25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises

Parallelizing DBSCAN Algorithm Using MPI

Ilias K. Savvas[1], and Dimitrios Tselios[2]

[1] Dept. of Computer Science and Engineering,
[2] Dept. of Business Administration,
T.E.I. of Thessaly, Larissa, Greece.
[1] savvas@teilar.gr, [2] tselios@teilar.gr

Abstract

The last years, huge bundles of information are extracted by computational systems and electronic devices. To exploit the derived amount of data, new innovative algorithms must be employed or the established ones may be changed. One of the most fascinating and productive techniques, in order to locate and extract information from data repositories is clustering, and DBSCAN is a successful density based algorithm which clusters data according its characteristics. However, its main disadvantage is its severe computational complexity which proves the technique very inadequate to apply on big datasets. Although DBSCAN is a very well studied technique, a fully operational parallel version of it, has not been accepted yet by the scientific community. In this work, a three phase parallel version of DBSCAN is presented. The obtained experimental results are very promising and prove the correctness, the scalability, and the effectiveness of the technique.

Keywords—clustering; parallel clustering; DBSCAN; parallel DBSCAN; MPI

I. Introduction

Data Mining is the term that expresses the extracting process of useful information from large datasets. Two well known clustering techniques of Data Mining are DBSCAN [1] and K-means algorithm [2]. The latter's inputs are a predefined number of clusters and a given dataset. The main idea behind K-means is that assigns the data objects to clusters according to produce one way or another k clusters. However, the operation of k-means algorithm on very large datasets needs a lot of computational time. This fact makes the algorithm unproductive.

On the other hand, DBSCAN is fed by only two inputs besides the dataset, the minimum accepted distance between two objects to be considered as neighbours, and the minimum number of them, in order to form a cluster. Thus, DBSCAN gives very high quality clusters. However, DBSCAN also suffers from severe computational time issues. Hence, some recent researchers have tried to change the above mentioned algorithms, in order to accelerate them. Some of these efforts are parallel versions of DBSCAN.

The Message Passing Interface - MPI provides the tools to send and receive data from one node to other nodes (send and receive operations), synchronization mechanisms (barrier operation) and combining results obtained by the worker nodes (reduce operation) which is similar to the MapReduce corresponding operation [3]. In addition, MPI includes many functions to obtain network's information like the number of participating nodes, the name of each node an so on. Finally, MPI can support various virtual network topologies and peer-to-peer operations can work on both synchronous and asynchronous ways. The number of the parallel working nodes is unlimited. Therefore, MPI is able to operate on terrabytes of data and as large number of worker nodes is needed.

The main idea of this work is the formulation of a parallel version of DBSCAN. This novel approach consists of three phases, i.e. the splitting phase, the pure DBSCAN execution and the forming of the final clusters. The proposed technique is explained by a fully described concrete example and then the accompanied experimental results are presented.

The rest of the paper is organized as follows. In Section II a brief overview of related work based on DBSCAN is given. The system model is presented in Section III, while in Section IV, we describe the original DBSCAN and in Section V its parallel version proposed in this study. In Section VI a detailed concrete example is provided in

978-1-5090-1663-1/16 $31.00 © 2016 IEEE
DOI 10.1109/WETICE.2016.26

order to clarify the technique. The experimental results are presented in Section VII. Finally, Section VIII concludes the paper and highlights the future research directions.

II. Related Work

DBSCAN [1] is a significant clustering technique that has been utilized extensively by various disciplines the last decades. However, the huge size of the collected data is a severe impediment for the researchers who aim to employ DBSCAN as their main solving tool. Januzaj et al. [4] indicate the drawbacks of the typical DBSCAN. These are the avoidance of local noise, the predefined numbers of the core-points, and the proximity of the core-points to the clusters' border.

Recently, some research works [1], [5], [6] have tried to overcome this issue by proposing parallel based variations of DBSCAN algorithm. Despite the obvious improvement that is observed when this amendment is applied, there are three problematic issues that arose in the proposed parallel approaches [1].

According to authors of interesting works [1], [5], the size of the datasets impact dramatically the processing time of the algorithm and they proposed a parallel alteration of the method. Moreover, the same research team [1], [5] highlights the main challenges of the parallel DBSCAN, i.e. the difficulty of the load balancing, the scalability of the algorithm, and the complexity of the parallel form. Finally they proposed a cost-based data method, in order to obtain a fair load balancing solution asserting that their approach is the first endeavour that gives a really parallel DBSCAN algorithm that is based on MapReduce technique.

Since its importance and its great computational complexity, many researchers in the past tried to parallelize DBSCAN algorithm. In [7], the authors employed the disjoint-set data structure to break the access sequentiality of DBSCAN and then a tree-based bottom-up approach used to construct the clusters. The goal of this research was to achieve a better-balanced workload distribution. The parallel DBSCAN algorithm was implemented in C++ both using OpenMP and MPI to run on shared-memory machines and distributed-memory machines, respectively. However, this approach does not fully resolve the problem in a parallel way. Generally, scientists have been using MPI on Grid Virtual Organizations, developing models to improve QoS on the Grid by organizing better Virtual Organizations [8], [9].

Januzaj et al. [10] propose a variation of DBSCAN algorithm that can be considered as a parallel form. Their approach is based on a trade-off between clustering quality and quantity of the transmitted data from distant local points to a server site. More specifically, the proposed improvement of DBSCAN consists of three consecutive phases, i.e. a preprocessing phase in local level, the sending phase that submits the representatives of the clusters from local sites to the server, and then the third phase returns the results to local sites, in order to update the clustering.

Moreover, another research work [11] tries to eliminate two drawbacks of the pure DBSCAN algorithm, i.e. the scalability and the complexity by mixing it with MapReduce, a method within cloud computing [12], [13].

In conclusion, none of the above mentioned works offers a parallel version of DBSCAN algorithm that is generally accepted from academia and practitioners. This is the main goal of the present paper.

III. System Model

The parallel model used in this work is the Single Instruction Multiple Data (SIMD). Therefore, each computational node of the system performs the same task on different data. Data resides on the master node which is responsible to split it and transfer it to the worker nodes which in turn they will perform independently the DBSCAN algorithm. The technique is applied on a computational cluster. A Computational Cluster ($V = \{v_0, v_2, \ldots, v_{P-1}\}$) is a collection of independent and potentially heterogeneous processing-nodes. We assume that each node v_i is autonomous, and has a full information on its own resources.

Finally, the data set D is divided into $P-1$ subsets, d_i, $D = \{d_0, d_1, \ldots, d_{P-2}\}$ of equal or almost equal size (considering that mode v_0 is the master node and no computations are performed on it). Each worker node v_i receives its data set d_i and applies the original sequential DBSCAN plus some more calculations in order to calculate the centroids (the arithmetic mean of all points of the cluster) and radiuses of each cluster. As a result, the worker nodes produce L_i lists of local centroids C_l. Therefore, each node has the information $v_i = (c_l^i, r_l^i)$, where c_l^i stands for the l^{th} centroid from node v_i and r_l^i represents its radius.

IV. DBSCAN

The original sequential DBSCAN algorithm [14], is a density-based spatial clustering technique which groups data points together according their attributes. In addition, DBSCAN takes as input the desired longest distance between two points, in order to be considered as neighbours (known as *eps*) and the minimum number of points that can be grouped together and considered as autonomous cluster (known as *MinPts*). The metric of distance used in this work is the Euclidean distance. Finally, one major advanage of DBSCAN is that characterizes the data points as follows:

1) Core: the points that belong to a cluster and in addition have larger number of neighbours than *eps*,
2) Border: the points that belong to a cluster but have less number of neighbours than *MinPts*. These points belong to the cluster because at least one neighbour of them belong to this cluster and it is core point to the specific cluster,

3) Noise: the points that do not belong to any cluster.

A brief description of the original DBSCAN algorithm is given in Algorithm 1.

Algorithm 1 DBSCAN

```
for each data point d_i ∈ D do
  if d_i is unvisited then
    Mark d_i as visited
    NumPts ← explore the rest of unvisited data points and examine how many and which of them are neighbours to d_i
    if NumPts < MinPts then
      Mark d_i as Noise
    else
      Generate new cluster containing d_i and its neighbours
      Mark d_i as Core
      Determine all the rest data points that belong to the cluster if they are either core or border points
    end if
  end if
end for
for each data point d_i characterized as noise do
  Examine if d_i belongs to any cluster
  if d_i belongs to a cluster then
    Mark d_i as Noise
  end if
end for
```

The main disadvantage of DBSCAN is its computational complexity where its worst time complexity is $O(N^2)$ [14]. So, the goal of this work is to parallelize the technique in order to reduce this high complexity. Other disadvantages of DBSCAN like the curse of dimensionality or the choice of meaningful parameters (eps and $MinPts$) are beyond the scope of this study.

V. Parallel DBSCAN

The main idea to parallelize DBSCAN is to split the data set to sub-sets and at the beginning, on each dataset to perform DBSCAN independently and in addition to the original DBSCAN the centroids of the clusters and their corresponding radiuses are calculated. Therefore, each processing node (v_i) produces its own local clusters, $c_{i_0}, c_{i_1}, \dots c_{i_n}$. Each cluster is characterized by its centroid and radius. So, each node produces a list of pairs $F_i = \{(c_{i_0}, r_{i_0}), (c_{i_1}, r_{i_1}), \dots, (c_{i_n}, r_{i_n})\}$. Then, each worker node sends its list to its neighbour node ($v_i \rightarrow v_{i+1}$) which in turn explores if these form common clusters. The goal is to find circles which their intersection is not the empty set. If there are, then calculates the points that intersect and ask from the corresponding worker node to send the data points that belong to that space. This procedure repeats till all the information will be on the final worker node.

To examine if these clusters could be grouped into bigger clusters, the circles that represent them are examined. Let $C_1 = (c_1, r_1)$ and $C_2 = (c_2, r_2)$ two circles and d be the distance between their centrers, and let $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$ represent the coordinates of their centrers in a 2-dimensional Euclidean space. Then, the relative position of them and the action must taken by the appropriate node is as follows:

1) one circle lies completely inside the other or touch internally *iff* $d \leq |r_1 - r_2|$: they may form one bigger cluster
2) intersect in two points *iff* $d > |r_1 - r_2|$ and $d < r_1 + r_2$: identify the points and the radical line
3) touch externally *iff* $d = r_1 + r_2$: identify the point they touch
4) do not intersect *iff* $d > r_1 + r_2$: do nothing

In order to identify the point or points of intersection, the corresponding node must solve the following equations:

$$\begin{aligned}(x - x_1)^2 + (y - y_1)^2 &= r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 &= r_2^2\end{aligned} \quad (1)$$

If the circles touch each other externally the possible point which may be responsible to form a common cluster are all these points which belong to a circle of centre that point and radius equal to *eps*.

If the circles intersect in two points, then the candidate points to be examined are all these points of both corresponding clusters within the rectangle defined by these points adding again *eps* to each coordinate.

If one circle lies completely inside the other or touch internally then if even one core point from one cluster is within *eps* with a core point from the other cluster then these two cluster can form a bigger one witch includes all the data points from both clusters. Otherwise, the examination has to be extended to the core points and the result will be that either they do or do not form a new cluster.

Finally, if the circles do not intersect then there is no action since these form different clusters. The parallel version of DBSCAN is given in Algorithm 2

The computational complexity of the algorithm (taking under consideration the complexity of the original technique) is depending on the number of the participating worker nodes and the communication overhead needed to transfer the data subsets to them and can be expressed as in Equation 2.

$$T_P = O(\frac{N^2}{P-1} + C) \quad (2)$$

where C stands for the communication overhead and P represents the number of participating nodes.

VI. Concrete Example

In order to clarify the technique, a simplified concrete example is given. Three computational nodes were used

Algorithm 2 Parallel DBSCAN

1: Master node, v_0 collects the number of available worker nodes, $(P-1)$
2: v_0: splits data set D into $D/(P-1)$ subsets
3: v_0: transfers data, eps and $MinPts$ to worker nodes
4: **for all** Worker nodes $v_i, v_i \in \{v_1, v_2, \ldots, v_{P-1}\}$ **do in parallel**
5: Receive data set
6: Apply DBSCAN algorithm
7: **end for**
8: i $\leftarrow$ myid {node identification number}
9: **while** $(i \leq \log(P-1))$ **do**
10: **for all** $v_i, v_i \in \{v_1, v_2, \ldots, v_{P-1}\}$ **do in parallel**
11: **if** $(i \ mod \ 2 = 1)$ **then**
12: Send the local centroids and the corresponding radius's F_i, to v_{i+1} {node is sender}
13: **else**
14: Receive F_{i-1}, from v_{i-1} {node is receiver}
15: Reform clusters
16: **end if**
17: **end for**
18: $i \leftarrow 2^i - 1$
19: **end while**
20: Last worker node transfers the clusters to master node
21: Master node, v_0 receives C

(the master node and two worker nodes), therefore the dataset (forty 2-d data points within the range $1 \rightarrow 100$) was divided into two subsets. After the application of DBSCAN at the worker nodes, each one of them produced its own clusters. The parameters used were $eps = 30$ and $MinPts = 3$. The centroids and the radiuses produced are presented in Tables I, and II, while the corresponding clusters are shown as in Figures 1, and 2.

TABLE I: Worker 1.

Cluster	Centroids	Radius	# of data points
$C_{1,0}$	83.00 , 77.00	18.79	4
$C_{1,1}$	66.85 , 28.77	44.96	13
Noise			3

TABLE II: Worker 2.

Cluster	Centroids	Radius	# of data points
$C_{2,0}$	33.22 , 69.78	29.80	9
$C_{2,1}$	25.20 , 21.80	23.29	5
$C_{2,2}$	88.00 , 34.25	14.37	4
Noise			2

After the application of DBSCAN at the worker nodes, the worker #1 transfers its dataset, and the list of centroids with the corresponding radiuses to worker #2 which in turn tries to unite (or not) these clusters. While eps was the same as on worker nodes, the *MinPts* doubled to 6 since the double size of data set was examined by worker #2.

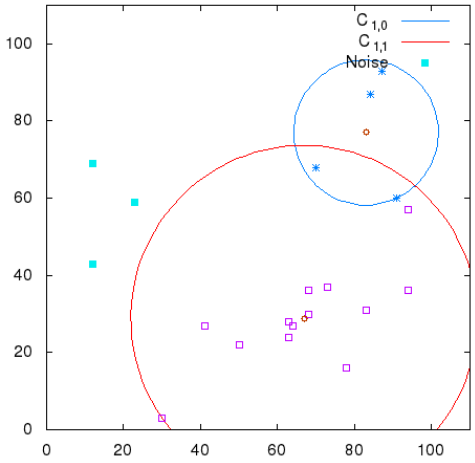

Fig. 1: Concrete example: Worker node #1

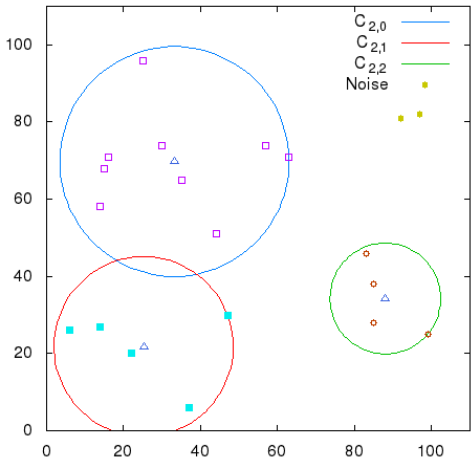

Fig. 2: Concrete example: Worker node #2

Initially, worker #2 has to examine the relative positions between all clusters in order to examine the possibility to form larger ones (Table III).

After that, worker #2, firstly starts to examine the clusters where the corresponding circles lie one of them inside the other to explore if new larger clusters are produced. If it is so, the new centroid and radius is calculated and replace the clusters that already have unite and the procedure continues till no other cluster can form larger one's.

So, after the examination, clusters $C_{2,2}$ and $C_{1,1}$ produce a biggest one with circle and radius as in Table IV. In the same table, the second union is presented which is the result of the union of the clusters $C_{2,1}$ and $Union_1$.

TABLE III: Worker #2: Examining circles. *int-2p* stands for that the corresponding circles intersect in 2 points, *not int* for not intersection, and *lies ent* for that one circle lies entirely into the other

Clusters	Distance	$\lvert r_1 - r_2 \rvert$	$r_1 + r_2$	result
$C_{1,0}C_{2.0}$	53.09	11.01	48.59	not int
$C_{1,0}C_{2.1}$	79.92	4.50	42.08	not int
$C_{1,0}C_{2.2}$	42.50	4.42	33.16	not int
$C_{1,1}C_{2.0}$	53.04	15.16	74.76	int - 2p
$C_{1,1}C_{2.1}$	42.23	21.67	68.25	int - 2p
$C_{1,1}C_{2.2}$	21.99	30.59	59.33	lies ent

TABLE IV: Worker #2: first union.

Clusters	Centre	Radius
$Union_1 \leftarrow C_{1,1} \cup C_{2.2}$	71.82 , 30.06	49.81
$Union_2 \leftarrow Union_1 \cup C_{2.1}$	61.23 , 28.18	55.27

Since no other unions are possible, worker node #2 examines the noise points if now they belong or not to the new formed clusters. In this case they do and the final results are shown in Table V and in Figure 5. Finally, worker node #2 transfers the result to the master node. Note, that in this case the role of worker #2 could play the master node reducing the communication overhead but we would like to make clear the whole procedure as it would work in a big computational cluster with many participating nodes.

TABLE V: Worker #2: Final results.

Cluster	Centroids	Radius	# of data points
0	81.67 , 74.78	24.68	9
1	59.67 , 26.81	53.67	21
2	22.60 , 65.40	30.69	10

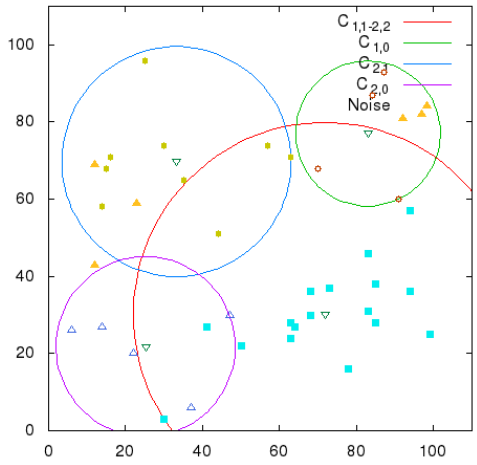

Fig. 3: Concrete example: worker #2, first union

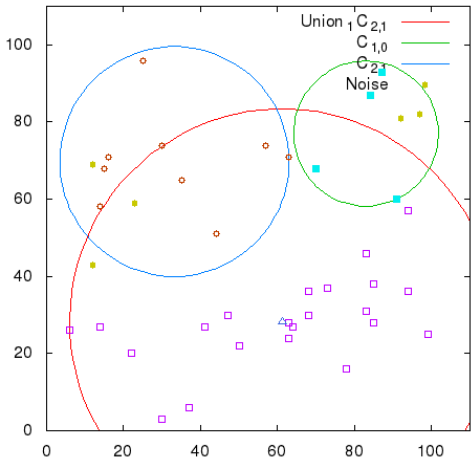

Fig. 4: Concrete example: worker #2, second union

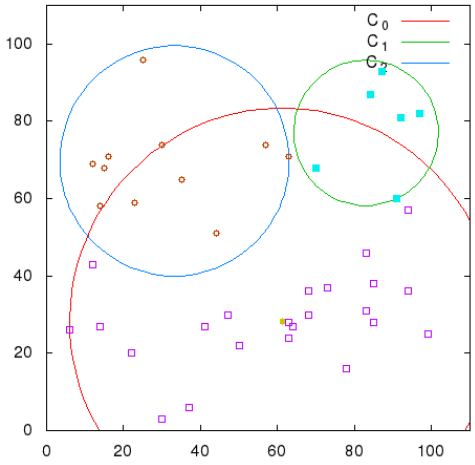

Fig. 5: Concrete example: worker #2, final custering

VII. Experimental Results

For the experiment of this work, 33 computational nodes were used ($Intel^{@}Core^{TM}$ i5, 2.60GHzx4, with UBUNTU 14.04 LTS operating system and MPICH 3.2 and Ethernet 100 Mbit/s). The number of data points varied from 5,000 to 100,000 (2-d data points) produced by the random generator of C programming language between 0 and 100,000 and the time needed is expressed in seconds.

The results obtained proved the efficiency and scalability of the proposed technique (Table VI and Figure 6). In particular, the parallel algorithm outperform the sequential one when for example $N \geq 60,000$ and the participating nodes $P = 3$. Note, that in Table VI, the

crossing points where increasing the participating nodes the technique outperforms the previous model with less nodes are denoted in bold.

TABLE VI: Experimental results (in seconds).

N	$P=1$	$P=3$	$P=5$	$P=9$	$P=17$	$P=33$
5000	0.05	2.27	2.39	2.44	2.47	2.49
10000	0.63	3.81	3.91	3.95	3.98	3.99
15000	0.99	5.25	5.37	5.44	5.47	5.48
20000	1.80	6.90	6.95	6.97	6.99	6.99
25000	2.69	8.60	8.55	8.52	8.51	8.51
30000	4.04	10.52	10.26	10.13	10.07	10.03
35000	6.06	12.78	12.14	11.82	11.66	**11.58**
40000	9.09	15.55	14.27	13.64	**13.32**	13.16
45000	13.64	19.07	16.78	**15.64**	15.07	14.79
50000	20.46	23.73	**19.86**	17.93	16.97	16.48
55000	30.69	30.09	23.80	20.65	19.07	18.29
60000	**46.03**	**39.01**	29.01	24.00	21.50	20.25
65000	69.04	51.77	36.14	28.32	24.41	22.45
70000	103.57	70.28	46.14	34.07	28.04	25.02
75000	155.35	97.42	60.46	41.98	32.74	28.12
80000	233.02	137.51	81.26	53.13	39.06	32.03
85000	349.54	197.02	111.76	69.13	47.81	37.16
90000	524.30	285.65	156.83	92.41	60.21	44.10
95000	786.46	417.98	223.74	126.62	78.06	53.78
100000	1179.69	615.84	323.42	177.21	104.11	67.55

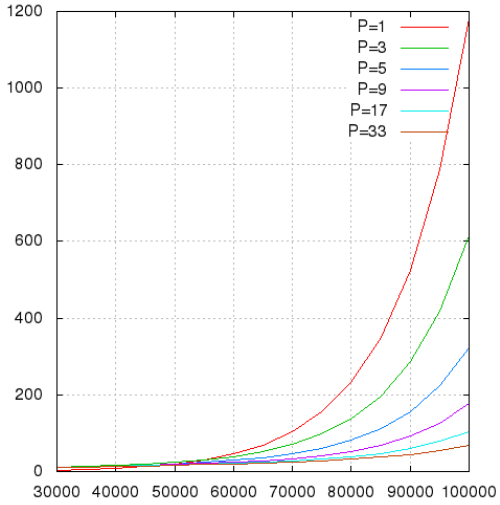

Fig. 6: Comparative results

VIII. Conclusions and Future Work

In this work, a parallel version of the well-known DBSCAN was presented and implemented using MPI. The results obtained from different concrete examples proved that were identical with the results produced by the application of the original sequential technique. In addition, the time complexity reduced dramatically and the experimental results shown that the algorithm scales in a very efficient manner.

The future work includes three main goals, proof of correctness, calculation of the time complexity, and comparisons. Firstly, we will prove theoretically the correctness of the proposed technique while we have to estimate accurately its time complexity as a function of the participating nodes and the size of the data set. Finally, we will compare our technique with other similar algorithms even if this is not fair, since the proposed technique is the only one found in the literature where the data set is transferred by the nodes instead of transferring the computations to the data sets.

References

[1] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "Mr-dbscan: A scalable mapreduce-based dbscan algorithm for heavily skewed data," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 83–99, 2014.

[2] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *5th Berkeley Symposium on Mathematical Statistics and Probability*, U. of California Press, Ed., 1967, pp. 281–297.

[3] MPICH, "Message Passing Interface," http://www.mpich.org/, [Online; accessed 2014].

[4] M. Goetz, M. Richerzhagen, C. Bodenstein, G. Cavallaro, P. Glock, M. Riedel, and J. Benediktsson, "On scalable data mining techniques for earth science," vol. 51, no. 1, 2015, pp. 2188–2197.

[5] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce," 2011, pp. 473–480.

[6] M. Patwary, D. Palsetia, A. Agrawal, W.-K. Liao, F. Manne, and A. Choudhary, "A new scalable parallel dbscan algorithm using the disjoint-set data structure," 2012.

[7] M. M. A. Patwary, D. Palsetia *et al.*, "A new scalable parallel dbscan algorithm using the disjoint-set data structure," in *The international Conference for High Performance Computing, Networking, Storage and Analysisi - SC12*, 2012.

[8] P. De Meo, F. Messina, D. Rosaci, and G. M. Sarné, "An agent-oriented, trust-aware approach to improve the qos in dynamic grid federations," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5411–5435, 2015.

[9] A. Comi, L. Fotia, F. Messina, D. Rosaci, and G. M. Sarnè, "A qos-aware, trust-based aggregation model for grid federations," in *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*. Springer Berlin Heidelberg, 2014, pp. 277–294.

[10] E. Januzaj, H.-P. Kriegel, and M. Pfeifle, "Scalable density-based distributed clustering," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3202, pp. 231–244, 2004.

[11] B.-R. Dai and I.-C. Lin, "Efficient map/reduce-based dbscan algorithm with optimized data partition," 2012, pp. 59–66.

[12] R. Giunta, F. Messina, G. Pappalardo, and E. Tramontana, "Providing qos strategies and cloud-integration to web servers by means of aspects," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 6, pp. 1498–1512, 2015.

[13] F. Messina, G. Pappalardo, D. Rosaci, C. Santoro, and G. Sarné, "A trust-aware, self-organizing system for large-scale federations of utility computing infrastructures," *Future Generation Computer Systems*, vol. 56, pp. 77–94, 2016.

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996, pp. 226–231.