The International Journal of Parallel, Emergent and Distributed Systems
Vol. 00, No. 00, Month 200x, 1–21

RESEARCH ARTICLE

Parallel and Distributed Clustering Framework for Big Spatial Data Mining

Malika Bendechache[a*], A-Kamel Tari[b] and M-Tahar Kechadi[a]

[a] *Insight Centre for Data Analytics, University College Dublin, Ireland*
[b] *University A-Mira of Bejaia, Algeria*

(*v3.6 released March 2018*)

Clustering techniques are very attractive for identifying and extracting patterns of interests from datasets. However, their application to very large spatial datasets presents numerous challenges such as high-dimensionality, heterogeneity, and high complexity of some algorithms. Distributed clustering techniques constitute a very good alternative to the Big Data challenges (e.g., Volume, Variety, Veracity, and Velocity). In this paper, we developed and implemented a Dynamic Parallel and Distributed clustering (DPDC) approach that can analyse Big Data within a reasonable response time and produce accurate results, by using existing and current computing and storage infrastructure, such as cloud computing. The DPDC approach consists of two phases. The first phase is fully parallel and it generates local clusters and the second phase aggregates the local results to obtain global clusters. The aggregation phase is designed in such a way that the final clusters are compact and accurate while the overall process is efficient in time and memory allocation. DPDC was thoroughly tested and compared to well-known clustering algorithms BIRCH and CURE. The results show that the approach not only produces high-quality results but also scales up very well by taking advantage of the Hadoop MapReduce paradigm or any distributed system.

Keywords: Big Data, MapReduce, Hadoop, Spatial data mining, Clustering, Distributed Clustering, Parallel clustering, DBSCAN, Dynamic K-means,

*Corresponding author. Email: malika.bendechache@gmail.com

ISSN: 1023-6198 print/ISSN 1563-5120 online
© 200x Taylor & Francis
DOI: 10.1080/1023619YYxxxxxxxx
http://www.informaworld.com

1. Introduction

The amount of data generated per year will reach more than 44 zettabytes (44 billion terabytes) in 2020, ten times more than in 2003 and this is likely to continue according to IDC survey [1]. This means more than 10 terabytes per person and per year of data were generated by the daily life. Big Data refers to very large datasets that are collected from different fields and continue to grow at rapid pace. Analysing and extracting relevant information from these datasets is one of the biggest challenges due to their needs to huge storage capacity, processing power, and efficient mining algorithms to deal not only with the size but also with heterogeneity, and noise. These require modifications in the data storage and in the data management, as well as the development of new algorithms to efficiently mine big data. In fact, the analysis of big data requires powerful, scalable, and accurate data analytics techniques that the traditional data mining and machine learning do not provide, as they cannot deal with Big data challenges (volume, velocity, veracity, variety) all at the same time. Distributed data mining constitutes a promising approach for big data analytics, as datasets are usually produced in distributed locations, and processing them on their local sites will reduce significantly the response time.

The Distributed Data Mining (DDM) is a line of research that has attracted much interest in recent years [2–6]. DDM was developed because of the need to process datasets which can be very large or geographically distributed across multiple sites [7]. This has two advantages: first, distributed systems have enough power to analyse the data within a reasonable response time frame. Second, it would be very advantageous to process data on their respective sites, this will allow to avoid the transfer of large volumes of data to a central site; to avoid heavy communications, network bottlenecks, to be able to process heterogeneous datasets, etc. Distributed and parallel data mining techniques can be divided into two categories based on the targeted architectures of computing platforms. One is based on the data parallel paradigm and uses traditional dedicated and parallel machines with tools for communications between processors. These machines are generally called super-computers and are very expensive. The second category targets a network of autonomous machines and is characterised by slow communications and heterogeneity of the machines (different OSs, different architecture, etc.), but they are very abundant and inexpensive [8]. The main goal of the second category of techniques is to distribute the work among the system nodes and try to minimise the response time of the whole application. Some of these techniques have already been developed and implemented in [9, 10].

However, the traditional DDM methods are not always effective, as they suffer from the problem of scaling. This has led to the development of techniques that rely on ensemble learning [11, 12]. The main idea is to produce a local model in each processing node on its own data and then try to generate global models by aggregating local models. In this approach, the first phase is simple enough but the aggregation phase is very complex. Some algorithms of this class generate significant communication overheads which usually cancel the benefit made in the first phase. With recent advances in cloud computing, both in hardware and software, the ensemble learning based distributed data mining methods are gaining in popularity and are very promising. The techniques that adopted MapReduce parallel programming paradigm are even more scalable and more efficient in terms of reducing the communication overheads.

Apache Hadoop becomes one of the most popular parallel and distributed processing model for big data. MapReduce, the heart of Apache Hadoop, is the programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. It is capable of processing large-scale datasets by exploiting the parallelism among clusters of processing nodes. MapReduce gained popularity for its simplicity, flexibility, fault tolerance, and scalability soon after its birth. Many data mining algorithms were implemented in Hadoop MapReduce to improve and accelerate their performance [13–15]. Especially, many researchers have proposed MapReduce implementations for big data clustering [16–19]. These usually consist of MapReduce implementations of the existing clustering algorithms for some given applications.

In this paper, we discuss the implementation of a dynamic parallel and distributed clustering (DPDC) approach using Map-Reduce programming paradigm. The first version of this approach, presented in [20–22], was specifically developed to deal with some issues of some traditional algorithms, such as K-Means. For instance, when the K-means was used as the local clustering algorithm, the whole approach acts as a k-means algorithm that is executed on distributed data with dynamic K calculation. This means that this distributed version of the K-means does not need to give the exact k as an input, which is really huge improvement compared to the traditional version the algorithm. The main objective of the work that is presented in this paper is to study and show that this approach is general and flexible to run on different parallel distributed environments. Implementing the PDC approach using the MapReduce paradigm will allow us to study its behaviour and its capabilities in dealing with very large and heterogeneous datasets. In addition, one wants to study the performance and scalability of the proposed approach without losing the quality of the final clustering.

The rest of the paper is organised as follows: In the next section we will give an overview of the state-of-the-art for distributed data mining and highlight the limitations of traditional techniques. Then we will introduce the MapReduce paradigm and the Hadoop Framework. We present our distributed clustering framework and its concepts in Section 4. In Section 5, we evaluate the performance of the distributed approach in terms of quality of clustering and compare its results to two well-known clustering algorithm; BIRCH and CURE. We also study the scalability of the approach implemented with MapReduce and see how does its performance behave to the size of the dataset. Finally, we conclude in Section 6.

2. Related Work

Clustering algorithms can be divided into two main categories, namely partitioning and hierarchical. Different clustering algorithms are proposed and studied in the literature [23–27]. Many parallel versions based on these algorithms have been proposed in the literature [27–33]. These algorithms are classified into two subcategories. The first consists of methods requiring multiple rounds of message-passing. They require a significant amount of synchronisations. The second subcategory consists of methods that build local clustering models and send them to a central site to build global clusters [34]. In [29] and [32], message-passing versions of the widely used K-means algorithm were proposed. In [27] and [33], the authors dealt with the parallelisation of the DBSCAN density-based clustering algorithm. In [30] a parallel message passing version of the BIRCH algorithm was presented. A parallel version of a hierarchical clustering algorithm called MPC for Message Passing Clustering, which is especially dedicated to Microarray data was introduced in [31]. Most of the parallel approaches need either multiple synchronisation constraints between processes or a global view of the dataset or both [28]. Another approach presented in [28] also applied a merging of local clusters to create the global clusters. Current approaches only focus on either merging local models or mining a set of local models to build global ones. If the local models cannot effectively represent local datasets then global models accuracy will be very poor [34]. Other researchers focus on optimising the merging (aggregation) phase. For instance, Le Khac et al. [35] and Laloux et al. [34] proposed an efficient aggregation phase that minimises the communications and the amount of data exchanged between the nodes. Many data reduction techniques have been proposed with the objective of reducing the communications and data exchange, which introduce significant overheads. Some of these techniques use Sampling methods [36, 37]. Sampling is the simplest approach for data reduction. The idea is to draw the desired number of random samples from the entire dataset and perform the clustering on those samples. However, naive sampling is more likely to produce poor results mainly on real world problems with noisy data. Another data reduction technique is discretisation [38]. Data discretisation can be used to reduce the number of values for continuous attributes by dividing the range of each attribute into intervals. Interval labels can then be used to replace actual data values. However, this technique requires a good interval selection process. This reduces significantly the size of the original data and therefore its complexity. However, both sampling and discretisation can be used for centralised clustering techniques to reduce their complexity, that are not specific to distributed datasets.

3. MapReduce and Hadoop Framework

Hadoop is an open-source framework designed for distributed processing of large datasets across a large and dynamic network of computers. This framework can process data in keeping with high volume, velocity, and variety. It transforms commodity computing hardware into a service that stores data and facilitates the development of distributed applications. Hadoop handles hardware failures at the application layer. The main features of Hadoop are:

(1) **Scalability:** a program that runs on a single machine can also run on thousand machines. If more power is needed, we just add more machines.
(2) **Fault tolerance:** a key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, the same data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use.
(3) **Fast:** Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located in a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. In case of dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and petabytes in few hours.
(4) **Simplicity:** Hadoop provides a lot of simple API's and is very powerful. It can deal with huge data (petabytes).

Hadoop is composed of two main components: MapReduce and HDFS. The first one is the processing part and the second is the data part. Several machines with Hadoop create a cluster. A cluster can be composed of hundreds or thousands of machines. Instead of processing data in a sequential way, Hadoop splits files into blocks that can be processed simultaneously.

3.1 HDFS

HDFS (Hadoop Distributed File System) is a file system that runs on top of the existing file system of each node. This file system works best with large files and uses the concept of 'blocks' to store files. These blocks are of a fixed size (64MB by default). The concept of blocks has several advantages. The fixed size makes it easier to compute how many blocks can be stored in a disk. Data blocks can be stored on one particular node. Blocks are also replicated on multiple nodes to ensure access to the data. There are two particular HDFS nodes: NameNode and DataNode. There is only one NameNode per cluster, it is responsible for the metadata of the files and for the file-system namespace. It should be the most powerful node of the cluster and it should have as much RAM as possible because it keeps the entire file-system metadata in memory. If the NameNode is lost, all the data of the cluster are also lost. In order to prevent this, a replication of the NameNode, called 'StandByNameNode', can be defined. In a cluster, there can be many nodes called 'DataNodes'. These nodes store blocks of data and when the clients want to retrieve some information, they have to query the NameNode to get which DataNodes store their data. Periodically, DataNodes send information about blocks that they store. DataNodes are the lowest layer of the cluster. When a new file arrives in the system, a 'create request' is sent to the NameNode. It chooses the DataNode where the blocks will be written and replicates the data on other nodes (See Figure 1).

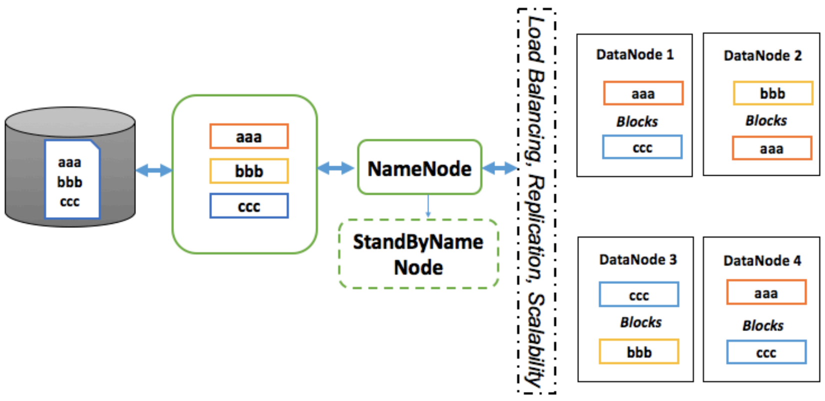

Figure 1.: HDFS architecture.

3.2 MapReduce Paradigm

MapReduce is a programming paradigm for data intensive applications, its main objective is to easily develop applications which process large amounts of data (multi-terabyte datasets) in parallel on large clusters (thousands of nodes) of commodity hardware in a reliable and fault-tolerant manner. A MapReduce program is composed of two basics components: Map task and Reduce task. A MapReduce program takes its inputs in a form of (Key, Value) pairs, which are processed by the map tasks in a completely parallel manner [39].

As it is shown in Figure 2, a job in MapReduce contains three phases: Map, Shuffle and Reduce. In most cases, the user only needs to write the map function and the reduce function. The map phase, for each input pair (k_1, v_1), the map function generates one or more output pairs list (k_2, v_2). In shuffle phase, the output pairs are partitioned and then transferred to reducers. In reduce phase, pairs with the same key are grouped together as $(k_2, list(v_2))$. In reduce phase, reduce function generates the final output pairs $list(k_3, v_3)$ for each group. The MapReduce process is summarised in the following.

The inputs and the outputs of a job are stored in the HDFS. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the HDFS are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already resides, resulting in very high aggregate bandwidth across the cluster [39] (see Figure 2).

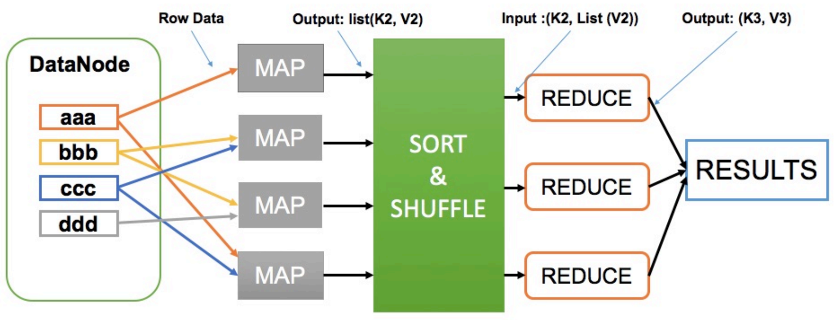

Figure 2.: MapReduce model.

The MapReduce framework consists of a single master JobTracker and multiple slaves TaskTrackers. Each node connected to the network has the right to behave as a slave TaskTracker. The master receives jobs from the client and schedules the map and reduce tasks on taskTrackers, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master (see Figure 3).

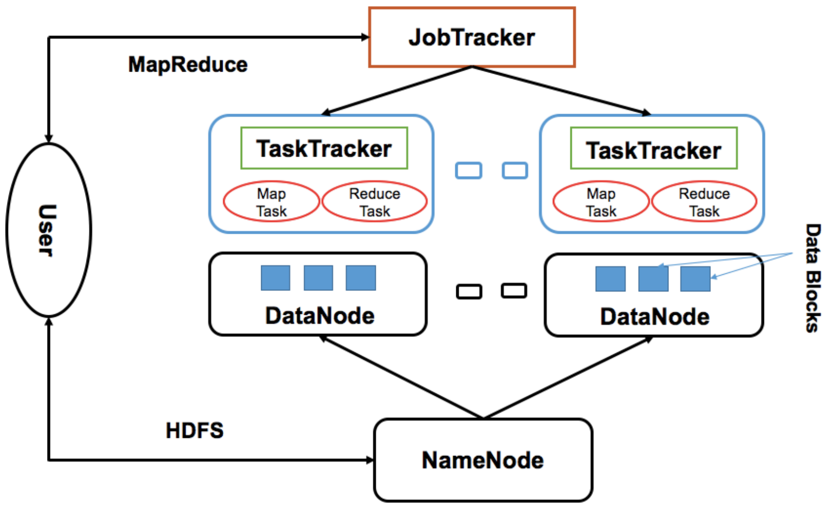

Figure 3.: Hadoop framework.

4. Dynamic Parallel and Distributed Clustering

Dynamic Parallel and Distributed Clustering (DPDC) model is introduced to deal with the limitations of the existing parallel and distributed clustering models while dealing effectively with big data challenges. DPDC combines the characteristics of both partitioning and hierarchical clustering methods, and more importantly, it does neither inherit the problem of the number of partitions to be fixed in advance (partitioning clustering) nor the problem of stopping conditions of hierarchical clustering. The number of final clusters is calculated dynamically and generates them in a hierarchical way. All these features look very promising and some of them have been studied in [20, 21, 40], and it is evaluated at a small scale [22].

In this study, we focus large scale implementation of DPDC using MapReduce paradigm. The main objective is to show that the DPDC approach can take advantage of a flexible parallel and distributed environment, such as Map-Reduce running on cloud computing. Implementing DPDC using MapReduce paradigm will allow to study its behaviour and capabilities on very large datasets without losing in the quality of the final results. We start by briefly explaining the approach and then present the implementation of the approach using MapReduce programming model.

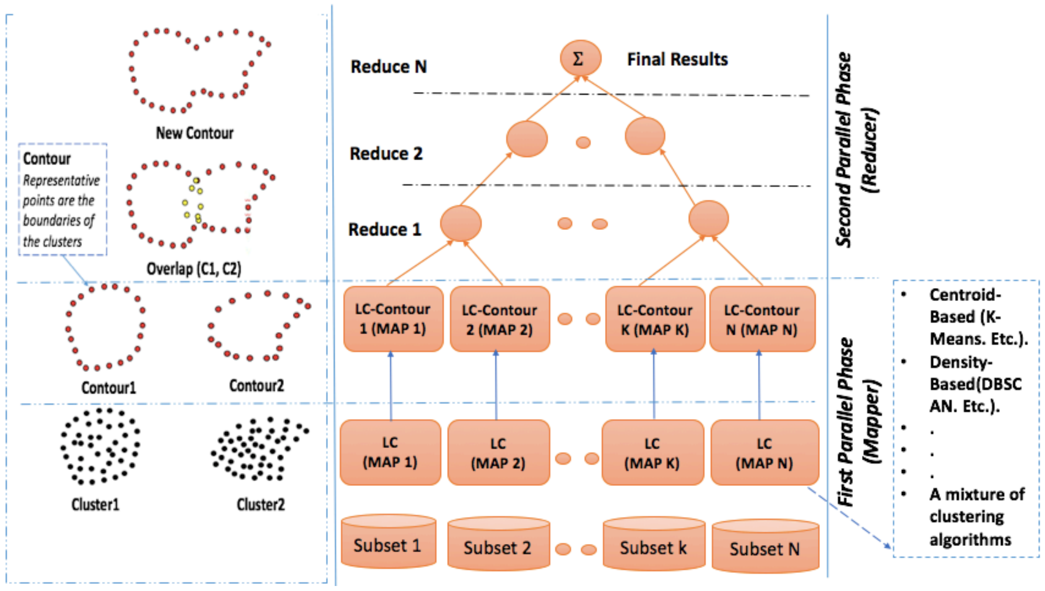

Figure 4.: An overview of the DPDC Approach.

The distributed dynamic clustering approach consists of two main phases, (i) The local model, where we generate local clustering, and (ii) the global model, where we merge the overlapping local clusters to generate global clusters. However, exchanging the local clusters the network nodes will create significant overheads and slowdown hugely the process. This is one of the major problems of the majority of distributed clustering techniques. To deal with this problem, we propose to exchange between nodes a minimum number of points. Instead of sending all the clusters data points, we only exchange their representative points, which constitute 1% to 2% of the total size of the dataset. The best way to represent a spatial cluster is by its shape and density. The shape of a cluster is represented by its boundary points (called contour) (see Figure 4). Many algorithms for extracting the boundaries from a cluster can be found in the literature [41, 42]. The α shape algorithm is based on triangulation to generate the clusters' boundaries [43]. It is an efficient algorithm for constructing non-convex boundaries. It is able to accurately characterise the shape of a wide range of different point distributions and densities with a reasonable complexity of $\mathcal{O}(n \log n)$.

The DPDC approach using MapReduce has also two phases, (i) the mapping phase and (ii) the reducing phase. The mappers perform the clustering algorithm and then they apply the contour algorithm to extract representative points from the local clusters. The second phase consists of multiple reducers to perform the merging between the contours of the overlapping local clusters produced by different processing nodes during the previous phase (mappers). Both mapping and reducing phases are done in parallel. The mappers generate the clusters locally in each node and calculate their contours in parallel without any communications. The reducers perform the merging of the contours in a hierarchical way. Each level of the hierarchy is executed in parallel. Note that at the end of the mapping phase and each level of the hierarchy the involved processing nodes exchange the contours of their clusters. The size of the data exchanged is very small, which leads to efficient communications

DPDC processes the data in a parallel and distributed fashion, while minimising the communications between the nodes. In the present implementation, the HDFS allocates the data randomly to different nodes. The parallel and distributed features of the algorithm are managed by the Hadoop system itself. The mappers are completely independent. Each mapper can implement a different clustering algorithm to mine its local dataset. The reducing phase starts as soon as the contours have been received.

4.1 Mapping Phase

The initial step of the mapper is to acquire the dataset that is allocated to its processing node by interrogating the file system. This is processed and stored in the HDFS files. The format of the data is standardised by the function ReadHDFS-File and the function StandardiseData. The function *ReadHDFSFile()* requires the specification of the pattern of the file. This information is provided by the '`line id`' and the function `StandardiseData` normalises the data.

The main step of the mapper is the execution of the clustering algorithm. Each mapper runs a DBSCAN algorithm with its specific parameters $(Eps, MinPts)$. After generating the local clusters, each node performs the contour algorithm on its local clusters. The mapper allocates the keys to the generated contours. The key is chosen to be the same for all the mappers $(key = 2)$ to simplify the reducing phase. The last step of the mapping phase is to write the result into the HDFS file to be read by the reducer in the next phase. The contours are written in a standardised format, every line represents a contour and has two columns: The first is the contour's key and the second is for the coordinates of the points. The pseudo-code for the mapping phase is given in Algorithm 1.

Algorithm 1 Mapping function.

```
Input  : $X_i$:List<lineNo, Point>, $Eps_i$:$\mathbb{R}$, $MinPts_i$:$\mathbb{N}$
Output: List<key, List<Contour>>
<key, List<Contour>>
$L_i \leftarrow$ DBSCAN($X_i$, $Eps_i$, $MinPts_i$); // Local clusters generated by $Node_i$
$C_i \leftarrow [\,]$;
// For each cluster
foreach $L_i^k \in L_i$ do
    $c^k \leftarrow$ ComputeContour($L_i^k$);
    Append($C_i$, $c^k$) // add contour to the list
end
return <$key$, $C_i$ >;
```

4.2 Reducing Phase

The input of the reduce phase is the standardised output from the mapper phase, which consists of pairs of (key, contour) of the local clusters. The main step of the reducer is the choice of the merging strategy. We set the key to be the same for all the mappers in the mapping phase. Therefore, each contour of each node is compared to all the contours of all others nodes. If two contours overlap, then they will be merged to generate a new contour. The reducer is responsible for merging the overlapping contours and aims to build iteratively bigger clusters until it reaches the final result.

We consider that a contour c^j overlaps with another contour c^k if and only if the polytope $(\mathcal{P}(c^j))$ that is obtained from the points making c^j intersects the polytope $(\mathcal{P}(c^k))$ that is obtained from the points making c^k. A polytope is a polygon in N-dimensional space.

The merging of two contours c^j and c^k yields another contour c^{jk} which is the union of the two polytopes forming c^j and c^k.

The reducing phase is done in a parallel and distributed manner. It is implemented using a tree structure. At each level of the hierarchy half of the nodes will send their contours to their neighbours. They run the merging algorithm on clusters and so on. Hadoop takes care of the communications management and control. The algorithm of the reducing phase is given in Algorithm 2.

Algorithm 2 Reducing function.

```
Input  : $\mathcal{C}$:List<key, List<Contour>>
Output: <key, List<Contour>>
$C_{res} \leftarrow [\,]$;
foreach <key, $C_i$ >$\in \mathcal{C}$ do
    foreach $c^j \in C_i$ do
        merged ← False;
        foreach $c^k \in C_{res}$ do
            if Overlap($c^j$, $c^k$) then
                $c^{jk} \leftarrow$ Merge($c^k$, $c^j$);
                Remove($C_{res}$, $c^k$); // remove contour from the list
                Append($C_{res}$, $c^{jk}$); // add contour to the list
                merged ← True;
            end
        end
        if not(merged) then
            Append($C_{res}$, $c^j$); // add contour to the list
        end
    end
end
return <key, $C_{res}$ >;
```

4.3 DPDC: Example of Execution

In this section, we give an example of execution to show how does the approach work. Figure 5 shows the mapping and the reducing phases with two mappers (2 nodes), each node runs a DBSCAN algorithm with its local parameters $Eps = 15.5$ and $MinPts = 20$. Once the local clusters are generated this will be followed by the execution of the contour algorithm. In this case, we use the α shape algorithm to generate the clusters' boundaries. Figure 5 shows also the reduce phase, which takes the results of mapping as input, and merges the overlapping contours to generate larger clusters.

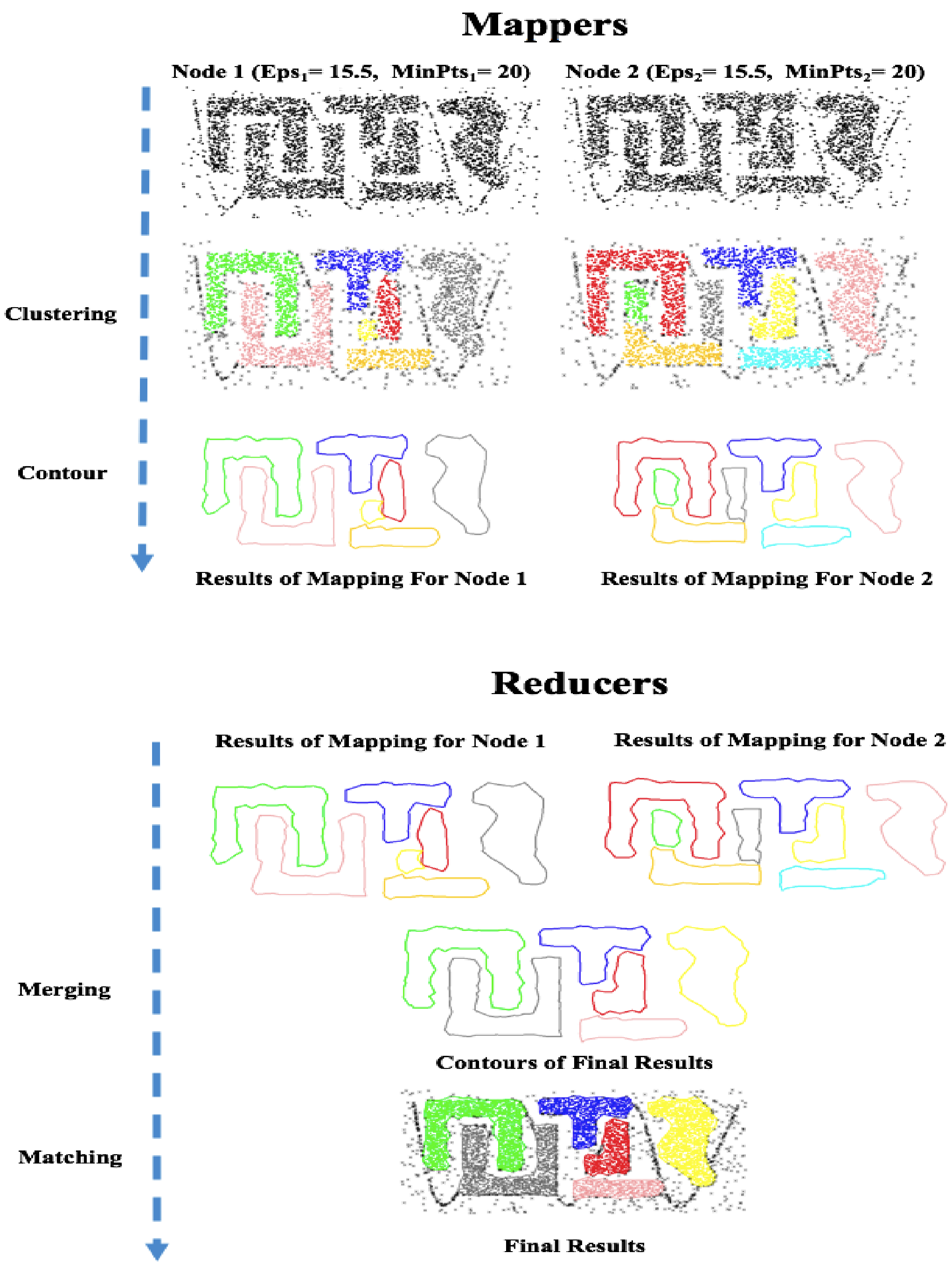
Mappers
Node 1 (Eps_1= 15.5, $MinPts_1$= 20)
Node 2 (Eps_2= 15.5, $MinPts_2$= 20)
Clustering
Contour
Results of Mapping For Node 1
Results of Mapping For Node 2
Reducers
Results of Mapping for Node 1
Results of Mapping for Node 2
Merging
Contours of Final Results
Matching
Final Results

Figure 5.: The Mapping and reducing phases (with 2 nodes).

We also take another example of execution with five mappers (5 Nodes) to see how do the results are affected with the increase of the number of mappers. Figures 6 shows that increasing the number of mappers does not affect the quality of the final clustering.

As can be seen from Figures 5 and 6, the DPDC approach returned exactly the correct number of clusters and their shapes. The approach is insensitive to the way the original data was distributed among the nodes. It is also insensitive to noise

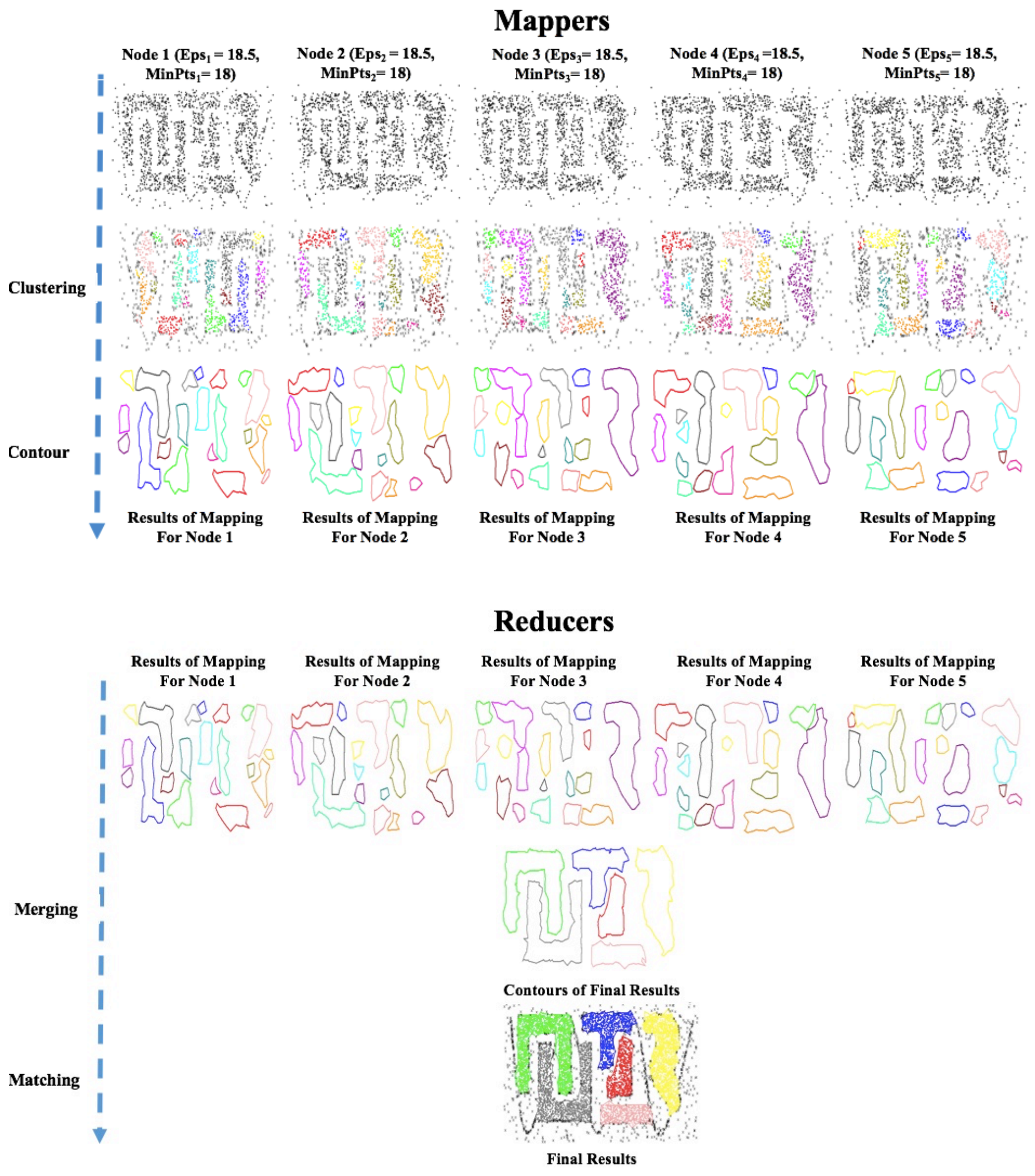
Mappers
Node 1 (Eps$_1$ = 18.5, MinPts$_1$= 18)
Node 2 (Eps$_2$ = 18.5, MinPts$_2$= 18)
Node 3 (Eps$_3$= 18.5, MinPts$_3$= 18)
Node 4 (Eps$_4$ =18.5, MinPts$_4$= 18)
Node 5 (Eps$_5$= 18.5, MinPts$_5$= 18)
Clustering
Contour
Results of Mapping For Node 1
Results of Mapping For Node 2
Results of Mapping For Node 3
Results of Mapping For Node 4
Results of Mapping For Node 5
Reducers
Results of Mapping For Node 1
Results of Mapping For Node 2
Results of Mapping For Node 3
Results of Mapping For Node 4
Results of Mapping For Node 5
Merging
Contours of Final Results
Matching
Final Results

Figure 6.: The reducing phase (5 nodes).

and outliers. As we can see, although each node executed DBSCAN locally with different parameters, the global clusters were generated dynamically even when performed on the noisy datasets.

5. Experimental Results

In this section, we study the performance of the DPDC approach and demonstrate its effectiveness compared to BIRCH, CURE algorithms. We choose these algorithms for the following reasons; they are both very well suited for clustering spatial datasets, BIRCH belongs to the same category as DPDC (hierarchical clustering), and CURE has an efficient optimisation approach.

BIRCH: We use the BIRCH implementation provided in [44]. It performs a pre-clustering and then uses a centroid-based hierarchical clustering algorithm. Note that the computational complexity and memory space required of this approach are quadratic to the number of points after pre-clustering. We set its parameters to the default values suggested in [44].

CURE: We use the CURE implementation provided in [45]. CURE uses representative points with shrinking towards the mean. As described in [45], when two clusters are merged, the representative points of the new merged cluster are selected from the ones of the two original clusters rather than all the points in the merged clusters.

DPDC: We run our experiments using 2 mappers (2 nodes) in the system. Table 1 shows the DBSCAN parameters *Eps* and *MinPts* set for the different datasets used. Note that the local parameters for DBSCAN do not have to be set to the optimal values, they only need to be close to the optimal values because very high accuracy is not required st this stage. Nevertheless, it is indeed better to set *Eps* and *MinPts* as close as possible to the optimal values in order to reduce the processing time in calculating the contours and also merging procedure.

Table 1.: DPDC Configuration (setting)

	Eps	*MinPts*
T1	35	10
T2	35	10
T3	35	10
T4	15.5	20
T5	18.5	18
T6	13.5	5
T7	5.6	9
T8	10	5

5.1 Datasets

We use eight benchmark datasets which are detailed in Table 2. The datasets are of different shapes and sizes. The first three datasets (T_1, T_2, and T_3) contain patterns of convex shapes and the remaining five datasets (T_4, T_5, T_6, T_7, and T_8) contain noise and patterns having non-convex shapes. They are very well-known benchmarks for evaluating density-based clustering algorithms. The number of points and clusters in each dataset is also given. These eight datasets contain a set of patterns which are not easy to extract with traditional techniques.

5.2 Quality of Clustering

We run BIRCH, CURE, and DPDC on the eight datasets in order to evaluate the quality of their final clusters. In the case of the DPDC approach, we consider a Hadoop system that contains two nodes (Mappers), the final results are the aggregation of the two local clustering. Figure 7 shows the generated clusters by each of the three algorithms performed on the first three datasets (T_1, T_2, and T_3) and Figure 8 shows the clusters returned by the tree algorithms performed on the remaining five datasets with noise (T_4, T_5, T_6, T_7, and T_8). We use different colours to show the clusters returned by each algorithm.

From the results shown in Figures 7 and 8, Both BIRCH and CURE could not find the correct clusters across the eight datasets. CURE performs better than BIRCH on the datasets with convex shaped patterns, whereas they both perform badly on datasets with noise and non-convex shaped patterns. The DPDC approach

Table 2.: The datasets used to evaluate the algorithms.

Type	**Dataset**	**Description**	**#Points**	**#Clusters**
Convex	T1	Big oval (egg shape)	14,000	5
	T2	4 small circles and 2 small circles linked	17,080	5
	T3	2 small circles, 1 big circle and 2 linked ovals	30,350	4
Non-Convex with Noise	T4	Different shapes, with some clusters surrounded by others	10,000	9
	T6	Different shapes with noises	8,000	6
	T7	Letters with noise	8,000	6
	T8	Different shapes with noises	321	6

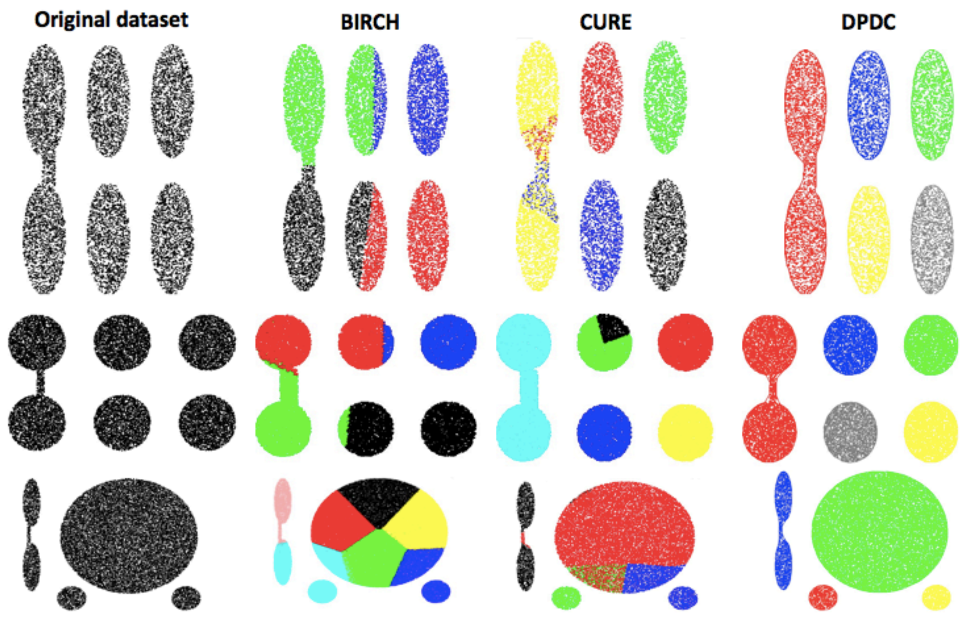

Figure 7.: Clusters generated by DPDC for datasets containing patterns of convex shapes.

generates accurate results across all the eight datasets. In fact DPDC obtained a 100% accuracy compared two BIRCH and CURE. DPDC performs very well on all types of datasets (with convex and non-convex shaped patterns and also with and without noise). Its implementation using MapReduce does not generate significant overheads, mainly for very large datasets.

5.3 DPDC: Speed-up

We have implemented the DPDC approach using MapReduce programming paradigm and we tested it in an actual Hadoop system that consists of a cluster of 21 processing nodes.

We conducted all the experiments on a 21-nodes cluster (it is configured as one master and twenty slaves). Each of the nodes runs Ubuntu 14.04 and is equipped

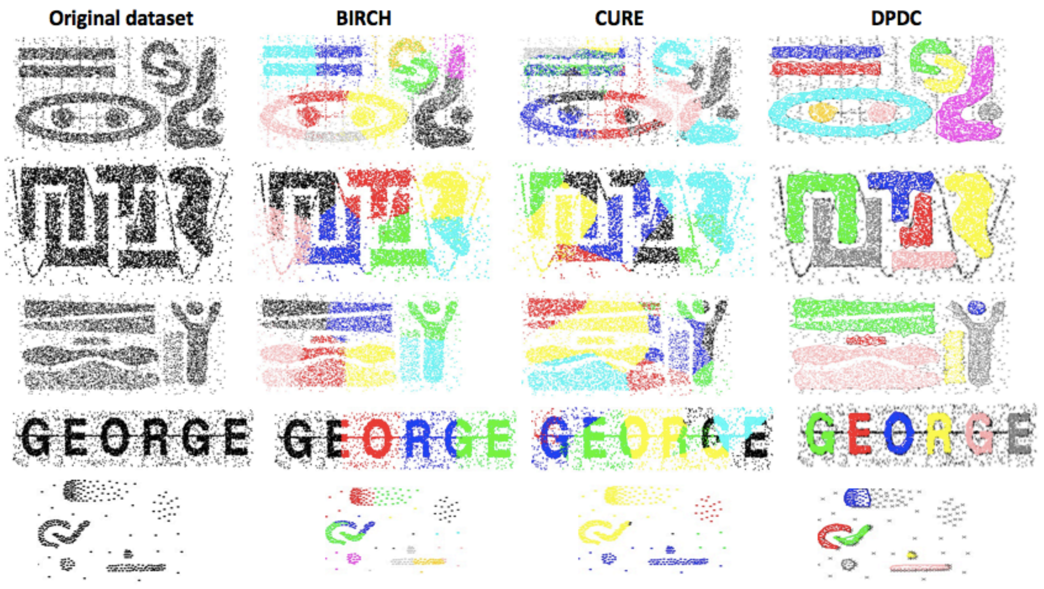

Figure 8.: Clusters generated by DPDC for datasets that contain patterns of non-convex shapes.

with 8 GB of RAM and 2.70 GHz quad Intel Core i5-6400 processor. All the cluster nodes are interconnected by a gigabit Ethernet switch. One of the nodes is configured as both `JobTracker` and `NameNode`. The other nodes are configured as computing nodes. For MapReduce platform, we use Hadoop 2.7.2. Both map and reduce slots of each slave node are set to 4 in accordance with the number of cores. Therefore, at most 80 map tasks and 80 reduce tasks can run concurrently in this computing cluster. The block size of HDFS is 256MB and each block is replicated three times for fault-tolerance. We also conducted the same experiments on a single-node (Hadoop 2.7.2) with different specifications from the above. We use a single node that runs Ubuntu 16.04 and equipped with 8 GB of RAM and 2.30 GHz quad Intel Core i5-6200 processor. The results are discussed in the following

5.3.1 DPDC on a Single Node

The goal here is to study the scalability of the DPDC algorithm. We cluster one million points in a system that contains up to 100 mappers executed on a single node. Figure 9 shows the results of this execution. In another experiment, we run DPDC on a single-node system using smaller datasets to see how does the DPDC is affected with different sizes of datasets. The results are shown in Figure 10.

As we can see from Figure 9, the execution time of the DPDC algorithm keeps decreasing sharply between 10 and 50 mappers, after that it becomes steady and it starts increasing around 70 mappers. The reason is that the algorithm is running on a single machine with 4 cores. Therefore, the mappers are not running in a purely parallel system. In addition, after 60 mappers we do not gain much from the DBSCAN execution time, whereas the time that Hadoop spends to read and write the results in the HDFS files keep increasing. As a result, the overall execution time of the algorithm increases.

Figure 10 shows that the execution time of the DPDC algorithm decreases from 1 to 2 mappers for dataset T_1, after that the execution time starts increasing from 2 to 4 mappers and it continues to increase linearly, the same for dataset T_2 and T_3. In fact, for T_3, the execution time starts increasing since the beginning. Because these three datasets are relatively small; the gain we get in in executing DBSCAN on smaller subsets is much smaller than the time that Hadoop spends on merging,

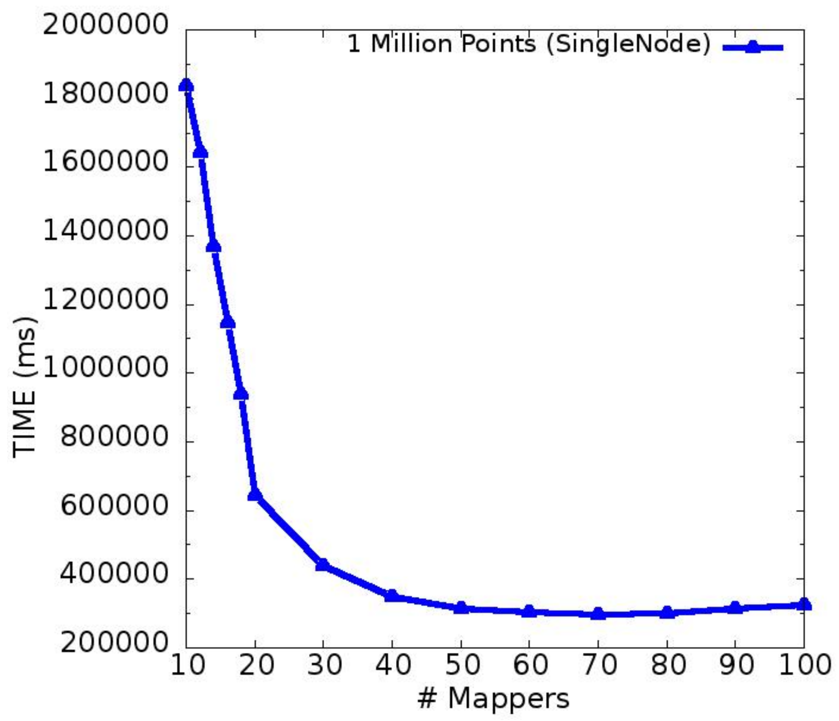

Figure 9.: DPDC scalability on a single-node on large dataset.

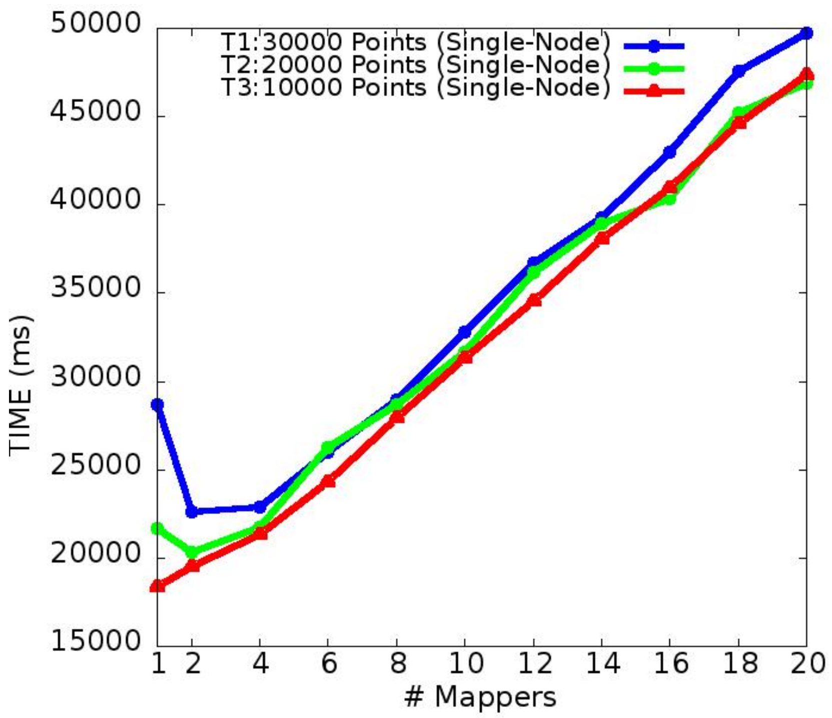

Figure 10.: DPDC scalability on a single-node on small datasets.

communicating and reading and writing the results into the file system.

5.3.2 DPDC on Multiple Nodes

The goal here is to perform the same experiments as above but on a Hadoop cluster with multiple nodes. We did two experiments; we first try to cluster one million points while varying the number of mappers in the system. Then, we run the algorithm on smaller datasets, T_1, T_2, and T_3, having 30,000, 20,000 and 10,000 points respectively.

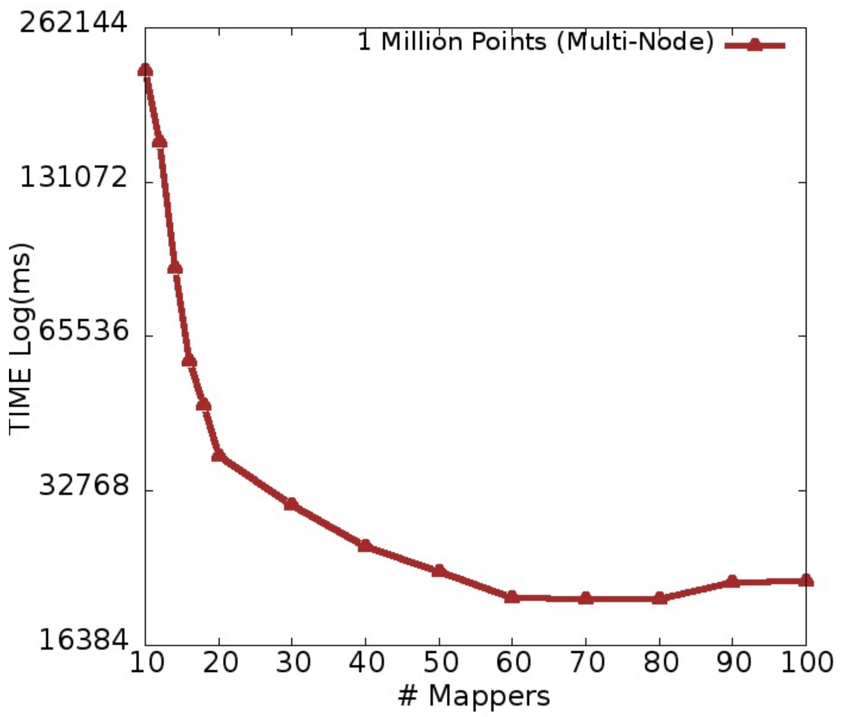

Figure 11.: DPDC scalability on a multi-node cluster on large dataset.

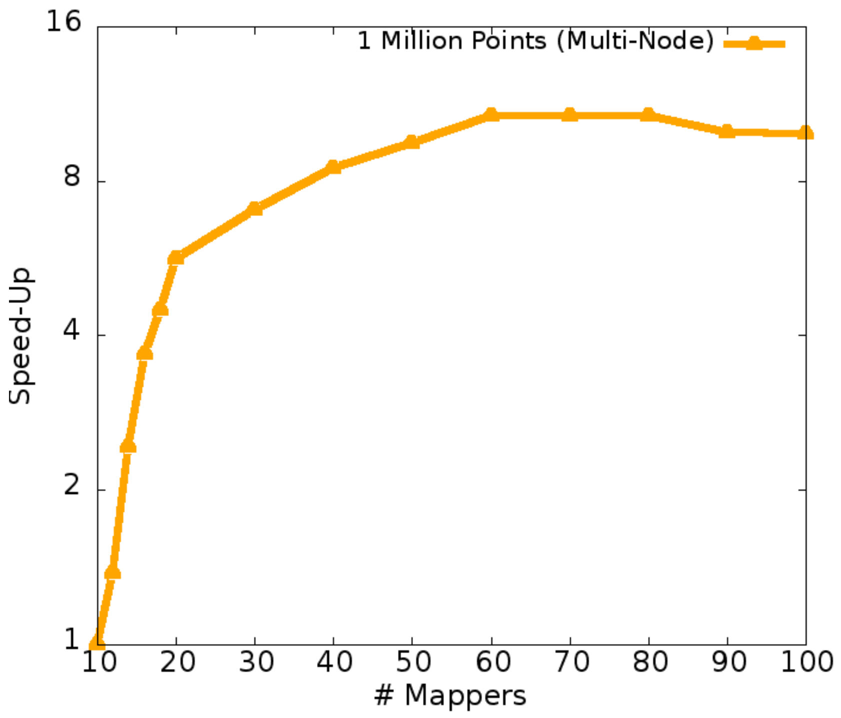

Figure 12.: DPDC Speed-up on a multi-node cluster on large dataset.

As we can see from Figure 11, the DPDC's execution time keeps decreasing as the number of mappers increases. In fact, the time decreases dramatically between 10 and 20 mappers as the gain in time for DBSCAN algorithm while dividing the data is huge. After 20 mappers the time continues to decrease gradually until it reaches 80 mappers. The reason for that is as mentioned above, we have 20 machines in the system and each machine has 4 cores, therefore, 80 mappers can run concurrently in parallel, after that, the execution is not purely parallel anymore. As a result, the execution time starts to increase after 80 mappers.

The execution times of DPDC are shown in Figure 11 using logarithmic scaling. Figure 12 shows the DPDC's speed-up and how it evolves with the number of mappers in the system.

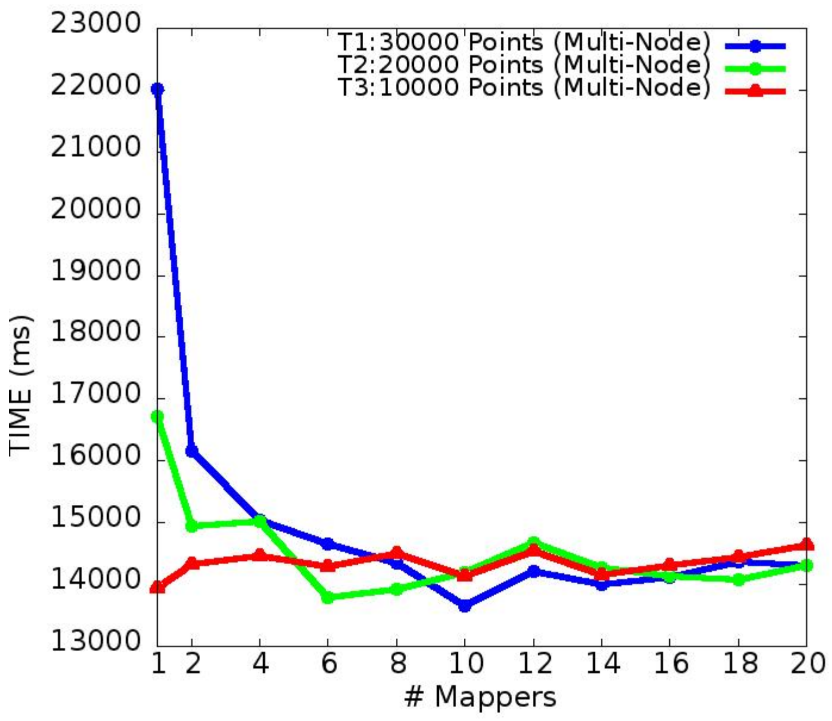

Figure 13.: DPDC scalability on a multi-node cluster on small datasets.

As one can see from Figure 13, the DPDC's execution time for T_1 decreases considerably from 1 to 2 mappers and it continues decreasing up to 10 mappers and then it starts increasing. The reason is that between 1 and 10 mappers the gain on execution time for DBSCAN is significant compared to the time that Hadoop spends on running the system. While after 10 mappers, the execution time of DBSCAN on very small datasets is not significant compared to the time that Hadoop spends to communicate the results. The same behaviour of the approach on other datasets (t_2 and T_3)

Finally, from both Figures 11 and 13, we can notice that MapReduce and Hadoop work better with medium to large datasets, Whereas, it does not perform well on small datasets.

5.4 DPDC: Computational Complexity

Let N be the number of data objects in the dataset. The complexity of our approach is the sum of the complexities of its three components: local clustering, local reduction, and global aggregation.

Phase1: Local clustering Assume that the local clustering algorithm is DBSCAN for all the nodes. The cost of this phase is given by:

$$T_{Phase_1} = \operatorname*{Max}_{i=1}^{M}(DBSCAN_i) + \operatorname*{Max}_{i=1}^{M}(Reduction_i)$$

Where M is the number of nodes in the system. The complexity of DBSCAN while including the distance matrix computation is $\mathcal{O}(n^2)$. Finally, the complexity of the local reduction algorithm is $\mathcal{O}(n \log n)$.

Phase2: Aggregation Our global aggregation depends on the hierarchical combination of contours of local clusters. As the combination is based on the intersection of edges from the contours, the complexity of this phase is $\mathcal{O}(v \log v + p)$. Where v is the number of vertices and p is the number of intersections between edges of different contours (polygons).

Total complexity The total complexity of our approach is $T_{Total} = \mathcal{O}(n^2) + \mathcal{O}(n \log n) + \mathcal{O}(v \log v + p)$, which is:

$$T_{Total} \simeq \mathcal{O}(n^2)$$

We notice that the time complexity of the whole DPDC approach while using DBSCAN as local clustering algorithm is absorbed by the time complexity of the DBSCAN algorithm which is of quadratic time $\mathcal{O}(n_i^2)$. that is what explain the fact that the more we divide the data the more we gain in terms of execution time, in fact, every time we divide the data, we gain 4 time in terms of execution time, that's makes the approach scalable.

6. Conclusion

Big Data arises with many challenges, which traditional techniques of data mining fail to address properly. As the data we collect nowadays is by nature distributed, and 80% of it is spatial or has an element of space and time in it, we proposed a parallel and distributed framework to efficiently analyse big spatial datasets. The proposed approach for clustering large spatial datasets exploits the advantages of distributed and parallel computing (cloud computing) while avoiding the inherent limitations of some existing approaches, such as partitioning and hierarchical clustering.

The Dynamic Parallel and Distributed Clustering technique is efficient, flexible, scalable, and can work with the existing data mining algorithms. The framework was tested on spatial datasets using the K-means and DBSCAN algorithms on various benchmarks datasets. The approach produced excellent results in terms of quality of the final results and response time. The DPDC approach has two main phases: the fully parallel phase where each node of the system calculates its own local clusters based on its dataset. There is no communications during this phase, it takes full advantage of task parallelism. The second phase is distributed; it generates some communications between the nodes, which are reduced to minimum. Moreover, the approach has low computational complexity when executed in a parallel environment.

We implemented the DPDC approach using one of the most popular programming paradigm, which is MapReduce. We showed the this approach is well suited for this paradigm and also showed that the MapReduce implementation performance depends on the size of the datasets at hand; it is scalable with the size of the datasets and the number of the processing nodes in the Hadoop system. In addition, the experimental results show that the approach returns high-quality clustering compared to the traditional clustering algorithms. As future work, we plan to extend this approach to non-spatial and high-dimensional datasets.

Acknowledgment

The research work is conducted in the Insight Centre for Data Analytics, which is supported by Science Foundation Ireland under Grant Number SFI/12/RC/2289.

References

[1] Big data and analytics builds the foundation for cognitive. `1www.idc.com/prodserv/4Pillars/bigdata`, 2017.

[2] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques.* Elsevier, 2011.

[3] Daniele Bacarella. Distributed clustering algorithm for large scale clustering problems, 2013.

[4] Grigorios Tsoumakas and Ioannis Vlahavas. Distributed data mining. *Encyclopedia of Data Warehousing and Mining*, 2009.

[5] Yongjian Fu. Distributed data mining: An overview. *Newsletter of the IEEE Technical Committee on Distributed Processing*, 5(9), 2001.

[6] Byung-Hoon Park and Hillol Kargupta. Distributed data mining: Algorithms, systems, and applications. *Citeseer*, 2002.

[7] Laurent Yeh Karine Zeitouni. Le data mining spatial et les bases de données spatiales. *Revue internationale de géomatique. Volume*, 9(4), 1999.

[8] Sukumar Ghosh. *Distributed systems: an algorithmic approach.* CRC press, 2014.

[9] Lamine Aouad, Nhien-An Le-Khac, and Tahar Kechadi. Image analysis platform for data management in the meteorological domain. In *7th Industrial Conference in Data Mining Proceedings*, volume 4597, pages 120–134. Springer Berlin Heidelberg, 2007.

[10] X. Wu, X. Zhu, G. Q. Wu, and W. Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, 2014.

[11] Lior Rokach, Alon Schclar, and Ehud Itach. Ensemble methods for multi-label classification. *Expert Systems with Applications*, 41:7507 – 7523, 2014.

[12] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[13] Jen-Wei Huang, Su-Chen Lin, and Ming-Syan Chen. Dpsp: Distributed progressive sequential pattern mining on the cloud. *Advances in Knowledge Discovery and Data Mining*, pages 27–34, 2010.

[14] Xin Yue Yang, Zhen Liu, and Yan Fu. Mapreduce as a programming model for association rules algorithm on hadoop. In *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*, pages 99–102. IEEE, 2010.

[15] Xueyan Lin. Mr-apriori: Association rules algorithm based on mapreduce. In *Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on*, pages 141–144. IEEE, 2014.

[16] Liang-Chi Hsieh, Guan-Long Wu, Yu-Ming Hsu, and Winston Hsu. Online image search result grouping with mapreduce-based image clustering and graph construction for large-scale photos. *Journal of Visual Communication and Image Representation*, 25(2):384–395, 2014.

[17] Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, and Jianping Fan. Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[18] Tianyang Sun, Chengchun Shu, Feng Li, Haiyan Yu, Lili Ma, and Yitong Fang. An efficient hierarchical clustering method for large datasets with map-reduce. In *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on*, pages 494–499. IEEE, 2009.

[19] Younghoon Kim, Kyuseok Shim, Min-Soeng Kim, and June Sup Lee. Dbcure-mr: An efficient density-based clustering algorithm for large data using mapreduce. *Inf. Syst.*, 42:15–35, June 2014.

[20] Malika Bendechache and M-Tahar Kechadi. Distributed clustering algorithm for spatial data mining. In *2nd International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*, pages 60–65. IEEE, 2015.

[21] Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. Efficient large scale clustering based on data partitioning. In *International Conference on Data Science and Advanced Analytics (DSAA)*, pages 612–621. IEEE, 2016.

[22] Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. Performance evaluation of a distributed clustering approach for spatial datasets. In *15th International Conference on Australasian Data Mining Conference (AusDM)*. CRPIT, 2017.

[23] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[24] Leonard Kaufman and Peter J Rousseeuw. Partitioning around medoids (program pam). *Finding groups in data: an introduction to cluster analysis*, pages 68–125, 1990.

[25] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[26] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[27] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[28] Lamine Aouad, Nhien-An Le-Khac, and Tahar Kechadi. Lightweight clustering technique for distributed data mining applications. *Advances in Data Mining. Theoretical Aspects and Applications*, pages 120–134, 2007.

[29] Inderjit Dhillon and Dharmendra Modha. A data-clustering algorithm on distributed memory multiprocessor. In *large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260. Springer-Verlag London, UK, 1999.

[30] A. Garg, A. Mangla, V. Bhatnagar, and N. Gupta. Pbirch: A scalable parallel clustering algorithm for incremental data. *10th Int'l. Symposium on Database Engineering and Applications (IDEAS-06)*, pages 315–316, 2006.

[31] Huimin Geng, Xutao Deng, and Hesham Ali. A new clustering algorithm using message passing and its applications in analyzing microarray data. In *Proceedings. Fourth International Conference on Machine Learning and Applications*, pages 6–pp. IEEE, 2005.

[32] Inderjit D. Dhillon and Dharmendra S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-Scale Parallel Data Mining*, pages 245–260. Springer Berlin Heidelberg, 2000.

[33] Xiaowei Xu, Jochen Jger, and Hans-Peter Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery archive*, 3:263–290, September 1999.

[34] J-F Laloux, N-A. Le-Khac, and M-T. Kechadi. Efficient distributed approach for density-based clustering. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 20th IEEE International Workshops*, pages 145–150, 27-29 June

2011.

[35] Nhien An Le Khac, Lamine M Aouad, and M-Tahar Kechadi. Knowledge map layer for distributed data mining. *Journal of ISAST Transactions on Intelligent Systems*, 1(1), 2008.

[36] John F Roddick, Kathleen Hornsby, and Myra Spiliopoulou. An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In *Temporal, Spatial, and Spatio-Temporal Data Mining*, pages 147–163. Springer, 2001.

[37] Jyrki Kivinen and Heikki Mannila. The power of sampling in knowledge discovery. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 77–85. ACM, 1994.

[38] Paolo Compieta, Sergio Di Martino, Michela Bertolotto, Filomena Ferrucci, and T Kechadi. Exploratory spatio-temporal data mining and visualization. *Journal of Visual Languages & Computing*, 18(3):255–279, 2007.

[39] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *17th International Conference on Parallel and Distributed Systems*, pages 473–480. IEEE, 2011.

[40] Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. Hierarchical aggregation approach for distributed clustering of spatial datasets. In *16th International Conference on Data Mining Workshops (ICDMW)*, pages 1098–1103. IEEE, 2016.

[41] A.Ray Chaudhuri, B.B. Chaudhuri, and S.K. Parui. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *Computer vision and Image Understranding*, 68:257–275, 03 December 1997.

[42] Mahmoud Melkemi and Mourad Djebali. Computing the shape of a planar points set. *Elsevier Science*, 33:14231436, 9 September 2000.

[43] Matt Duckhama, Lars Kulikb, Mike Worboysc, and Antony Galtond. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Elsevier Science Inc. New York, NY, USA*, 41:3224–3236, 15 March 2008.

[44] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD-96 Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, volume 25, pages 103–114. ACM New York, USA, 1996.

[45] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Information Systems*, volume 26, pages 35–58. Elsevier Science Ltd. Oxford, UK, 17 November 2001.