Distributed Spatial Data Clustering as a New Approach for Big Data Analysis

Malika Bendechache, Nhien-An Le-Khac, M-Tahar Kechadi
Insight Centre for Data Analytics
School of Computer Science
University College Dublin, Ireland
malika.bendechache@ucdconnect.ie

Abstract

In this paper we propose a new approach for Big Data mining and analysis. This new approach works well on distributed datasets and deals with data clustering task of the analysis. The approach consists of two main phases: the first phase executes a clustering algorithm on local data, assuming that the datasets was already distributed among the system processing nodes. The second phase deals with the local clusters aggregation to generate global clusters. This approach not only generates local clusters on each processing node in parallel, but also facilitates the formation of global clusters without prior knowledge of the number of the clusters, which many partitioning clustering algorithm require. In this study, this approach was applied on spatial datasets. The proposed aggregation phase is very efficient and does not involve the exchange of large amounts of data between the processing nodes. The experimental results show that the approach has super-linear speed-up, scales up very well, and can take advantage of the recent programming models, such as MapReduce model, as its results are not affected by the types of communications.

Keywords: Distributed data mining, distributed computing, synchronous communication, asynchronous communication, super-speedup, spacial data mining.

1 Introduction

Nowadays big data is becoming a commonplace. It is generated by multiple sources at rapid pace, which leads to very large data volumes that need to be stored, managed, and analysed for useful insights. From organisations point of view, it is not the size of the generated data which is important. It is what we learn from it that matters, as this may help understanding the behaviour of the system that is governed by this data or help to make some key decisions, etc. To extract meaningful value from big data, we need appropriate and efficient mining and analytics techniques to analyse it. One of the most powerful and common approaches of analysing datasets for extracting useful knowledge is clustering. Clustering has a wide range of applications and its concept is so interesting that numerous algorithms for various types of data have been proposed and implemented. However, big data come up with new challenges, such as large volumes, velocity, variety, and veracity, that the majority of popular clustering algorithms are inefficient at very large scale. This inefficiency can be that the final results are not satisfactory or the algorithm has high complexity which requires large computing power and response time to produce final results. There are two major categories of approaches to deal with computational complexity of these clustering algorithms: 1) the first category consists of reducing the size of the initial dataset. One can use either sample-based techniques or dimensionality reduction techniques. The second category consists of using parallel and distributed computing to speed up the response time. In this case we can try to parallelise or model the algorithm in the form of a client-server model using MapReduce mechanism. However, these algorithms are inherently difficult to parallelise, and designing an efficient distributed version of the algorithm is not straightforward either. This is due to the fact that the processing nodes, either in the parallel or distributed versions, need to communicate and coordinate their efforts in order to obtain the same results. These communications are extremely expensive and can cancel the benefit of the parallelised version. To deal with these challenging issues, we propose to study a distributed approach that takes advantage of parallel and distributed computing power, while getting ride of the drawbacks of the previous methods. In addition, one of the main advantages of our approach is that it can be used as a framework for all clustering algorithms. In other words, while it is well known that there is no clustering algorithm that can universally be used to cluster every dataset of a given application, our approach can be used for all algorithms or a set of algorithms to derive a distributed clustering approach for a given data having specific characteristics.

The proposed approach has two main phases: the first phase, based on the SPMD paradigm, consists of dividing the datasets into K partitions, where K is the number of processing nodes. Then, for each partition we cluster its data into C_i clusters. This phase is purely parallel, as each processing node executes a clustering algorithm on its data partition independently of the others. The obtained clusters on each node are called local clusters. This phase does not require any communications, and in addition, in the majority of applications the data is collected by various sources, which are geographically distributed. Therefore, the data is already partitioned. All required is to cluster locally the data. The second phase consists of aggregating (or merging) the local clusters to obtain global clusters by merging overlapping cluster. In order to determine whether two local clus-

ters belonging to two different nodes are overlapping or not, one needs to exchange the local clusters between the nodes. This operation is extremely expensive when the dataset is very large. The main idea of our approach is to minimise the data exchange while maximising the quality of the global clusters. The methode used to aggregate spatial local clusters into global clusters allows only to exchange about 2% of the original datasets (Laloux et al. 2011), which is highly efficient. In this paper, we want to study the performance of such distributed clustering technique by calculating its speedup compared to the sequential version of the algorithm, its scalability, its communication overheads, and its complexity in general.

The rest of the paper is organised as follows: In the next section we will give an overview of the state-of-the-art of parallel and distributed data mining techniques and discuss their limitations. Then we will present in more details the proposed distributed framework and its concepts in Section 3. In section 4, we evaluate its performance based on two types of implementations; synchronous and asynchronous communications. In section 5, we discuss the experimental results based on speedup, scalability, communication overheads, and compare the two implementation models; synchronous and asynchronous. Finally, we conclude in Section 6.

2 Related Work

Distributed Data Mining (DDM) is a line of research that has attracted much interest in recent years (Jiawei Han 2006). DDM was developed because of the need to process data that can be very large or geographically distributed across multiple sites. This has two advantages: first, a distributed system has enough processong power to analyse the data within a reasonable time frame. Second, it would be very advantageous to process data on their respective sites to avoid the transfer of large volumes of data between the site to avoid heavy communications, network bottlenecks, etc.

DDM techniques can be divided into two categories based on the targeted architectures of computing platforms (Zaki 2000). The first, based on parallelism, uses traditional dedicated and parallel machines with tools for communications between processors. These machines are generally called supercomputers. The second category targets a network of autonomous machines. These are called distributed systems, and are characterised by distributed resources, low-speed network connecting the system nodes, and autonomous processing nodes which can be of different architectures, but they are very abundant (Ghosh 2014). The main goal of this 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 (Aouad, Le-Khac & Kechadi. 2007, Wu et al. 2014).

However, the traditional DDM methods are not always effective, as they suffer from the problem of scaling. One solution to deal with large scale data is to use parallelism, but this is very expensive in terms of communications and processing power. Another solution is to reduce the size of training sets (sampling). Each system node generates a separate sample. These samples will be analysed using a single global algorithm (Tian Zhang 1996, A. K. Jain 1999). However, this technique has a disadvantage that the sampling in this case is very complex and requires many communications between the nodes which may impact on the quality of the samples and therefore the final results. This has led to the development of techniques that rely on ensemble learning (Rokach et al. 2014, Eric Bauer 1999). These new techniques are very promising, as each technique of the ensemble netwrok attempts to learn from the data and the best or compromised results of the network will emerge as the winner. Integrating ensemble learning methods in DDM framework will allow to deal with the scalability problem, as it is the case of the proposed approach.

Clustering algorithms can be divided into two main categories, namely partitioning and hierarchical. Different elaborated taxonomies of existing clustering algorithms are given in the literature. Many parallel clustering versions based on these algorithms have been proposed in the literature (Aouad, Khac & Kechadi 2007, Dhillon & Modha 1999, Ester et al. 1996, Garg et al. 2006, H.Geng et al. 2005, Dhillon & Modha 2000, Xu et al. 1999). These algorithms are further classified into two sub-categories. The first consists of methods requiring multiple rounds of message passing. They require a significant amount of synchronisations and data exchange. The second subcategory consists of methods that build local clustering models and send them to a central site to build global models (Laloux et al. 2011).

In (Dhillon & Modha 1999) and (Dhillon & Modha 2000), message-passing versions of the widely used K-Means algorithm were proposed. In (Ester et al. 1996) and (Xu et al. 1999), the authors dealt with the parallelisation of DBSCAN; density-based clustering algorithm. In (Garg et al. 2006) 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 (H.Geng et al. 2005). Most of the parallel approaches need either multiple synchronisation constraints between processes or a global view of the dataset, or both (Aouad, Khac & Kechadi 2007). All these approaches deal with the parallelisation of the sequential version of the algorithm by trying phases of the algorithm which can be executed in parallel by several processors. However, this requires many synchronisations either to access shared data (for the shared memory model) or communications (for message passing model). In some algorithm these synchronisations and communications are extremely expensive and it is not worth parallelising them. This approach is not usually scalable.

In (Brecheisen et al. 2006) a client-server model is adopted, where the data is equally partitioned and distributed among the servers, each of which computes the clusters locally and sends back the results to the master. The master merges the partially clustered results to obtain the final results. This strategy incurs a high communication overhead between the master and slaves, and a low parallel efficiency during the merging process. Other parallelisations using a similar client-server model include (Arlia & Coppola 2001, Chen et al. 2010, Coppola & Vanneschi 2002, Fu et al. 2011, Guo & Grossman 2002, Zhou et al. 2000). Among these approaches, various programming mechanisms have been used, for example, a special parallel programming environment, called skeleton based programming in (Coppola & Vanneschi 2002) and parallel virtual machine in (Guo & Grossman 2002). A Hadoop-based approach is presented in (Fu et al. 2011).

Another approach presented in (Aouad, Khac & Kechadi 2007) also applied a merging of local models to create the global models. 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 (Laloux et al. 2011). In addition, both partitioning and hierarchical categories have some issues which are very difficult to deal with in parallel versions. For the partitioning class, it needs the number of clusters to be fixed in advance, while in the majority of applications the number of classes is not known in advance. For the hierarchical clustering algorithms, they have the issue of stopping conditions for clustering decomposition, which is not an easy task and mainly in distributed versions.

3 Dynamic Distributed Clustering

Dynamic Distributed Clustering (DDC) model is introduced to deal with the limitations of the parallel and master-slave models. DDC combines the characteristics of both partitioning and hierarchical clustering methodes. In addition, it does neither inherit the problem of the number of partitions to be fixed in advance nor the problem of stopping conditions. It is calculated dynamically and generates global clusters in a hierarchical way. All these features look very promising and some of them have been thoroughly studies in (Bendechache, Kechadi & Le-Khac 2016), such as the dynamic calculation of the number of the clusters and the accuracy of the final clustering, in this study one wants to show the effect of the communications on the response time, the communication model used, the scalability of the approach, and finally its performance in terms of speed up compared to the sequential version. In this paper we will focus on

- Synchronous and asynchronous communications, as this approach can be implemented either with synchronous or asynchronous communications. Both implementations produce the same results.
- The speed-up of the DDC approach using DBSCAN as the basic algorithm for clustering the partitions. This algorithm is known to have non-polynomial complexity ($O(n^2)$).
- Scalability of the approach as the size of the dataset increases.

We start by briefly explaining the algorithm and then present a performance and evaluation model for the approach.

The DDC approach has two main phases. In the first phase, we cluster the datasets located on each processing node and select good local representatives. All local clustering algorithms are executed in parallel without communications between the nodes. As DBSCAN is the basic algorithm for clustering local datasets, we can reach a super lineare speed-up of p^2, where p is the number of processing nodes. The second phase collects the local clusters from each node and affects them to some special nodes in the system; called leaders. The leaders are elected according to their characteristics such as capacity, processing power, connectivity, etc. The leaders are responsible for merging the local clusters. In the following we explain how the local clusters are represented and merged to generate global clusters.

3.1 Local Models

The local clusters are highly dependent on the clustering techniques used locally in each node. For instance, for spatial datasets, the shape of a cluster is usually dictated by the technique used to obtain them. Moreover, this is not an issue for the first phase, as the accuracy of a cluster affects only the local results of a given node. However, the second phase requires sending and receiving all local clusters to the leaders. As the whole data is very large, this operation will saturate very quickly the network. So, we must avoid sending all the original data through the network. The key idea of the DDC approach is to send only the cluster's representatives, which constitute between 1% and 2% of the whole data. The cluster representatives consist of the internal data representatives plus the boundary points of the cluster.

There are many existing data reduction techniques in the literature. Many of them are focusing only on the dataset size. For instance, they try to reduce the storage capacity without paying attention to the knowledge contained in the data. In (Le-Khac et al. 2010), an efficient reduction technique has been proposed; it is based on density-based clustering. Each cluster is represented by a set of carefully selected data-points, called representatives. However, selecting representatives is still a challenge in terms of quality and size (Januzaj et al. 2004, Laloux et al. 2011).

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 1). Many algorithms for extracting the boundaries from a cluster can be found in the literature (Fadilia et al. 2004, Chaudhuri et al. 1997, Melkemi & Djebali 2000, Edelsbrunner et al. 1983, Moreira & Santos 2007). We use an algorithm based on triangulation to generate the clusters' boundaries (Duckhama et al. 2008). 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 $O(n \log n)$.

3.2 Global Models

The global clusters are generated in the second phase of the DDC. This phase is also executed in a distributed fashion but, unlike the first phase, it has communications overheads. This phase consists of two main steps, which can be repeated until all the global clusters were generated. First, each leader collects the local clusters of its neighbours. Second, the leaders will merge the local clusters using the overlay technique. The process of merging clusters will continue until we reach the root node. The root node will contain the global clusters (see Figure 1).

The pseudo code of the algorithm is given in Algorithm 1.

In DDC we only exchange the boundaries of the clusters. The communications can be synchronous or asynchronous. We implemented this phase using both types of communications. An evaluation model is presented in the next Section.

4 DDC EVALUATION

In order to evaluate the performance of the DDC approach, we use different local clustering algorithms. For instance, with both K-Means (Bendechache & Kechadi 2015) and DBSCAN (Bendechache, Kechadi & Le-Khac 2016, Bendechache, Le-Khac & Kechadi 2016), the DDC approach outperforms existing algorithms in both quality of its results and response time including K-Means and DBSCAN applied to the whole dataset. In this section we evaluate its speed-up, scalability, and which architecture is more appro-

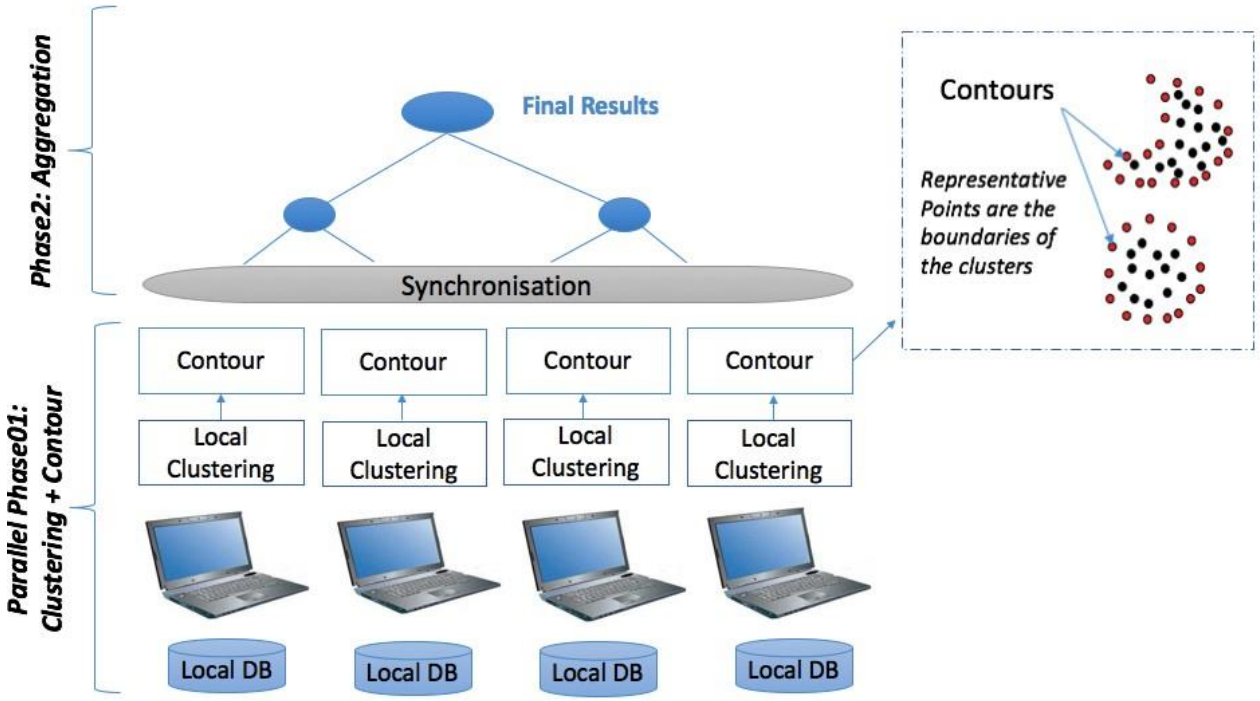

Figure 1: An overview of the DDC Approach.

priate to implement it. In addition, we compare DDC with the sequential version of the basic clustering algorithm used within DDC.

The proposed approach is more developed for distributed systems than pure parallel systems. Therefore, it is worth analysing the benefits of using synchronous or asynchronous processing mechanism, as distributed systems are asynchronous and the blocking operations have a strong impact in communication time (Solar et al. 2013).

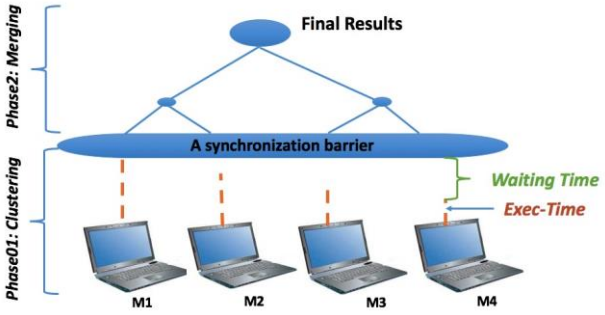

Figure 2: Synchronous communications.

In synchronous model, as illustrated in Figure 2, although machines M_3 and M_4 have finished their computations before M_1 and M_2, they can not send their results until M_1 and M_2 finish as well. In this model, Not only the computations and communications are not overlapped but also the machines which finished early wasted sometime waiting for the other to finish (Solar et al. 2013).

In asynchronous model the machines which finished early can advance to the next step. The machines manage their communications and the 1st and 2nd phase overlap. This model is much more suitable for distributed computing, where the nodes are heterogeneous and the communications are usually slow.

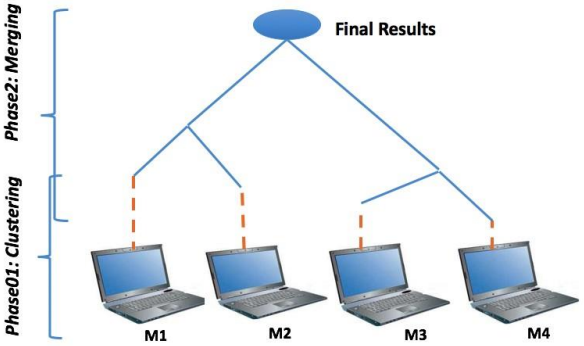

Figure 3: Asynchronous communications.

As can be seen in the example given in Figure 3, M_3 and M_4 start merging their results before M_1 and M_2 finish their computations.

4.1 DDC Computational Complexity

Let M be the number of nodes and n_i the dataset given to each node v_i in the system. The complexity of our approach is the sum of its components' complexity; local mining, local reduction, and global aggregation.

Phase1 - Local clustering: Let $\Gamma(n_i)$ denote the local clustering algorithm running on node (v_i), and $\Delta(c_i)$ be the time required to execute the reduction algorithm. The cost of this phase is given by:

$$T_{Phase_1} = \max_{i=1}^{M}(\Gamma(n_i) + \Delta(c_i)) \qquad (1)$$

Where c_i is the cluster points generated by node v_i. Note that the reduction algorithm is of complexity $O(c_i \log c_i)$.

Algorithm 1 DDC Algorithm

initialization
$Node_i \in N$, N: The Total nodes in the system.

Phase 1 – Local Clustering

input : X_i: Dataset Fragment, $Params_i$: Input parameters for the local clustering: $Params_i$= (Eps_i, $MinPts_i$) for DBSCAN
output: C_i: Cluster's contours of $Node_i$

foreach $Node_i$ **do**
L_i=Local_Clustering(X_i,$Params_i$)
// $Node_i$ executes a clustering algorithm locally.
C_i=Contour(L_i) // $Node_i$ executes a contour algorithm locally.
end

Phase 2 – Merging

input : D: tree degree, C_i: Local cluster's contours generated by $Node_i$ in the phase01
output: $C_{G_{k,level}}$: Global Cluster's contours (global results, level=0)

repeat
$level = treeheight$
$Node_i$ joins a group $G_{K,Level}$ of D elements
// $Node_i$ joins its neighbourhood
$Node_j$=ElectLeaderNode($G_{K,Level}$)
// $Node_j$ is the leader of the group G
// In parallel
foreach $Node_i \in G_{K,Level}$ **do**
if $(i <> j)$ **then**
Send (C_i, $Node_j$) // Each node sends its contours to others nodes in the same group of neighbourhood
else
Recv ($C \equiv (\{C_l\}, Node_l)$) // If the node is the leader, it will receive the others node's contours in the same group of neighbourhood
$G_{K,Level}$= Merge (C_i, C) // Merge the overlapping contours
end
end
$level$ - -
until $(level == 0)$;
return $(C_{G_{k,0}})$

Phase2 - Aggregation: The 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 $O(w_i \log w_i + p)$. Where w_i is the total vertices of the contours by node v_i and p is the intersection points between edges of different contours (polygons).

Total complexity: The total complexity of the DDC approach, assuming that the local clustering algorithm is DBSCAN which is of complexity $O(n^2)$, is:

$$T_{Total} = O(n_i^2) + O(c_i \log c_i) + O(w_i \log w_i + p) \approx O(n_i^2) \quad (2)$$

4.2 DDC Speedup

The DDC speedup is calculated against the sequential version of the approach. The sequential version consists clustering all the data on one machine. Therefore, it does require neither reduction nor aggregation. Let T_1 be the execution time of the sequential version and T_p the execution time of the DDC on p nodes. The speedup α is given by

$$\alpha = \frac{T_1}{T_p} \quad (3)$$

Note that if the complexity of the clustering algorithm is polynomial then the optimal speedup that can be reached is P, under the condition that there is no overhead due to communications and extra work. If the complexity of the clustering algorithm is $O(n^2)$ then the optimal speedup can be P^2; this is called *super speedup*. In the following section we will evaluate the speedup in the case of DBSCAN.

5 Experimental Results

We have implemented our approach on a distributed computing system. The distributed computing system consists of heterogeneous desktops (different CPUs, OSs, memory sizes, loads, etc.). We use JADE (Java Agent DEvelopment), as a development platform to implement the approach. JADE is based on a 2P2 communication architecture, It allows to use heterogeneous processing nodes, it is scalable, and dynamic (Cortese 2005, Bellifemine et al. 2005).

The system nodes (desktops) are connected to local area networks. This allows us to add as many nodes as required, depending on the experiment. Table 1 lists types of machines used to perform the experiments. The main goal here is to demonstrate the performance of the DDC in a heterogeneous distributed computing environment.

Table 1: The characteristics of the used Machines

Machine's name	Operating System	Processor	Memory
Dell-XPS L421X	Ubuntu (V.14.04 LTS)	1.8GHz*4 Intel Core i5	8 GB
Dell-Inspiron-3721	Ubuntu (V.14.04 LTS)	2.00GHz*4 Intel Core i5	4 GB
Dell-Inspiron-3521	Ubuntu (V.16.04 LTS)	1.8 GHz*4 Intel Core i5	6 GB
iMac-Early 2010	cinux Mint (V.17.1 Rebecca)	3.06GHz*2	4 GB
Dell-Inspiron-5559	Ubuntu (V.16.04 LTS)	2.30GHz*4 Intel Core i5	8 GB
iMac-Early 2009	OS X El Capitan (V.10.11.6)	2.93 *2 GHz Intel Core Due	8 GB
MacBook Air	OS X El Capitan (V.10.11.3)	1.6 *2 GHz Intel Core i5	8 GB

We used two benchmarks of datasets from Chameleon (Fränti 2015). These are commonly used to test and evaluate clustering. Table 2 gives details about the datasets.

Table 2: Datasets

Benchmark	Size	Descriptions
D1	10,000 Points	Different shapes, with some clusters surrounded by others
D2	30,000 Points	2 small circles, 1 big circle and 2 linked ovals

The DDC approach is tested using various partitions of different sizes. Various scenarios were created

based on the goals of the experiments. These scenarios mainly differ on the way the datasets are divided among the processing nodes of the distributed platform. For each scenario, we recorded the execution time for the local clustering, the merging step including contour calculations, aggregation time and Idle time.). finally we capture also the total execution time that the approach takes to finish all the steps. in the following we describe the different scenarios considered.

5.1 Experiment I

In this scenario we give each machine a random chunk of the dataset, the size of the partition that was generated for each machine is in the range between 1500 points and 10000 points. As the dataset is relatively small we chose eight machines for the computing platform.

Table 3 shows the execution time taken by each machine to run the algorithm (step one and step two) using synchronous and asynchronous communications respectively, it also shows the overall time taken to finish all the steps.

From Table 3, we can see that the time taken by each machine to accomplish the first step of the algorithm is the same for both synchronous and asynchronous, whereas the time of the second step is different. We can also notice that each machine returns different execution time of the whole algorithm. This is because the machines have different capacities (see Table 1).

The total execution time of the algorithm while using synchronous communication is smaller compared to when using synchronous communication. This is because in synchronous communications, machines have more waiting time (up to 60% waiting time).

5.2 Experiment II

In this scenario we allocate the whole dataset size to one machine and the remaining machines were allocated one eight of the dataset each. This scenario is chosen to show the worst case of waiting time.

Table 4 shows the execution time taken by each machine to execute the DDC technique (step one and step two) using synchronous and asynchronous communications respectively, it also shows the overall time taken to finish all the steps.

From Table 4, we can notice that the difference between the execution times of the synchronous and asynchronous DDC is still significant. Because with synchronous communications the machines need to wait for the last machine to finish its first step before they all start merging their results (step 2), whereas for asynchronous model the seven machines did the merging (step2) while the last machine finishes its clustering (step1).

5.3 Experiment III

In this scenario we allocate to seven machines the whole dataset and the one machine was allocated one eight of the dataset. This scenario is chosen to show the effect of the complexity of the local clustering complexity on the machines and on the waiting time of some powerful machines.

Table 5 shows the execution time taken by each machine to run the algorithm (step one and step two) using synchronous and asynchronous communications respectively, it also shows the overall time taken to finish all the steps.

This scenario is the opposite of the previous scenario. Unlike the previous scenarios, Table 5 shows that the difference between the execution times of synchronous and asynchronous versions of the DDC is smaller. This is because in both cases the machines spend more time finishing the first step, therefore, the waiting time is less for synchronous over asynchronous model.

5.4 Experiment IV

In this scenario we took into account the machines capabilities and we divide the datasets according to their capacities. Therefore the work load is evenly distributed among them and we expect them to finish the first phase more or less at the same time. This allows to reduce the waiting time of the machines and follow immediately with the second phase. The total execution times of synchronous and asynchronous versions should be the same. This case favours more the synchronous implementation of the approach.

As predicted, Table 6 shows that there is no significant difference between the two execution times. Note that the little difference in favour of the synchronous version is due to the fact that in the asynchronous model the machines still need to execute the algorithm that checks which one finished first and receive the contours for merging.

5.5 Effective Speedup

The goal here is to compare our parallel clustering to the sequential algorithm and show the DDC speedup over the sequential version of clustering, as mentioned in Equation 3.

Considering the best scenario of executing the sequential version of DBSCAN on the fastest machine in the system. For instance, $T_1 = 15841$ ms. Clustering a partition of the same dataset on the same machine will take $= T_1^d = 258$ ms. The execution time of the DDC on the same datasets on eight heterogeneous machines with load balancing is $T_p = 1761$ ms (see Table 6). Therefore, from equation 3, we can deduce a speedup of 9, which is still a super-linear speedup. In the next section we will show how many processing nodes are required to cluster a dataset of size N.

5.6 Scalability

The goal here is to show that the DDC technique scales well and also we can dynamically determine the optimal number of processing nodes required to cluster a dataset of size N. We consider two datasets, the first dataset D_1 contains 10,000 data points and the second D_1 contains 30,000 data points. Figure 4 shows the execution time (y_axis is in log_2) against the number of machines in the system using the first dataset and Figure 5 shows the execution time (y_axis is in log_2) against the number of machines in the system using the second dataset contains 30,000 data points.

As one can see, from both Figures 4 and 5, the execution time of the first phase (Clustering and Contour) keeps decreasing as the number of machines in the distributed system increases. However, the time of the second phase (merging) keeps increasing gradually with the number of machines in the distributed system that is because the amount of communications in the second phase increases when the number of machines increases.

In addition, the total execution time of the algorithm (which is the sum of the two times, phase one

Table 3: Time (ms) taken by eight machines to run scenario I using synchronous and asynchronous communications

		Synchronous			Asynchronous		
Machine	**DS Size**	**STEP01**	**STEP2**	**Time**	**STEP1**	**STEP2**	**Time**
M1	10000	21270	1104	22374	21270	554	21824
M2	2500	1060	20862	21922	1060	2515	3575
M3	3275	5093	16930	22023	5093	2017	7110
M4	5000	4592	17644	22236	4591	2620	7211
M5	1666	227	21642	21869	227	391	618
M6	2000	292	21736	22028	292	416	708
M7	5000	7520	14665	22185	7515	13949	21464
M8	1500	200	21842	22042	195	4605	4800
		Total Exec-Time		**22374**	**Total Exec-Time**		**21824**

Table 4: Time (ms) taken by eight machines to run scenario II using synchronous and asynchronous communications

		Synchronous			Asynchronous		
Machine	**DS Size**	**STEP1**	**STEP2**	**Time**	**STEP1**	**STEP2**	**Time**
M1	10000	21270	973	22243	21270	595	21865
M2	1250	215	21775	21990	215	518	733
M3	1250	640	21383	22023	640	20100	20740
M4	1250	304	21730	22034	304	497	801
M5	1250	161	22034	22195	161	394	555
M6	1250	171	21856	22027	170	286	456
M7	1250	245	21918	22163	245	509	754
M8	1250	185	21854	22039	185	858	1043
		Total Exec-Time		**22243**	**Total Exec-Time**		**21865**

Table 5: Time (ms) taken by eight machines to run scenario III using synchronous and asynchronous communications

		Synchronous			Asynchronous		
Machine	**DS Size**	**STEP1**	**STEP2**	**Time**	**STEP1**	**STEP2**	**Time**
M1	10000	21270	35978	57248	21270	905	22175
M2	10000	21590	34869	56459	21590	11513	33103
M3	10000	53005	3008	56013	53005	3292	56297
M4	10000	32424	24691	57115	32424	6996	39420
M5	10000	17364	38493	55857	17364	4612	21976
M6	10000	15841	41237	57078	15841	2066	17907
M7	10000	38732	18483	57215	38727	18459	57186
M8	1250	185	56915	57100	184	16077	16261
		Total Exec-Time		**57248**	**Total Exec-Time**		**57186**

Table 6: Time (ms) taken by eight machines to run scenario IV using synchronous and asynchronous communications

		Synchronous			Asynchronous		
Machine	**DS Size**	**STEP1**	**STEP2**	**Time**	**STEP1**	**STEP2**	**Time**
M1	1500	256	1505	1761	256	1159	1415
M2	1660	260	598	858	260	1512	1772
M3	500	252	1061	1313	252	626	878
M4	1000	253	621	874	253	608	861
M5	1500	255	1492	1747	255	600	855
M6	1400	260	605	865	260	514	774
M7	1000	259	1030	1289	259	939	1198
M8	1500	250	603	853	250	1500	1750
		Total Exec Time		**1761**	**Total Exec Time**		**1772**

and two) keep decreasing as the number of processing nodes increases until it reaches a certain points where the total execution time starts to increase (at 8 machines for dataset D_1 and at 16 machines for dataset D_2). The optimal number of processing nodes required to execute DDC is returned when the overhead of the approach exceeds the execution time of the local clustering. This is a very interesting characteristic, as one can determine the number of machines that can be allocated in advance.

6 Conclusion

In this paper, we proposed an efficient and flexible distributed clustering framework that can work with existing data mining algorithms. The approach exploits the processing power of the distributed platform by

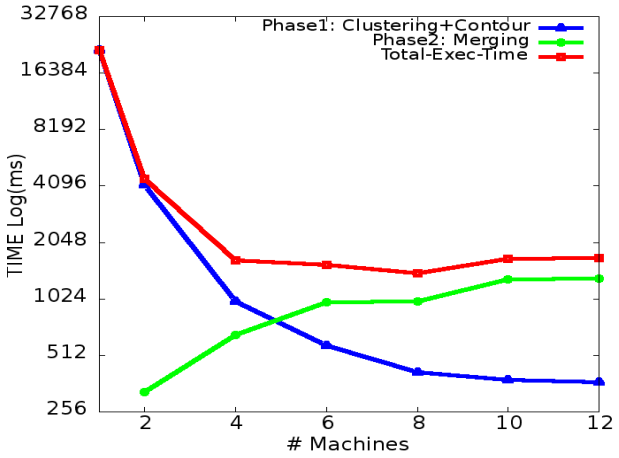

Figure 4: Scalability Experiment using dataset T_1.

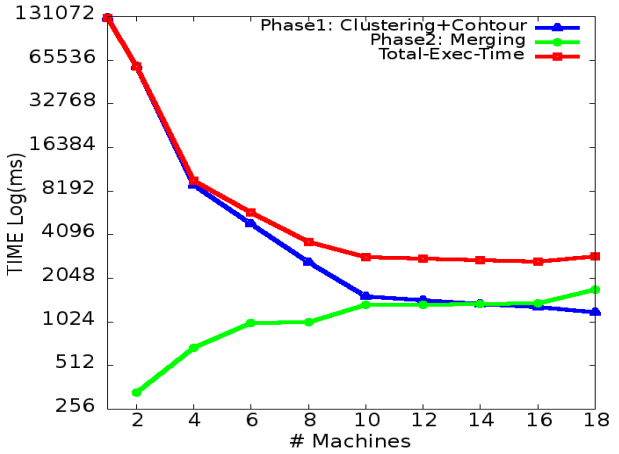

Figure 5: Scalability Experiment using dataset T_2.

maximising the parallelism and minimising the communications and mainly the size of the data that is exchanged between the nodes of the system. It is implemented using both synchronous and asynchronous communications, and the results were significantly in favour of the asynchronous model. The approach has an efficient data reduction phase which reduces significantly the size of the data exchanged therefore, it deals with the problem of communication overhead. The DDC approach has a super-linear speedup when the complexity of the local clustering has an NP complexity. We also can determine the optimal number of processing nodes in advance.

References

A. K. Jain, M. N. Murty, P. J. F. (1999), 'Data clustering: a review', *ACM Computing Surveys (CSUR)* **31**, 264–323.

Aouad, L., Khac, N.-A. L. & Kechadi, M.-T. (2007), *Advances in Data Mining. Theoretical Aspects and Applications: 7th Industrial Conference (ICDM 2007), Leipzig, Germany, July 14-18, 2007. Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter Lightweight Clustering Technique for Distributed Data Mining Applications, pp. 120–134.

Aouad, L., Le-Khac, N.-A. & Kechadi., M.-T. (2007), Image analysis platform for data management in the meteorological domain, *in* '7th Industrial Conference, ICDM 2007, Leipzig, Germany, July 14-18, 2007. Proceedings', Vol. 4597, Springer Berlin Heidelberg, pp. 120–134.

Arlia, D. & Coppola, M. (2001), Experiments in parallel clustering with dbscan, *in* 'European Conference on Parallel Processing', Springer, pp. 326–331.

Bellifemine, F., Bergenti, F., Caire, G. & Poggi, A. (2005), Jadea java agent development framework, *in* 'Multi-Agent Programming', Springer, pp. 125–147.

Bendechache, M. & Kechadi, M.-T. (2015), Distributed clustering algorithm for spatial data mining, *in* 'Spatial Data Mining and Geographical Knowledge Services (ICSDM), 2015 2nd IEEE International Conference on', IEEE, pp. 60–65.

Bendechache, M., Kechadi, M.-T. & Le-Khac, N.-A. (2016), Efficient large scale clustering based on data partitioning, *in* 'Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on', IEEE, pp. 612–621.

Bendechache, M., Le-Khac, N.-A. & Kechadi, M.-T. (2016), Hierarchical aggregation approach for distributed clustering of spatial datasets, *in* 'Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on', IEEE, pp. 1098–1103.

Brecheisen, S., Kriegel, H.-P. & Pfeifle, M. (2006), 'Parallel density-based clustering of complex objects', *Advances in Knowledge Discovery and Data Mining* pp. 179–188.

Chaudhuri, A., Chaudhuri, B. & Parui, S. (1997), '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.

Chen, M., Gao, X. & Li, H. (2010), Parallel dbscan with priority r-tree, *in* 'Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on', IEEE, pp. 508–511.

Coppola, M. & Vanneschi, M. (2002), 'High-performance data mining with skeleton-based structured parallel programming', *Parallel Computing* **28**(5), 793–813.

Cortese, E. (2005), 'Benchmark on jade message transport system', *URL: http://jade. cselt. it/doc/tutorials/benchmark/JADERTTBenchmark. htm* .

Dhillon, I. D. & Modha, D. S. (2000), A data-clustering algorithm on distributed memory multiprocessors, *in* 'Large-Scale Parallel Data Mining', Springer Berlin Heidelberg, pp. 245–260.

Dhillon, I. & Modha, D. (1999), A data-clustering algorithm on distributed memory multiprocessor, *in* 'large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD', Springer-Verlag London, UK, pp. 245–260.

Duckhama, M., Kulikb, L., Worboysc, M. & Galtond, A. (2008), '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.

Edelsbrunner, H., Kirkpatrick, D. G. & Seidel, R. (1983), 'On the shape of a set of points in the plane', *Information Theory, IEEE Transactions on* **29**(4), 551–559.

Eric Bauer, R. K. (1999), 'An empirical comparison of voting classification algorithms: Bagging, boosting, and variants', *springer Link:Machine Learning* **36**, 105–139.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. (1996), A density-based algorithm for discovering clusters in large spatial databases with noise., *in* 'Kdd', Vol. 96, pp. 226–231.

Fadilia, M., Melkemib, M. & ElMoataza, A. (2004), *Pattern Recognition Letters:Non-convex onion-peeling using a shape hull algorithm*, Vol. 24, ELSEVIER.

Fränti, P. (2015), 'Clustering datasets'.
URL: *http://cs.uef.fi/sipu/datasets/*

Fu, Y. X., Zhao, W. Z. & Ma, H. F. (2011), Research on parallel dbscan algorithm design based on mapreduce, *in* 'Advanced Materials Research', Vol. 301, Trans Tech Publ, pp. 1133–1138.

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

Ghosh, S. (2014), *Distributed systems: an algorithmic approach*, CRC press.

Guo, Y. & Grossman, R. (2002), 'A fast parallel clustering algorithm for large spatial databases, high performance data mining', *Data Mining and Knowledge Discovery* .

H.Geng, Omaha & Deng, X. (2005), A new clustering algorithm using message passing and its applications in analyzing microarray data, *in* 'ICMLA '05 Proc. of the 4th Int'l. Conf. on Machine Learning and Applications', IEEE, p. 145150.

Januzaj, E., Kriegel, H.-P. & Pfeifle, M. (2004), *Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology*, Springer Berlin Heidelberg, chapter DBDC: Density Based Distributed Clustering, pp. 88–105.

Jiawei Han, M. K. (2006), *Data Mining: Concepts and Techniques*, 2nd edn, Elsevier, Diane Cerra, San Francisco, CA 94111, chapter Introduction.

Laloux, J.-F., Le-Khac, N.-A. & Kechadi, M.-T. (2011), 'Efficient distributed approach for density-based clustering', *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 20th IEEE International Workshops* pp. 145–150.

Le-Khac, N.-A., Bue, M., Whelan, M. & M-T.Kechadi (2010), 'A knowledgebased data reduction for very large spatio-temporal datasets', *International Conference on Advanced Data Mining and Applications, (ADMA2010)* .

Melkemi, M. & Djebali, M. (2000), 'Computing the shape of a planar points set', *Elsevier Science* **33**, 14231436.

Moreira, A. & Santos, M. Y. (2007), Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points, *in* 'Int'l. Conf. on Computer Graphics Theory and Applications (GRAPP-2007)', Barcelona, Spain, pp. 61–68.

Rokach, L., Schclar, A. & Itach, E. (2014), 'Ensemble methods for multi-label classification', *Expert Systems with Applications* **41**, 7507 – 7523.

Solar, R., Borges, F., Suppi, R. & Luque, E. (2013), 'Improving communication patterns for distributed cluster-based individual-oriented fish school simulations', *Procedia Computer Science* **18**, 702–711.

Tian Zhang, Raghu Ramakrishnan, M. L. (1996), 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', Vol. 25, pp. 103–114.

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

Xu, X., Jger, J. & Kriegel, H.-P. (1999), 'A fast parallel clustering algorithm for large spatial databases', *Data Mining and Knowledge Discovery archive* **3**, 263–290.

Zaki, M. J. (2000), *Large-Scale Parallel Data Mining*, Springer Berlin Heidelberg, Berlin, Heidelberg, chapter Parallel and Distributed Data Mining: An Introduction, pp. 1–23.

Zhou, A., Zhou, S., Cao, J., Fan, Y. & Hu, Y. (2000), 'Approaches for scaling dbscan algorithm to large spatial databases', *Journal of computer science and technology* **15**(6), 509–526.