the same group than those of the data points in other groups. The method of identifying similar class of data in a dataset is called clustering. This is one of the most popular techniques in the field of data science[15].

Clustering comes under the category of unsupervised machine learning. In Unsupervised learning, we only have input data and no corresponding output variables. The goal of unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above there are no correct answers and there is no teacher. Algorithms are left to their own device to discover and present the interesting structure in the data.

B. Dimensionality Reduction

Techniques for clustering high dimensional data includes feature selection and feature transformation techniques. Feature transformation techniques summarize the given input dataset in less number of dimensions by creating combinations of all the possible attributes[16]. These techniques are very successful in uncovering latent structures in the given datasets.

However, since they compute the relative distances between objects, they are less effective when there is a vast number of irrelevant attributes that hide the clusters in noise. Also, the new attributes are combinations of the original's features and may be very difficult to interpret the new features in the context of the domain. These methods only select the most relevant dimensions from a dataset to reveal groups of objects that are similar on only a subset of their attributes.

While fairly successful in many datasets, feature selection algorithms have difficulties when clusters are to be found in different subspaces. This type of data is the motivation behind the advancement of subspace clustering algorithms. These algorithms improve the concepts of feature selection methods a step further by selecting relevant subspaces for each cluster separately.

While feature extraction might appear to be superior in performance compared to feature selection, the choice is predicated by the application[17]. The attributes from feature extraction usually lose physical interpretation, which may be an issue based on the task at hand. For example, if you are designing a very expensive data collection task with costly sensors and need to economize on the attributes (number of different sensors), you'd want to collect a small pilot sample using all the available sensors and then pick the ones that are most informative for the big data collection task. So, finding important features using subspace clustering should be a solution to extract the features based on their significance[18].

IV. Subspace Clustering

Clustering analysis seeks to discover groups or clusters, of similar objects. The objects are usually represented as a vector of measurements or a point in multidimensional space[19]. The similarity between objects is often determined using distance measures over the various dimensions in the data set. Subspace Clustering is an extension of traditional clustering that seeks to find clusters in different sub spaces within a dataset as shown in Figure 2.1. Technology advances have made data collection easier and faster resulting in larger more complex data sets with many objects and dimensions.

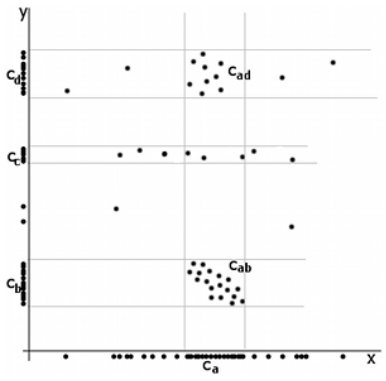

Fig. 1. Subspaces in PROCLUS

As the size of the dataset becomes larger and more data entries are created, the data should be able to adapt to the existing algorithms and cluster quality and speed are to be maintained. Normal clustering algorithms consider

all the dimensions of an input dataset in an attempt to gain insights on the data as much as possible so that it gains properties about each object described[3]. In higher dimensional data, many of the dimensions are usually irrelevant. These inapt dimensions can confuse clustering algorithms by hiding clusters which can be formed by some dimensions in the whole data. In high dimensions it is usual for all of the objects within the dataset to be nearly equidistant from each other, completely masking the clusters.

Clustering algorithms also suffer from another problem with the high dimensional data which is known as the curse of dimensionality. As the features in the dataset increases, distance measures can become increasingly meaningless. Adding more dimensions to the data can spread out the points until they are almost equidistant from each other[20]. If we continue to add dimensions, the points continue to spread out till all of them are almost equally far apart and distance is no longer very meaningful.

The problem gets worsen when objects are related in different ways in various subsets of dimensions. In these types of relationship, the subspace clustering algorithms seek to uncover the clusters that are hidden in subspaces. To find such clusters, the irrelevant features need to be removed to allow the clustering algorithm to focus only on the useful dimensions. Clusters found in lower dimensions tend to be more easily interpretable, allowing the user to get direct further study.

Certain methods are needed which can uncover clusters formed in different subspaces of massive and high dimensional datasets. We also need to represent them in easily interpretable and meaningful ways. This scenario is becoming common as we try to examine data from different perspectives. A query for the term Bush could return documents on the president of the United States as well as information on landscaping. If these documents are clustered using the words as attributes, the two groups of documents would likely be related to different sets of attributes. Subspace clustering algorithms is a solution to answer these challenges.

V. Distributed Platforms

A. Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce model and it also extends the MapReduce model which helps us to efficiently use it for much more types of computations, which include interactive queries and stream processing. The main feature of Apache Spark is its in-memory cluster computing that accelerates the processing speed of a spark application. Spark design can help us to do batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workloads in a respective system, it reduces the management burden of maintaining separate tools.

B. Spark Architecture

Spark applications are similar to the Hadoop MapReduce Jobs. Each application consists of a self-contained computation method which runs some user given code to get a result. Spark applications can use the resources which are distributed across multiple nodes. Spark has its own data structure called as resilient distributed datasets(RDD), which is a fault-tolerant collection of elements that are operated in parallel mode[21]. There are currently two types of RDDs: parallelized collections, which take an existing Scala collection and run functions on it in parallel, and Hadoop datasets, which run functions on each record of the files in Hadoop distributed file system. Both types of RDDs can be operated by the same methods.

In spark, Each application has a driver process which controls the execution as shown in Figure **??**. This process can run in the client mode or in the cluster mode. Client mode is a little simpler, but cluster mode allows you to easily log out after starting a Spark application without terminating the application. For the client mode, the input file path must be given a local file. Spark starts the executors within each node to perform computations. There may be many executors, distributed across the cluster, depending on the size of the job[6]. After loading some of the executors, Spark attempts to match tasks to executors. Note that Spark does not start executors on nodes with cached data, and there is no further chance to select them during the task-matching phase. This is not a problem for most workloads since most workloads start executors on most or all nodes in the cluster. Spark can run in multiple modes, Standalone mode where master machine coordinates the efforts of all the workers in the nodes, which run the executors. It is a default mode and cannot be used on secure clusters. Other mode called YARN mode where YARN resource manager performs the functions of spark master on all executors[21]. Although the YARN mode is complex to setup, it supports security, and it provides better integration with Yarn's cluster-wide resource management policies.

For writing Spark programs it is vital to know about RDD's and how transformations and actions are applied on the RDD's. Every spark application has a driver process which is useful for the interaction between the executors and the nodes on the cluster. The driver program is responsible for the flow of the work and the executor processes are responsible for this work in the form of tasks. Each executor have some slots for running these tasks. When an action is invoked inside a Spark application it triggers the launch of a Spark job to fulfil it. Spark examines the graph of the RDDs on which that action depends and formulates an execution plan. It starts with the farthest-back RDDs which depend on no other RDDs or reference already-cached data and ends in the final RDD which produces the actions results[7]. The execution plan consists of assembling the jobs transformations into stages as shown in Figure **??**. A

stage corresponds to a collection of tasks that all execute the same code, each on a different subset of the data. Each stage contains a sequence of transformations that can be completed without shuffling the full data[7].

However, Spark also supports transformations with wide dependencies such as groupByKey and reduceByKey. In these dependencies, the data required to compute the records in a single partition may reside in many partitions of the parent RDD. All of the tuples with the same key must end up in the same partition, processed by the same task. To satisfy these operations, Spark must execute a shuffle, which transfers data around the cluster and results in a new stage with a new set of partitions.

There are many different tasks that require shuffling of the data across the cluster, for instance, table joins to join two tables on the field id, you must be sure that all the data for the same values of id for both of the tables are stored in the same chunks[22]. Imagine the tables with integer keys ranging from 1 to 1,000,000. By storing the data in same chunks I mean that for instance for both tables values of the key 1-100 are stored in a single partition/chunk, this way instead of going through the whole second table for each partition of the first one, we can join partition with partition directly, because we know that the key values 1-100 are stored only in these two partitions.

C. Dispy (Distributed Python)

Dispy is a comprehensive and easy to use framework for creating and using clusters to execute computations in parallel across multiple processors in a single machine, among many machines in a cluster, grid or cloud[9]. Dispy is best suited for data parallel paradigm where a computation is evaluated with different huge datasets independently with no communication among computation tasks (except for computation tasks sending Provisional/Intermediate Results or Transferring Files to the client). If communication/cooperation among tasks is needed, Distributed Communicating Processes module of pycos framework could be used. Dispy works with Python versions 2.7+ and 3.1+ and tested on Linux, OS X, and Windows.

D. Advantages of Distributed computing:

Distributed systems are inherently scalable as they scale horizontally and work across data spread across different machines[23]. This means a user can add more machines to handle the increasing workload instead of having to update a single system over and over again. There is virtually no capacity on how much a user can scale. A system under high demand can run each machine to its full capacity and take machines offline when the workload is low.

Distributed systems are much more cost-effective compared to very large centralized systems. Their initial cost is higher than standalone systems, but only up to a certain point after which they are more about economies of scale[24]. A distributed system made up of many minicomputers can be more cost-effective than a mainframe machine.

VI. SYSTEM DESIGN

We have created a spark Cluster environment with multiple systems with each machine configured with the system properties that can be shown in the Table 1. There is a master system which has a driver node. It controls all other worker machines like resource sharing, assigning tasks, stages and jobs. When we run our Subspace Clustering algorithm on the Driver Node the cluster manager coordinates all these machines and executes the SPARK Program. The data files from other systems are accessed from the Hadoop Distributed File System. Depending on the input of k Value, we get the final resultant ”k” number of medoids from all these Nodes. We can compare the results by changing the number of cores, memory per executor, k, the average number of dimensions (l) values to analyse the cluster performance.

Spark Environment	
Feature	Configuration
OS	Ubuntu 16.04
Programming Language	Python 3.6.1
Spark version	2.4.0
Processor	Intel Quadcore
Memory	8GB
Worker Machines	4
Total Cores	20

TABLE I
SYSTEM CONFIGURATION ON ALL EXECUTOR NODES

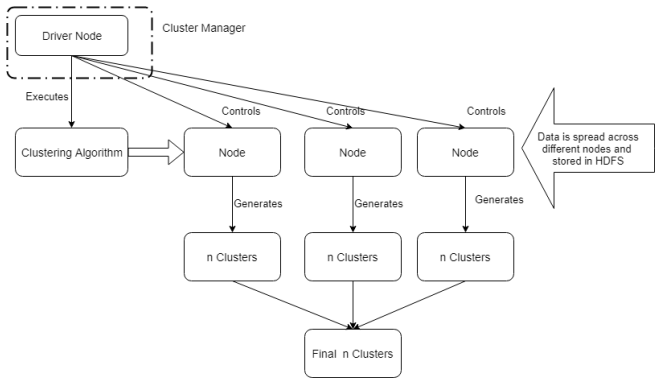

Fig. 2. System Design

VII. ALGORITHMS

While both CLIQUE and PROCLUS aim to discover interesting correlations among data in various subspaces of the original high dimensional space, their output is significantly different. CLIQUE is successful in exploring dense regions in all subspaces of some desired dimensionality. For many applications in customer segmentation and trend analysis, a partition of the points is required[20]. Furthermore, partitions provide clearer interpret ability of the results, as compared to reporting dense regions with very high overlap. In such cases, PROCLUS is preferable to CLIQUE.

A. PROCLUS

Projected clustering seeks to assign each point to a unique cluster, but clusters may exist in different subspaces. The general approach is to use a special distance function together with a regular clustering algorithm. Several distance measures like euclidean and manhattan distance are used here.

PROCLUS uses a similar approach with a k-medoid clustering. Initial medoids are guessed, and for each medoid the subspace spanned by attributes with low variance is determined. Points are assigned to the medoid closest, considering only the subspace of that medoid in determining the distance[21]. The algorithm then proceeds as the regular PAM algorithm.

```
Algorithm Proclus(K = number of clusters ,l = Average Dimensions )
[R is the spark RDD]
[Ci is the ith cluster]
[Di is the set of dimensions associated with cluster Ci]
[Mcurrent is the set of dimensions in current iteration]
[Mbest is the best set of medoids found so far]
[N is the final set of medoids with associated dimensions]
[A,B are constant integers]
begin

1. Initialization Phase
S = random sample of size A . k
M = Greedy(R,S,B.k)

2. Iterative Phase
Mcurrent = Random set of medoids m1,m2, .... mk M
repeat
Approximate the optimal set of dimensions
for each medoid mi Mcurrent do
begin
Let L be distance to nearest medoid from mi
Li = points in sphere center mi with radius
end;
L = L1,.....,Lk
  D1,D2,.....,Dk = FindDimensions(R,k,l,L)
  Form the clusters
  C1,.......,Ck = AssignPoints(D1,....,Dk)
        ObjectiveFunction=EvaluateClusters(C1,.,Ck,D1,.,Dk)
        if ObjectiveFunction ¡ BestObjective then
        begin
              BestObjective = ObjectiveFunction
              Mbest = Mcurrent
        Compute the bad medoids in Mbest
        Compute Mcurrent by replacing the bad medoids
        until (termination criterion)
3. Refinement Phase
  L = C1,.....,Ck
  (D1,D2,.......,Dk) = FindDimensions(R,k,l,L)
  (C1,.....,Ck) = AssignPoints(R,D1,....,Dk)
  O = FindOutliers(C1,.....,Ck)
  N = (Mbest,D1,D2,....,Dk)
  return(N)
  end
```

VIII. IMPLEMENTATION DETAILS

A. PROCLUS

In order to understand the algorithm, we should introduce some notations. Let N denote the total number of data points and D denote the dimensions or number of features in each data. The centroid of a cluster is the algebraic average of all points within the cluster. Various distance measures have been used depending on a particular problem. Manhattan Distance and Euclidean distance measures are used.

B. Initialisation Phase

The initialization phase is geared towards finding a small enough superset of a piercing set, so that it is possible to efficiently perform hill-climbing on it, as opposed to the entire database of points. In this phase, we will use the greedy technique in order to find a good superset of a piercing set of medoids. In other words, if we wish to find k clusters in the data, we will pick a set of points of cardinality a few times larger than the constant k. We will perform two successive steps of subset selection in order to choose our superset.

In the first step, we choose a random sample of data points of size proportional to the number of clusters we wish to generate. We shall denote this sample by S. In the second step, we apply the above- mentioned greedy technique to S in order to obtain an even smaller final set of points on which to apply the hill climbing technique during the next phase. In our algorithm, the final set of points has size B k; where B is a small constant. We shall denote this set by M.

In Spark, we first initialize the data and convert into the resilient distributed dataset (RDD). Then inside the greedy function, we use a mapper for the first step while finding the distances from each random points to all these data points before filtering out the random subset of points to find the Euclidean distance. It is then reduced to a Euclidean distance array. In the second step, this distance array is used to update the distances so that they reflect the distances to the closest medoid. Then we get the final set of medoids M.

C. Iterative Phase

This phase is started by choosing a random set of k medoids from M and progressively improve the quality of medoids by iteratively replacing the bad medoids in the current set with new points from M. The hill-climbing technique can be viewed as a restricted search on the complete graph with vertex set defined by all sets of medoids of cardinality k. The current node in the graph, denoted by Mbest, represents the best set of medoids found so far. The algorithm tries to advance to another node in

```
Algorithm Greedy(RDD, Set of points: S, Number of medoids: K)
{ d(' ,') is the distance function }
begin
M = {m1} { m1 is a random point of S }
{ compute distance between each point and medoid m1}
for each x ∈ S \ M
    dist = d.map(RDD, m1)
for i = 2 to k
begin
    { choose medoid mi to be far from previous medoids }
    Let mi ∈ S \ M be st. dist(mi) = max{ dist(x) | x ∈ S \ M }
    M = M ⋃ {mi}
    { compute distance of each point to closest medoid }
    for each x ∈ S \ M
        dist(x) = min{dist(x), d(RDD, mi)}
end
return M
end
```

Fig. 3. Greedy Algorithm

the graph as follows: It first determines the bad medoids in Mbest using a criterion. Then it replaces the bad medoids with random points from M to obtain another vertex in the graph, denoted Mcurrent. If the clustering determined by Mcurrent is better than the one determined by Mbest , the algorithm sets Mbest = Mcurrent. Otherwise, it sets Mcurrent to another vertex of the graph, obtained by again replacing the bad medoids in Mbest with random points of M. If Mbest does not change after a certain number of vertices have been tried, the hill-climbing phase terminates and Mbest is reported.

Consider k medoids and we need to evaluate the clustering given by these for finding a set of dimensions for each medoid in the set and forming the cluster corresponding to each medoid. For finding dimensions we need to evaluate the locality of the space near them in order to get features that are important.

For each I, we find the Locality Li, which is a set of points that are within the given above distance from mi. This is done by filtering the RDD with the points within each cluster and then mapping those points by computing the average distance along each dimension from the points in Li to mi. Let Xij denote this distance along each dimension, then the mapper is reduced to find the average along each dimension. We then associate those dimensions j for which Xij are small as possible such that the total number of dimensions related to medoids must be equal to k*l and at least 2. We then compute the mean and standard deviation of the RDD.

These dimensions and the medoids which are formed in the previous steps are then mapped into each data points with a key value. For each tuple, the manhattan segmental distance relative to Di between the point and medoid mi is found and the resultant is reduced to find the closest medoid.

Evaluate Cluster function is used to evaluate the clusters we obtain from the assignment of points. To evaluate the quality of these, for each set of medoids the manhattan segmental distance from each point in the cluster to the centroid is found by mapping the RDD for each partition. Then mean() action is performed on all the dimensions of the resultant RDD. Then we get a centroid which might typically differ from the medoid centre. In the next step, we use another mapper on the RDD to find the bad medoids.

For each cluster, medoid of any cluster which is less than [(N/k). min Deviation] are considered to be bad clusters. This type of medoids are usually considered to be outlier medoids, or they can be pierced by at least one other medoids in the data. So we assume that outliers are formed by very few points and two or more set of medoids are likely to form a cluster which contains most of the points that come in that dimension.

D. Refinement Phase

After the best set of medoids is found, we do one more pass over the data to improve the quality of the clustering. The refinement of our RDD is done here with both finding the dimensions for the RDD and reducing them with the best ones, later assigning each row in RDD with the medoids is also done with other mapper and reducer. Let f [C1; C2; .; Ck] be the clusters corresponding to these medoids, formed during the Iterative Phase.

We discard the dimensions associated with each medoid and compute new ones by a procedure similar to that in the previous subsection. The only difference is that in order to analyze the dimensions associated with each medoid, we use the distribution of the points in the clusters at the end of the iterative phase, as opposed to the localities of the medoids. In other words, we use Ci instead of Li. Once the new dimensions have been computed, we reassign the points to the medoids relative to these new sets of dimensions.

IX. Experimental Results

A. Working Environment

We have distributed working environment using multiple machines and a driver node which controls all these machines by a cluster manager. The master system has the driver node. It controls all other worker machines like resource allocation and sharing, assigning tasks, stages and jobs. When we run our Subspace Clustering algorithm on the Driver Node the cluster manager coordinates all these machines and executes the SPARK Program. The spark architecture don't have a distributed file system within its core. So, the data files from other systems are accessed by the distributed platform present in the hadoop called HDFS.

Depending on the input values like number of clusters and average number of dimensions for finding the subspaces and input dataset, the proclus algorithm on spark gives the final resultant "k" number of medoids from all these Nodes. We can compare the results by changing the number of cores, memory per executor, k, the average number of dimensions (l) values to analyse the cluster performance.

Purity of the clusters is also found by finding the maximum number of labelled points in each cluster. It can be calculated by Sum of Total number of points for data with maximum labels in each of the medoid divided by total number of points in the whole dataset. The clustering is well performed if the purity of the clustering is high.

We did some preliminary experiments on various datasets mentioned in Table II. We ran PROCLUS on these datasets to find the F1-Value and purity of the clusters formed in subspaces, the results can be observed in Table III and Table V. Next to implement the algorithm we need to choose number of cores for each node. We used SYNTHETIC4 dataset and Bcell dataset to run all the experiments. Here Time1 denotes execution time for synthetic dataset and Time2 denotes time for Bcell dataset. We varied the number of cores per worker node to find how the algorithm performs as seen in Table IV. We also ran the PROCLUS on incremental number of nodes and found that increasing number of nodes can drastically reduce the execution time for the completion of the algorithm as seen in Table VI.

Dataset Description			
DATASET	Dimensions	Records	Labels
LEUKEMIA1	7130	34	2
LEUKEMIA2	7130	38	2
BCELL1	4027	45	2
EMBROYONAL	7130	60	2
COLON	2001	62	2
ECML	27680	90	43
BCELL3	4027	96	9
BCELL2	4027	96	11
GCM	16064	144	14
GLASS	10	214	6
DIABETES	9	768	2
WAVEFORM	41	5000	3
PENDIGITS	17	8000	10
GERMAN	25	8000	2
SYNTHETIC	75	10000	13
BANK	63	41189	2
SYNTHETIC2	6	10000	12
SYNTHETIC3	6	100000	12
SYNTHETIC4	6	1000000	12

TABLE II

DATASET DESCRIPTION

For the experimentation part, We changed the number of clusters(k) and Average number of dimensions(l) in the subspaces to find the execution time for the million row datasets. We observed that as the number of clusters increases the execution time also increases as seen in Table VIII. The same can be said for the increase in average number of dimensions for each subspace, as l value increases the time increases to run the algorithm as shown in Table VII. We also found the results of the F-value and purity values on the spark environment and compared these values with the normal implementation on single machine. These experiment results show that the performance or the predictability of finding the exact clusters groups as they are labeled is more precise in spark. The results show high clustering purity on distributed spark platform as seen in Table IX and Table X.

F1-value					
Dataset	K	Kmeans	l	Proclus	Clique
B-Cell2	k=11	o.6839	l=1007	0.4544	0.3583
B-Cell3	k=9	o.7620	l=1007	0.6047	0.4869
leukemia1	k=2	o.5405	l=1783	0.5594	0.4792
leukemia2	k=2	o.7092	l=1783	0.4356	0.6204
Embryonal	k=2	o.5298	l=1783	0.5210	0.5037
GCM	k=14	o.4137	l=4016	0.5079	–
ECML	k=43	o.6848	–	–	–
Synthetic	k=13	o.9436	–	–	–
Pen Digits	k=10	o.7553	l=4	0.4592	0.3697
German	k=2	o.5312	l= 6	0.5296	0.48522
WaveForm	k=3	o.5247	l= 10	0.5442	0.4479

TABLE III

RESULTS OF F1-VALUE OF DIFFERENT DATASETS

Changing Cores/Worker nodes	
Cores	Time
1	1.4hr
2	1.3hr
3	57 min
4	56 min

TABLE IV

PERFORMANCE OF PROCLUS ON SPARK FOR SYNTHETIC DATASETS

Purity					
Dataset	K value	Kmeans	l value	Proclus	Clique
B-Cell2	k=11	o.8229	l=1007	0.5833	0.2633
B-Cell3	k=9	o.9166	l=1007	0.6145	0.4792
leukemia1	k=2	o.5882	l=1783	0.6176	0.5984
leukemia2	k=2	o.8157	l=1783	0.5526	0.7545
Embryonal	k=2	o.6500	l=1783	0.6166	0.65
GCM	k=14	o.5069	l=4016	0.2569	–
ECML	k=43	o.7222	NA	–	–
Synthetic	k=13	o.9949	NA	–	0.2077
Pen Digits	k=10	o.7622	l=4	0.4896	0.19315
German	k=2	o.7047	l= 6	0.6935	0.7052
WaveForm	k=3	o.5246	l= 10	0.4683	0.3810

TABLE V

RESULTS OF PURITY OF DIFFERENT DATASETS

Nodes vs Runtime			
Nodes	Synthetic2	Synthetic3	Synthetic4
1	1.4 hrs	2hrs+	2hrs+
2	35 mins	1.1hr	1hr+
3	3.4 mins	26mins	41mins
4	2.7 mins	14mins	27mins
5	1.5 mins	5.9mins	15mins

TABLE VI

PERFORMANCE OF PROCLUS ON SPARK FOR SYNTHETIC DATASETS

Changing Avg. no of Dimensions				
	Dataset1	Dataset1	Dataset2	Dataset2
k(Cluster)	Avg. Dimension	Time1	Avg. Dimension	Time2
3	2	1.1hr	500	3.2 min
3	3	56 min	1000	3.3 min
3	4	1.2hr	1500	3.2 min
3	5	1.4hr	2000	3.6 min
3	6	1.5hr	2500	3.9 min

TABLE VII

PERFORMANCE OF PROCLUS ON SPARK FOR SYNTHETIC DATASETS

Changing Number of Clusters(k)			
k(Cluster)	Time1	k(Cluster)	Time2
2	56min	2	3.2 min
4	1.3hr	3	3.5 min
6	1.5hr	4	4.2 min
8	1.7hr	5	5.0 min
10	1.8hr	6	5.1 min

TABLE VIII

PERFORMANCE OF PROCLUS ON SPARK FOR SYNTHETIC DATASETS

So the performance of forming the clusters is improved in spark environment.

Comparison of F-value of clusters		
Dataset	On Single machine	On Spark
Bcell1	0.5613	0.6007
Bcell2	0.4544	0.5964
Bcell3	0.3047	0.5738
Leukemia1	0.5407	0.5987
Leukemia2	0.4356	0.6445
Pendigits	0.4542	0.6168
Diabetes	0.6562	0.7346
GCM	0.5079	0.5372

TABLE IX

COMPARING F-VALUE PERFORMANCE OF PROCLUS ON SPARK AND SPARK ON SINGLE MACHINE

Comparison of Purity of clusters		
Dataset	On Single machine	On Spark
Bcell1	0.5833	0.6841
Bcell2	0.6251	0.612
Bcell3	0.6149	0.8269
Leukemia1	0.6823	0.7681
Leukemia2	0.5526	0.7258
Pendigits	0.4896	0.5662
Diabetes	0.7244	0.6736
GCM	0.2569	0.5924

TABLE X

COMPARING PURITY PERFORMANCE OF PROCLUS ON SPARK AND SPARK ON SINGLE MACHINE

X. CONCLUSION

In this paper, we present a PROCLUS algorithm on top of Spark. We carried out some preliminary tests on a modest cluster showing promising results with respect to scalability. PROCLUS on spark performs better than that of PROCLUS algorithm which can be seen through Clustering Metrics like purity and F-value. The obtained results show that PROCLUS on Spark efficiently finds correct subspace clusters and scales better than traditional PROCLUS algorithm.

However, future work is needed to compare the scalability and performance of PROCLUS on spark with other subspace algorithms like CLIQUE on spark platform.

REFERENCES

[1] Singh Vijendra , 2011. Efficient Clustering for High Dimensional Data: Subspace Based Clustering and Density Based Clustering. Information Technology Journal, 10: 1092-1105.

[2] Masih, Shraddha, and Sanjay Tanwani. "Data mining techniques in parallel and distributed environment-a comprehensive survey." International Journal of Emerging Technology and Advanced Engineering (March 2014) 4.3 (2014): 453-461.

[3] Zhu, Bo, Alexandru Mara, and Alberto Mozo. "CLUS: parallel subspace clustering algorithm on spark." East European Conference on Advances in Databases and Information Systems. Springer, Cham, 2015.

[4] Zimek, Arthur, Ira Assent, and Jilles Vreeken. "Frequent pattern mining algorithms for data clustering." Frequent Pattern Mining. Springer, Cham, 2014. 403-423.

[5] Dean, J., Ghemawat, S.: MapReduce: simplied data In Proc. on large clusters. Communications of the ACM 51(1), 107113 (2008)

[6] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[7] Kaur, Mantripatjit, and G. Dhaliwal. "Performance comparison of map reduce and Apache Spark." International Journal of Computer Sciences and Engineering 3.11 (2015).

[8] Ryza, Sandy, et al. Advanced analytics with spark: patterns for learning from data at scale. " O'Reilly Media, Inc.", 2017.

[9] Pemmasani, Giridhar. "dispy: Python Framework for Distributed and Parallel Computing." (2013).

[10] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." HotCloud 10.10-10 (2010): 95.

[11] Iqbal, Muhammad Hussain, and Tariq Rahim Soomro. "Big data analysis: Apache storm perspective." International journal of computer trends and technology 19.1 (2015): 9-14.

[12] Sparks, Evan R., et al. "MLI: An API for distributed machine learning." 2013 IEEE 13th International Conference on Data Mining. IEEE, 2013.

[13] Jiang, Yaobin, and Jiongmin Zhang. "Parallel K-Medoids clustering algorithm based on Hadoop." 2014 IEEE 5th International Conference on Software Engineering and Service Science. IEEE, 2014.

[14] Solaimani, Mohiuddin, et al. "Spark-based anomaly detection over multi-source VMware performance data in real-time." 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS). IEEE, 2014.

[15] Cordova, Irving, and Teng-Sheng Moh. "Dbscan on resilient distributed datasets." 2015 International Conference on High Performance Computing Simulation (HPCS). IEEE, 2015.

[16] Kraska, Tim, et al. "MLbase: A Distributed Machine-learning System." Cidr. Vol. 1. 2013.

[17] Jin, Ruoming, Anjan Goswami, and Gagan Agrawal. "Fast and exact out-of-core and distributed k-means clustering." Knowledge and Information Systems 10.1 (2006): 17-40.

[18] Agrawal, Rakesh, et al. Automatic subspace clustering of high dimensional data for data mining applications. Vol. 27. No. 2. ACM, 1998.

[19] Cheng, Chun-Hung, Ada Wai-Chee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. Diss. Chinese University of Hong Kong, 1999.

[20] Aggarwal, Charu C., et al. "Fast algorithms for projected clustering." ACM SIGMoD Record. Vol. 28. No. 2. ACM, 1999.

[21] Firdaus, Sabhia, and Md Ashraf Uddin. "A survey on clustering algorithms and complexity analysis." International Journal of Computer Science Issues (IJCSI) 12.2 (2015): 62.

[22] Chen, Min, Simone A. Ludwig, and Keqin Li. ”Clustering in Big Data.” Big Data Management and Processing. Chapman and Hall/CRC, 2017. 333-346.

[23] Kalousis, Alexandros, Julien Prados, and Melanie Hilario. ”Stability of feature selection algorithms: a study on high-dimensional spaces.” Knowledge and information systems12.1 (2007): 95-116.

[24] Yan, Su, et al. ”Semi-supervised dimensionality reduction for analyzing high-dimensional data with constraints.” Neurocomputing 76.1 (2012): 114-124.