Received: October 22, 2022. Revised: January 21, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Parallel Implementation of Statistical DBSCAN Algorithm for Spark-based Clustering on Google Cloud Platform

Ahmad M. Awaad[1]* **Hesham Hefny[1]**

[1]*Computer & Information Sciences Department, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza, Egypt*
* Corresponding author's Email: eng.ahmadawaad@gmail.com

Abstract: We present a new parallel density-based spatial clustering of applications with noise (DBSCAN) algorithm for spark on the google cloud platform (GCP). Statistical analysis is applied to determine DBSCAN's optimal parameters to enhance clustering performance. for scalability cost-based, R-tree partitioning is selected based on the distribution of the dataset into balanced workloads. Parallel DBSCAN consists of three parts: local DBSCAN, partitioning, and merging. Optimizing the partitioning of parallel DBSCAN is important to save time and space compared to serial DBSCAN. This approach can improve the performance and time cost of large datasets. the modified statistical cost-based (SCbs-DBSCAN) is applied to the UCI (university of california irvine) standard datasets, basic benchmark clustering and large different scales of data. For clustering performance and time cost, the experimental results show that the proposed algorithm achieve 10~15% more efficiently, and can run about 1.5x~3x faster than alternative Parallel DBSCAN method on Spark without sacrificing clustering quality.

Keywords: Spark, Data mining, Parallel algorithms, DBSCAN algorithm, Data partition.

1. Introduction

One of the most fascinating technologies of our era is the Internet of Things (IoT). The use of intelligent and self-configuring devices distinguishes the IoT. Objects that can communicate with one another over a network's global architecture. As a result of these interactions between different sets of objects, the IoT is portrayed as a disruptive technology that enables computing applications that are ubiquitous and pervasive [1]. As a result, a different set of industrial IoT applications [2] have been developed and deployed in a variety of industries, including transportation, agriculture, and food processing, as well as health monitoring systems, environmental monitoring, and security monitoring.

Big data necessitates various approaches for dealing with it to extract value via four elements: -Dimensions: size is determined by the amount of data collected and produced. -Diversity: a wide range of data sets, including large structured, semi-structured, and unstructured datasets, Speed is determined by the schedule between information technology and processing. -Validity: validity means whether or not the records are valid [3-5].

Clustering is a data mining technique for categorizing information as intelligible, useful, or both [6]. Bioinformatics, machine learning, information retrieval, and statistics are just a few of the disciplines where cluster analysis has proved successful [6]. K-means [7], BIRCH [8], Wave Cluster [9], and DBSCAN [10] are all well-known algorithms. Partitioning-based, hierarchy-based, grid-based, and density-based clustering algorithms are the four types of modern clustering algorithms [10]. The DBSCAN algorithm [10] is a density-based clustering technique that can detect and remove noisy data from clusters of any shape.

Because clustering is a fundamental operation for Big Data processing and analytics, this paper focuses on a specific and critical component of big data, clustering algorithms in Big Data. In this study, we discussed clustering algorithms in big data and explained how clustering techniques in big data can be applied to IoT.

Parallelization is difficult in DBSCAN due to the sequential data access order, and depending on MPI or OpenMP settings, there are problems regarding fault tolerance and no guarantee that workloads are balanced. Furthermore, data scientists must manage the connectivity between nodes while programming with MPI, which is a significant barrier. MPI and OpenMP have been used to implement parallel DBSCAN [11,12]. In general, an MPI implementation can improve performance, but it requires programmers to pay close attention to implementation details such as data partitioning, communication, synchronisation, file location, and workload balancing. Aside from MPI parallelization, a MapReduce-based technique is also presented [13-15].

However, parallel DBSCAN methods have previously been suggested by numerous academics [16]. The four issues are listed below: First, when the amount and distribution of data points in each sub-region are taken into account, a region-splitting technique can quickly become troublesome. The cost of such a split increases as the number of dimensions in a data set increases. Second, despite the sophisticated area split mechanism, DBSCAN execution time varies significantly between sub-regions due to highly skewed data distributions.

Third, due to sub-regional overlaps, the overall number of data points processed is always greater than the total number of data points handled in all sub-regions. The execution time rises in lockstep with the amount of data. Furthermore, the cost of merging clusters is relatively high due to the massive amount of duplicated data points. Fourth, there is a fundamental issue with picking the appropriate two input parameters: the Eps (ε) radius and the minPTs density threshold. When cluster density variation is high, selecting these criteria becomes incredibly difficult.

This study aims to improve density-based clustering techniques using DBSCAN [10]. However, due to the giant volume of records to be processed, the operational effectivity and data storage of prevalent sequential DBSCAN can be a problem. Our thinking is to beef up a parallel model of DBSCAN based totally on SPARK. After developing DBSCAN in parallel, we used a variety of information constructions to increase the clustering algorithm's performance. To increase the efficiency of the parallel algorithm, we tried two approaches specifically based on the idea of the R-Tree to attain a more balanced partition. Boundary-based and cost-based partitions Finally, we assessed the effectiveness of the cost-based partition implementation strategies and carried out statistical analysis [11] to obtain the optimal DBSCAN parameters (minPTs and epsilon), which affect clustering performance, where in [11] a baseline Monte Carlo method is applied to estimate the significance of clusters and a dual-convergence algorithm to accelerate the computation.

Some of the primary contributions to this work are:

- Decoding and executing the Parallel DBSCAN algorithm for spark cluster on google cloud platform (GCP) called SCbs-DBSCAN.
- SCbs-DBSCAN uses statistical analysis to obtain the optimum DBSCAN parameters to enhance DBSCAN clustering performance and cost-based R-trees to optimize partitioning and merging for efficient cost time.
- Study the effectiveness and scalability of the proposed clustering algorithm SCbs-DBSCAN using Basic benchmark clustering UCI standard datasets, and different scales of the data sample.
- Time costs for partitioning, parallel DBSCAN, and merging are computed. Silhouette coefficient (SC), completeness (Comp.), and adjusted rand index (ARI) are used to measure performance.

The content of this paper is represented by the following sections: The applicable section 2 includes related work on existing parallel DBSCAN models. Section 3 describes the DBSCAN Algorithm and the Spark Cluster. Section 4 describes Parallel DBSCAN and the enhancement partitioning method. Section 5 compares our optimised parallel DBSCAN to other serial and parallel DBSCANs for time cost and performance. Section 6 concludes the whole work.

2. Related work

In a statistical set, clustering is the process of grouping like objects together. Clustering is divided into a plethora of types, each of which necessitates an iterative procedure, making it unsuitable for large-scale data processing. As a result, the single-traffic-scale evolving clustering method (ECM) had to be transformed into a parallel clustering methodology (PECM) capable of handling large amounts of data [17]. PECM (parallel evolving clustering method) is a statistics evaluation technique that runs in the Apache spark framework and leverages HDFS (Hadoop distributed file system) for statistics storage [18].

In addition, DBSCAN is presented in parallel with the Spark platform in [19]. Spark is a new generation of large-scale data processing speeds, achieving real-time RDD (resilient distributed dataset), being highly scalable, and having high fault tolerance. DBSCAN was paralleled with the single-node and cluster Spark platforms, and it was found that when dealing with applications with many iterations, spark-yarn is more suitable than spark-mesos.

It is proposed in [20] to apply parallelism between DBSCAN and Apache spark by applying it to data processed by KD-tree (K-dimensional tree) to reduce search time, which aims to organise spaces between data, and HashMap, which is one of the Java tools for indexing data by storing it in the form of values and keys, allowing faster access to the required data.

In an Apache spark framework that supports large-scale cloud resources, the process in [21] was directed to give a parallel version of DBSCAN and real-time factor realisation based on cluster checkpoints with temporal and spatial data sequencing dubbed RT-DBSCAN.

He et al. [22] introduced the MR-DBSCAN algorithm based on the MapReduce model, which introduces a data partitioning strategy to increase the efficiency of the algorithm. Cordova et al. [23] propose RDD-DBSCAN. It is a kind of DBSCAN algorithm based on RDD and DBSCAN-PSM [24], which applies a new data partitioning and merging method. In [25] a parallel adaptive DBSCAN (PDBSCAN) algorithm is proposed based on a k-dimensional tree partition and carries out parallel computing in spark distributed computing framework. At the same time as creating local clusters, this algorithm also puts the mapping relationship between data points and adjacent points into the HashMap data structure of the master node and uses it to merge local clusters into whole clusters, which can reduce the time cost of data merging.

The approach in [26] attempts to fully parallelize the DBSCAN implementation by utilizing HPCC systems' ideal distributed architecture and executing a tree-based union to combine local clusters.

From the presented related work, it is clear that:

1. Iteratively, spark is the best choice in the fields of machine learning and data mining with iterations.
2. The experimental performance of some algorithms with spark is 100 times higher than with MapReduce.
3. The data file can be downloaded and used frequently through it, with the results of intermediary operations being stored directly in memory rather than in HDFS.

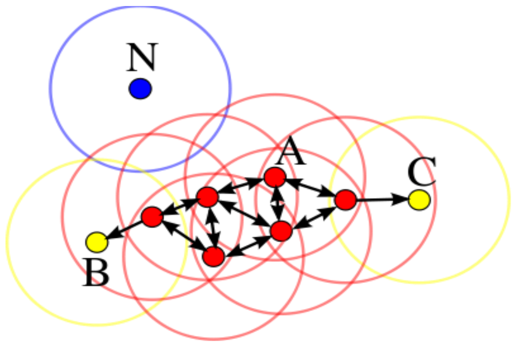


Figure. 1 DBSCAN method

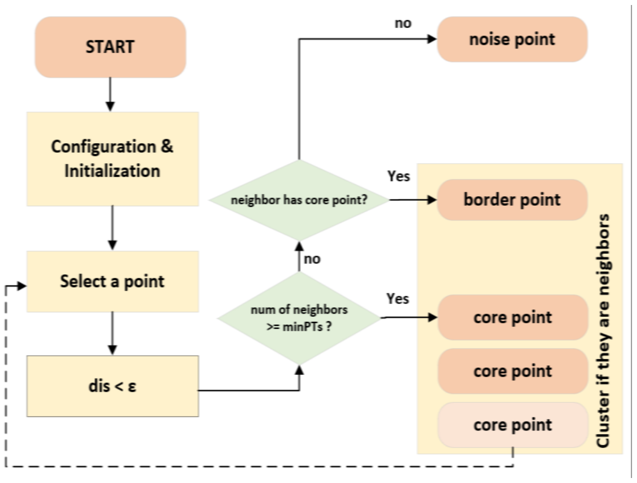


Figure. 2 DBSCAN Flow chart

3. DBSCAN algorithm

this section, we'll go through the basics of the DBSCAN algorithm. The spark distribution system was then introduced.

A. DBSCAN algorithm

Ester [11] proposed the DBSCAN clustering algorithm. Because it can discover arbitrarily shaped clusters and reduce noisy data, it has become one of the most widely used clustering methods. The fundamental idea behind this method is to discover all of the core points and then group them with all of the extra locations (core or noncore) that can be reached from them to form clusters.

In Fig. 1. An Overview of DBSCAN method, minPTs = 4. Point A and the other red points are core points because the area surrounding these points in an ε radius contains at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly reachable.

Han [27] provides the pseudocode for the DBSCAN algorithm. The DBSCAN flow chart is illustrated in Fig. 2. The procedure starts with a random point $p \in D$ and checks its eps-neighborhood (ε). If the ε size is greater than a predetermined number of minPTs, the function constructs a new cluster C. The program then retrieves all the densely accessible places from p in D and adds them to cluster C. If the eps neighbourhood has fewer than minPTs points, p is considered noise. The computational complexity of the algorithm is $O(n^2)$, where n is the number of data points. When spatial indexing is employed, the complexity is lowered to $O(nlogn)$ [3].

B. Spark:

A Spark application's driver process communicates with many executor processes, giving tasks and receiving results. The first step in a spark application is to construct a spark context object in the driver code, which instructs Spark on how to connect to a cluster. Several HDFS files are read and processed as RDDs, which are collections of elements partitioned across nodes and processable in parallel. Spark's primary abstraction is the RDD, which may be produced from a Hadoop file system file or by modifying previous RDDs. One or more RDDs may be dependent on an RDD as shown in Fig. 3. The DAG Scheduler constructs stages as a form of execution unit from the graph of RDD dependencies.

DAG Scheduler generates a directed acyclic graph (DAG) of stages for each work, recording which RDDs and stage outputs were satisfied. Task sets are then given to the task scheduler in phases. Task scheduler distributes tasks to executors through the resource manager, which is YARN in this case. After finishing their jobs, executors will either return the results to the driver (see Fig. 3), if it is the end of the RDD of an operation like count ()), [28], or write the output to external storage. All of MapReduce's essential functions are included in the spark framework. It also comes with Dataproc improvements. Dataproc on Google cloud platform (GCP) is fast and easy to use for cloud computing. Open-source data analytics tools perform batch processing, querying, streaming, and machine learning, which introduces a high-performance infrastructure. With the following features [29]: Super-fast, automated management, low cost, autoscaling clusters, cloud integrated, flexible and familiar.

4. Implementation of parallelization

According to existing research, the four phases of the DBCAN on spark process are partitioning, local DBSCAN, data merging, and global clustering construction as shown in Fig. 4. In [19] Rather than partitioning with a virtual fishnet as in traditional DBCAN-Spark work, this project divides the partitions with different sample points into multiple batches to send to the driver end, then broadcasts to each executor to calculate the distance separately, then combines the results to achieve the double traversal indirectly. The spatial index R-tree is constructed before broadcasting to reduce the amount of calculation required for broadcasting.

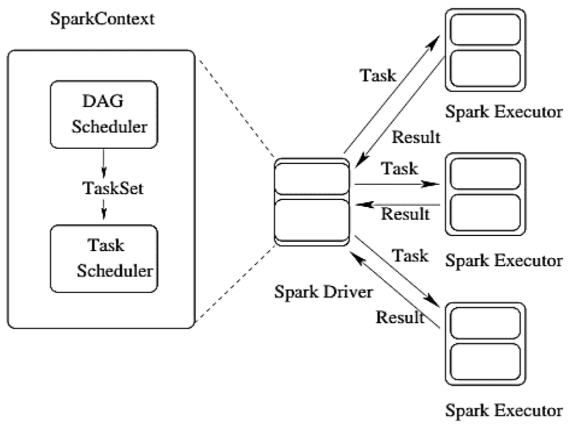


Figure. 3 Data flow in spark

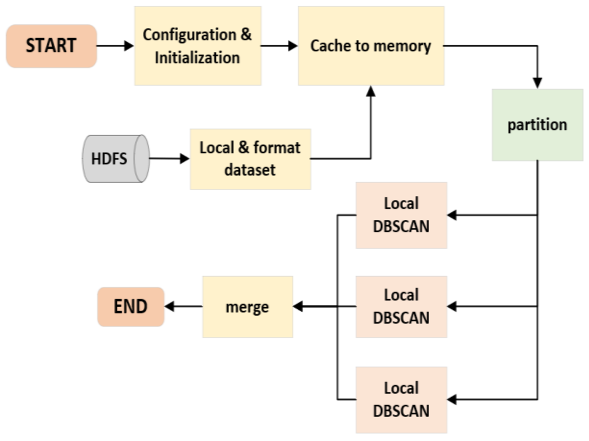


Figure. 4 Parallel DBSCAN

A. Naive parallelizied DBSCAN

The parallel matrix DBSCAN technique on spark [28] is implemented in three stages: partition, local matrix DBSCAN, and merging. The partitioning stage will be determined by specifying the partition number, and the position of the partition block in the space, and constructing the RDD, which will include the corresponding partition id and partition dataset assigned by determining whether or not the points are in the partition block.

Each RDD will use the local matrix DBSCAN method described in the prior section to perform the local matrix DBSCAN. The local dataset information as well as the local cluster label list will be produced after the approach is completed. In the merge step, the results of each RDD will be pooled using a global cluster list. If a label exists for a point in the global list, it will be used to update the local partition points with the same cluster label.

B. *Optimization* of partitioning

The partitioning stage is crucial to the overall performance of the clustering process. The top and bottom limits of all data points in each dimension are first calculated, generating a rectangle (bounding box), which is then proportionally divided into the whole height and width rectangles. Only when the data points are evenly distributed will this method succeed. In reality, it's practically impossible. If data points are distributed unevenly, the amount of data allocated to the staff will be highly unbalanced, and overall performance will be heavily dependent on the execution time of the most demanding job and the benefits of parallel processing. The computation will be as straightforward as possible.

A partitioning plan must be established to ensure that the personnel's tasks are spread as evenly as possible. To do so, we first store the data points in an R-tree data structure, which allows us to quickly get aggregate data in a given area.

1) Rtree based on Boundary: Because the points within a long distance of the boundary line are within the overlapping region and will be counted and calculated separately for each partition during their local clustering stage, reducing the number of points inside the overlapping region will reduce both the overall computation of each partition and the merging step.

Determine the smallest box that contains all of the data to begin splitting. Start splitting the rectangle box after that. Divide the box in half to create temporary B_1 and B_2, and consider the junction section, which is made up of two provided by both sides, to be the boundary section. Calculate the cost by multiplying the number of points in B_1 and B_2 by the number of points in the boundary part. The R tree's purpose is to lower the score so that B_1 and B_2 are both balanced and there are fewer edge points. If B_1 has fewer points than B_2, create a new splitter line at the location of the fourth segment in B_2. Each loop calculates the large rating while updating the smaller cost. Select the axis that corresponds to the decreased rating to split. Place the ultimate B_1 and B_2 in the queue. Remove the queue's leftmost B_m to do the calculation. We get a lot of B_m until we come to the final partitions at the end of the loop Fig. 5.

Because the factors inner the length of the boundary line are placed in the overlapping area, the boundary reduction approach reduces the wide variety of factors in the boundary. The distinct partitions will be counted and calculated for the duration of the applicable local clustering phases. The typical computation and merging techniques for every partition will be faster if the wide variety of points in the overlapping location is reduced.

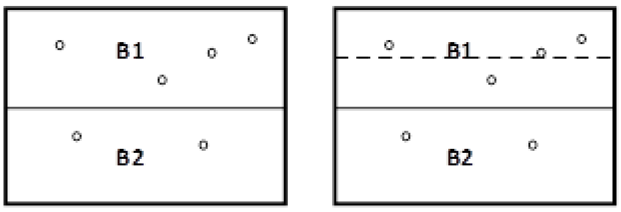
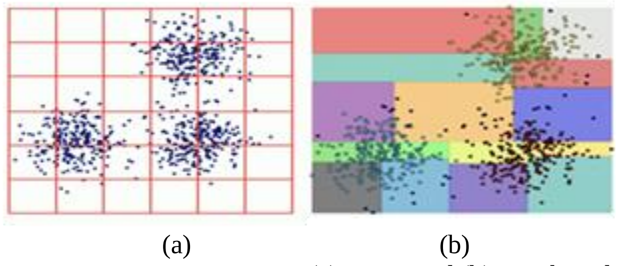


Figure. 5 Calculate the cost difference between B_1 and B_2


(a) (b)
Figure. 6 Partition strategy: (a) Naïve and (b) cost-based

2) Rtree based on cost (CBP): The cost-based technique seeks to distribute workload among partitions as equally as possible. here we use the notation "cost" to quantize the workload of performing local DBSCAN in a region. The division result of CBP is illustrated in Fig. 6 (b).

The collection of recommendations above determines a price that indicates the point's form. At this phase, the goal is to keep each worker's workload (expressed in points) as evenly distributed as feasible. Cut-up lines are typically tapped by price, usually in a relatively simple fashion, such as shrinking bounds, but the valuation features stay practically unchanged. The road can be drawn at the center of the rectangle (vertically and horizontally, respectively), resulting in B_1 and B_2. Then it's discovered that B_2 has one extra point. In other words, it was a better deal for the set of suggestions that arose in an uneven situation.

As a result, at the next stage of B_2, the avenue may be initialised within the center. Continue using this method until the cutline + double epsilon benefits from the elements that extend beyond the boundaries intended to retain the overlapping region's position after merging.

Statistical rtree based on cost (SCBP): The cost-based technique seeks to distribute workload among partitions as equally as possible. Before partitioning a statistical analysis [11] of data is applied to obtain the DBSCAN parameters which affect the clustering performance.

Specifically, the equations below are used to implement the function *EstimateCost* (*EC*) as Eq. (1):

if f represent the fanout of R-tree index, B represents the bounding box of partition and Np represents number of data points in partition B; DA represents non-overlapping cells with side length 2-ε inside the partition B, and N_{C_i} represents total number of data points in cell C_i.

Received: October 22, 2022. Revised: January 21, 2023.

Algorithm 1: Statistical *Rtree based on cost (SCbs)*

```
Input: the rectangle needed to be split
Result: $B_1$, $B_2$
1. split line candidates ← all horizontal and vertical
    lines aligned to cell boundaries in B;
2. minCostDiff ← 1;
3. $(B_1, B_2)$ ← (NULL, NULL) ;
  for each split line in split lines candidates do
     $(B'_1, B'_2)$ ← sub-rectangles split by split line
     costDiff ← | $EC(B'_1)$ - $EC(B'_2)$ |;
     if costDiff < minCostDiff then
       minCostDiff = costDiff;
       $(B_1, B_2) = (B'_1, B'_2)$;
       (eps, minPTs ) = statistical analysis $(B_1, B_2)$
     end
  end
return $(B_1, B_2$, eps, minPTs )
```

$$\mathrm{EC}(B) = \sum \mathrm{EC}(C_i) \tag{1}$$

$$\mathrm{EC}(C_i) = N_{C_i}.DA\left(N_{C_i}\right) \tag{2}$$

$$A\left(N_{C_i}\right) \approx 1 + h + N_{C_i}\sqrt{N_{C_i}}.\frac{2}{\sqrt{f}+1} + N_{C_i}.\frac{1}{f=1} \tag{3}$$

$$h = 1 + \left[log_f(Np/f)\right] + N_{C_i} \tag{4}$$

The Statistical *Rtree based on cost (SCbs) is shown in* Algorithm 1.

For the statistical *Rtree based on cost (SCbs)***.** If $x_1, x_2, \ldots, x_n$ are independently and identically distributed random variables on the d dimensional space R^d , the probability distribution density function $f(x)$ is estimated as Eq. (5):

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right) \tag{5}$$

h is the bandwidth, n represents the number of samples, $K(.)$ is the kernel function. In the subsequent approach, the Gaussian kernel function is applied to fit the probability density functions of radius and sample number of pre-clusters as Eq. (6):

$$K(x - x_i) = \exp\{-\|x - x_i\| 2\sigma^2\} \tag{6}$$

The values of **eps** and **minPTs** are determined as Eq. (7) and Eq. (8):

$$\boldsymbol{eps} = \arg\max_x \hat{f}_R(x) \tag{7}$$

$$\mathbf{minPTs} = \arg\max_x \hat{f}_P(x) \tag{8}$$

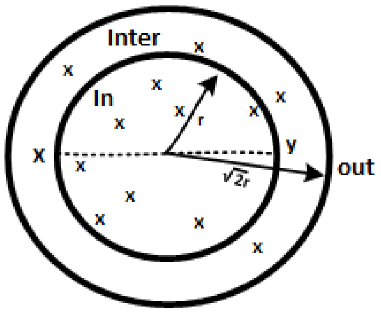


Figure. 7 Calculating $N_{in} + N_{inter}$

where, $\hat{f}_R(x)$ and $\hat{f}_P(x)$ is the probability density functions (pdf) of outer radius R and sample numbers $P = N_{in} + N_{inter}$ of pre-clusters.

5. Experiments

A. Experiments setup

We gathered a total of three datasets from the university of eastern Finland's Clustering Vital Benchmark [26] to show the practicality and scalability of our parallel approaches.

In particular, the dataset consists of sizeable datasets such as D31, which in reality display parallel computation capabilities. In addition, the dataset consists of a few unbalanced datasets, such as Jain. In addition, we looked at the effect on a high-dimensional dataset, such as Aggregation, to see how it affected top-notch distance measurements. We additionally utilised an excessive number of partitions (2, 4, 8, 12, 16, 24, and 32) to consider what influence they had on the going-for-walks performance on a big dataset to test the scalability of parallelization.

B. Performance evaluation

We introduce silhouette coefficient (SC), completeness (Comp.), and adjusted rand index (ARI) to measure the clustering performance of the proposed method compared to other recent parallel DBSCAN methods.

1) Silhouette coefficient (SC)

The silhouette coefficient is calculated using the mean intra-cluster distance a and the mean nearest-cluster distance b for each sample. The Silhouette Coefficient for a sample is $(b - a) / max(a, b)$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of.

2) Completeness (Comp.):

A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.

3) Adjusted rand index (ARI):

Rand index (RI): a function that computes a similarity measure between two clusterings. For this computation, RI Eq. (9) considers all pairs of samples

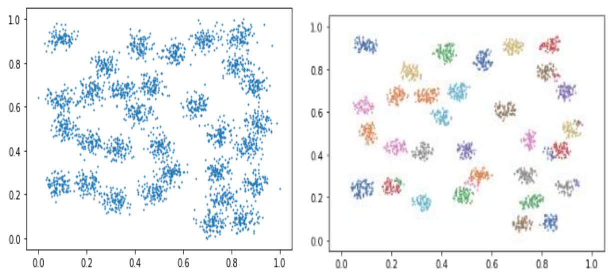


Figure. 8 Clustering result of D31 dataset

and counting pairs that are assigned in similar or different clusters in the predicted and true clustering. Afterward, the raw RI score is 'adjusted for a chance' into the ARI score by Eq. (10):

$$ARI = \frac{RI - Expected\_RI}{max(RI) - Expected\_RI} \quad (9)$$

$$RI = \frac{C_{in} + C_d}{\binom{n}{2}} \quad (10)$$

where C_i is the number of points that are in the same cluster for both clusterings, C_d is for those that are in a different cluster for both clusterings, and n is the total number of samples.

C. Dataset

1) D31 (N=3100, k=31, D=2):

The clustering result for the D31 dataset is shown in Fig. 8. It's worth observing that the same color of numerous of the clusters on the left of Fig. 8., although they are independent clusters. The goal is to eliminate any uncertainty about the clustering results from current and other datasets.

Naive parallel DBSCAN has a higher time cost than optimized parallel DBSCAN. Furthermore, the time cost of naive parallel DBSCAN is larger than that of our proposed optimized parallel DBSCAN (SCbs-DBSCAN) utilizing eight partitions, as shown in Table 1. The best performance is indicated in bold. We want to investigate how successful parallelization can boost algorithm efficiency in this dataset.

This could be owing to a significant variation between the two techniques. The naive parallel DBSCAN picks a neighborhood at random to compute the diversity of its internal N neighbors without storing it. While using the Optimized Parallel DBSCAN to classify kernel points reduces time by keeping all distances in a matrix, it does necessitate adequate storage average overall performance. We chose to document the time approach via this approach to identify which step wastes the most time because the overall performance of the RTree-based full technique falls short of that of the SCbs-DBSCAN.

The performance of SC, Comp., and ARI for SCbs compared to Rbs and Cbs for the D31 dataset is illustrated in Table 2. The best performance is indicated in bold.

Table 1. Time cost and performance of different datasets

Time Cost in milliseconds (ms)			
Alg.	**D31**	**JAIN**	**Aggregation**
Naive	10472	441	7531
Optimized	2712	202	4107
Rbs-DBSCAN	3948	230	6163
Cbs-DBSCAN	3631	187	3930
SCbs-DBSCAN	**1037**	**124**	**1877**

Table 2. Performance of D31 dataset

	Partition Number						
	2	**4**	**8**	**12**	**16**	**24**	**32**
Rbs-DBSCAN							
SC.	27	33	25	18	**29**	22	17
Comp.	**89**	**90**	**88**	85	**87**	**87**	82
ARI	43	46	49	42	54	47	52
Cbs-DBSCAN							
SC.	**65**	**45**	**26**	12	17	11	11
Comp.	77	82	82	81	77	73	73
ARI	25	15	28	28	25	19	35
SCbs-DBSCAN (eps = 2.2 minPTs = 8)							
SC.	34	31	25	**26**	21	**26**	**22**
Comp.	86	86	85	**86**	83	82	**87**
ARI	**58**	**58**	**58**	**60**	**59**	**62**	**65**

The difference between the various optimal approaches and the RTree method lies in the level of partitioning. This is because it's fairly easy to split all the statistics into each range of a fairly evenly distributed dataset using a good vintage split approach. Throughout the RTree-based solution, we need to find a high-quality partition plan that evenly distributes the dataset throughout each partition. As a result, after some search time, the RTree method provides the same partitioning approach as a similar historical partitioning strategy. As a result, processing RTree partitions takes a long time, but clustering and merging take even longer.

This outcome can be explained by two separate considerations:

- The number of executors does not affect the serial computation. the procedure will run serially on one of them. As a result of the increased number of executors, the time may no longer be developed
- Parallel complexity has a strong influence on overall parallel processing performance. When the number of executors and walls increases. As a result, the time cost for each partition, as well as the overall operating time, will likely be reduced.
- The parameter of each partition must be investigated before the partition to boost clustering performance.

2) Dataset Jain (N=373, k=2, D=2)

The data set Jain is a classic unbalanced data set,

Received: October 22, 2022. Revised: January 21, 2023.

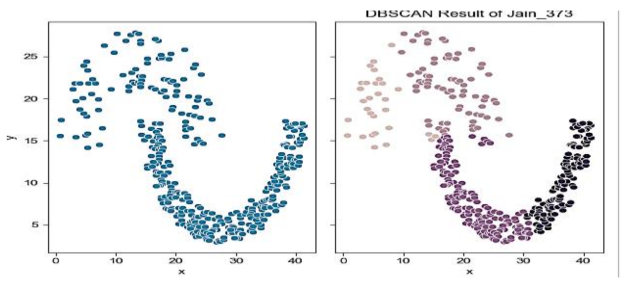


Figure. 9 Clustering result of Jain dataset

Table 3. Performance of Jain dataset

	Partition Number						
	2	**4**	**8**	**12**	**16**	**24**	**36**
Rbs-DBSCAN							
SC.	**41**	**34**	**30**	15	25	16	21
Comp.	25	24	22	22	21	21	**22**
ARI	18	16	11	13	09	11	13
Cbs-DBSCAN							
SC.	12	14	05	00	12	01	**26**
Comp.	21	21	18	12	13	19	14
ARI	12	13	12	05	05	11	08
SCbs-DBSCAN (eps = 5.2 #2.2 minPTs = 23)							
SC.	18	27	08	**20**	**34**	**20**	17
Comp.	**27**	**28**	**25**	**23**	**23**	**23**	20
ARI	**24**	**21**	**21**	**19**	**18**	**17**	**26**

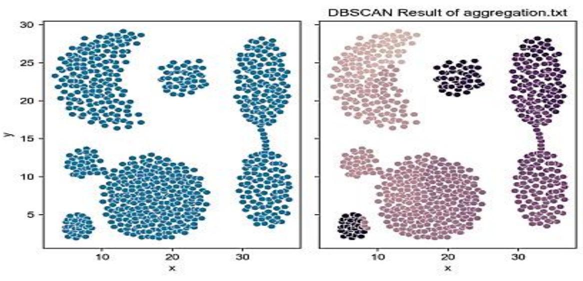


Figure. 10 Performance on aggregation dataset

that is, only a small part of the data set contains data points, while other spaces are empty as shown in Fig. 9. In the process of clustering, it is easy to cause workload tilt and lead to a sharp increase in time cost. So, the result of cost based (Cbs) method is to balance the data between workers.

The performance of SC, Comp., and ARI for SCbs compared to Rbs and Cbs is illustrated in Table 3. The best performance is indicated in bold.

3) Dataset Aggregation (N=788, k=7, D=2)

High-dimensional datasets include the Aggregation dataset shown in Fig. 10. Where different color indicates different cluster. The performance of SC, Comp., and ARI for SCbs compared to Rbs and Cbs is illustrated in Table 4. The best performance is indicated in bold.

In Figs. 11, 12, and 13, the time costs of merge, partition, and parallel DBSCAN phases are illustrated for the three different clustering datasets D31, Jain, and Aggregation. The partition number against the cost time in milliseconds (ms) is listed.

Table 4. Performance of aggregation dataset

	Partition Number						
	2	**4**	**8**	**12**	**16**	**24**	**36**
Rbs-DBSCAN							
SC.	30	27	**26**	14	**22**	17	**19**
Comp.	62	64	59	59	56	59	55
ARI	49	52	43	48	36	46	37
Cbs-DBSCAN							
SC.	24	20	18	05	15	01	04
Comp.	56	54	53	49	53	48	46
ARI	36	33	33	28	33	27	23
SCbs-DBSCAN (eps = 5.2 minPTs = 29)							
SC.	**47**	**35**	24	**27**	19	**22**	17
Comp.	**85**	**78**	**72**	**72**	**76**	**72**	**71**
ARI	**84**	**79**	**70**	**70**	**76**	**66**	**69**

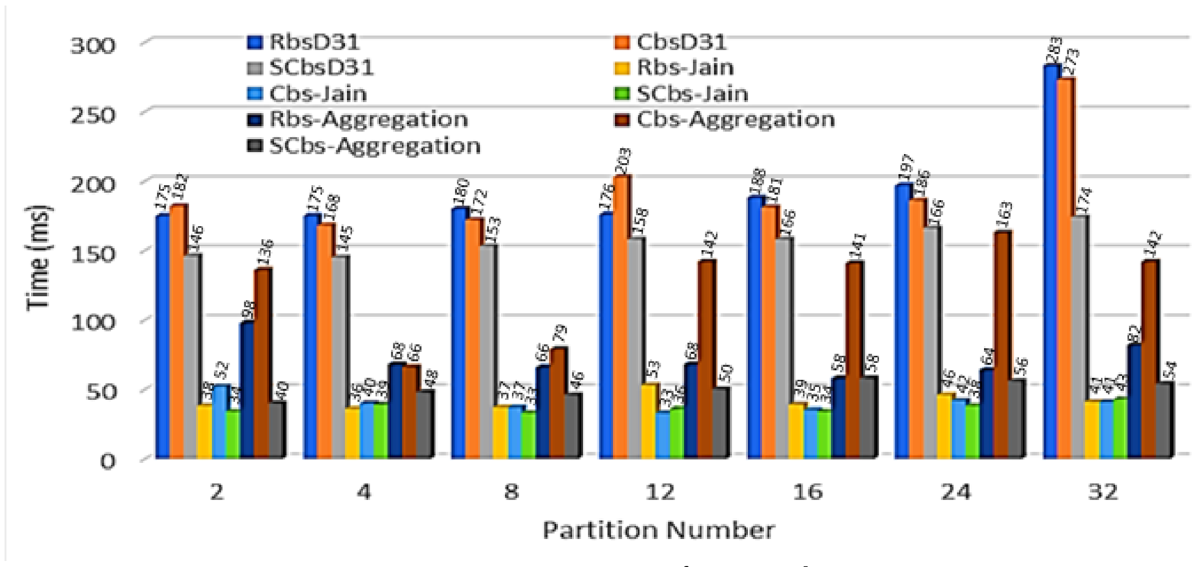


Figure. 11 Time Cost of partition phase

Received: October 22, 2022. Revised: January 21, 2023.

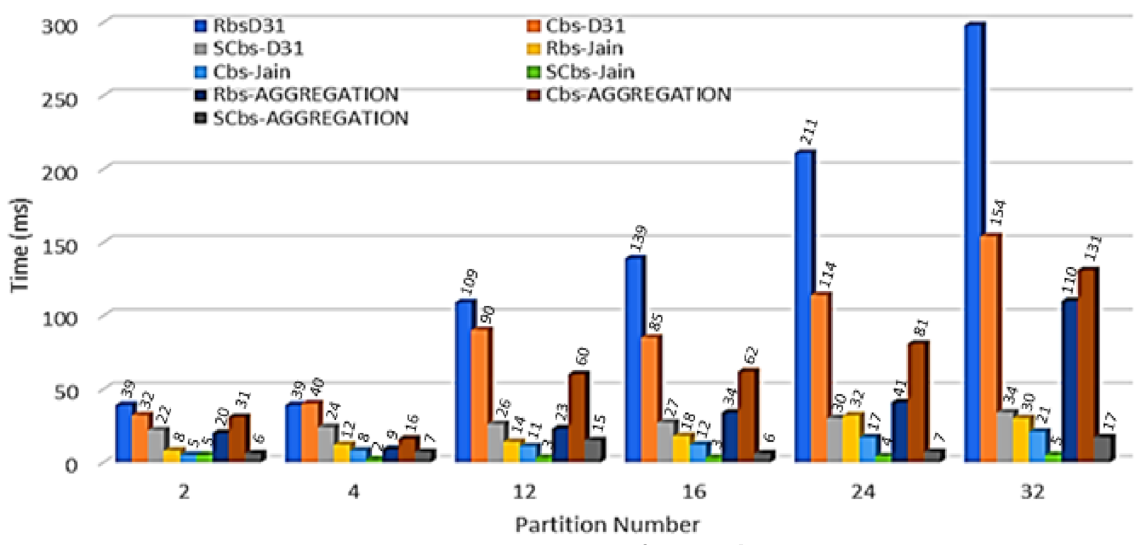


Figure. 12 Time Cost of merge phase

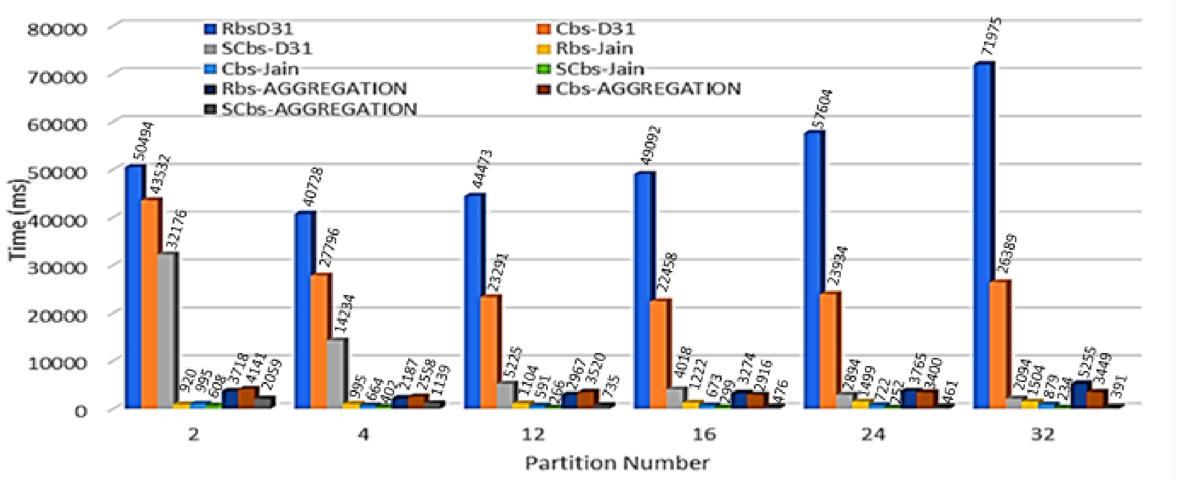


Figure. 13 Time Cost of parallel DBSCAN phase

As shown in Fig. 11, the time cost of the partitioning phase increases as the partitioning number increases. The time cost increases due to the time required to partition the data among the different workers. The time cost also depends on the data distribution in the dataset. But among the different partitioning methods, our method outperforms the other two methods for the three datasets. The cost time for partitioning in the Jain dataset is approximately equal, and this is due to the unbalanced dataset. That is, only a small part of the data set contains data points, while other spaces are empty. This requires additional time to compute the oration section. For the other two datasets, our method has a 1.6x and 2.8x time cost increase over the Rbs and Cbs methods, respectively.

As shown in Fig. 12, the time cost of the merging phase increases as the partitioning number increases. The time cost increases due to the time required to merge the results from different workers. Here, the time cost depends on the number of data workers used. But among the different partitioning methods, our method outperforms the other two methods for the three datasets. For the three datasets, our method has a 6x, 6.8x, and 8.7x time cost increase over the Rbs and Cbs methods, respectively.

As shown in Fig. 13, the time cost of the parallel DBSCAN Phase increases as the partitioning number increases. The time cost increases due to the time required to partition the data among the different workers. The time cost also depends on the data distribution in the dataset. But among the different partitioning methods, our method outperforms the other two methods for the three datasets. The statistics on the Jain dataset may be distributed inconsistently among walls, causing a few walls' burdens to be too high, requiring extra time to complete the clustering step as shown in Fig. 13. For the three datasets, our method has a 12x, 6.4x, and 9x time cost increase over the Rbs and Cbs methods, respectively.

Although using the RTree based method causes the partition procedure to take longer, the benefits outweigh the drawbacks. If you utilise the RTree partition method, all of the walls will share the same

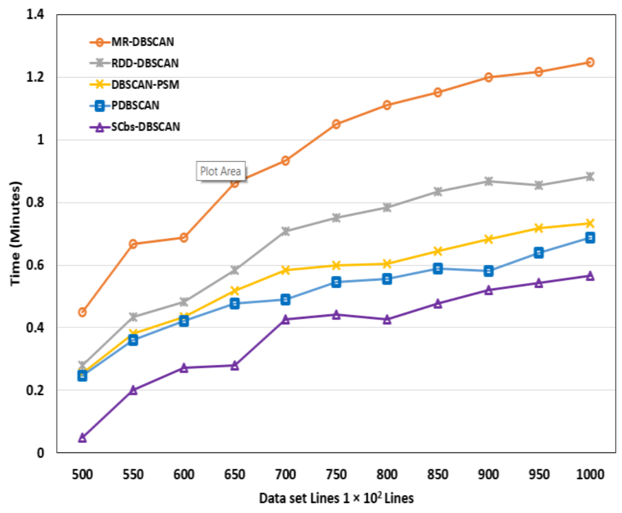


Figure. 14 Comparing the time cost of SCbs-DBSCAN and related work

burden. As a result, it saves a significant amount of time while clustering. For comparison of our proposed SCbs-DBSCAN with other related work, we chose MR-DBSCAN [22], RDD-DBSCAN [23], DBSCAN-PSM [24] and PDBSCAN [25]. along with serial-DBSCAN. To generate testing datasets, we us8e the scilearn [31] tool-sample generator (dataset loading utilities) to effectively test the clustering effect under different scales of data.

Fig. 14 illustrates the time cost of SCbs-DBSCAN against related work where the time decreases by a factor of 3, 2.1, 1.8 and 1.5 for MR-DBSCAN [22], RDD-DBSCAN [23], DBSCAN-PSM [24] and PDBSCAN [25]. Also, SCbs-DBSCAN shows scalability where the time increases a little with dataset lines of higher dimension. Experiments show that compared with the original DBSCAN algorithm and its related algorithms, the SCbs-DBSCAN algorithm is faster and more suitable for large-scale data sets.

6. Conclusion

In DBSCA Clustering, it is crucial to evaluate the best scenes in each portion to fully grasp the benefits of the partitioning provided by the R-Tree and make use of them in conjunction with an efficient segmentation strategy (R-Tree). This will speed up parallel processing. Performance of R-Tree is largely dependent on the initial distribution of datasets as a consequence of DBCAN clustering. This strategy helps speed up parallel computing. Disseminating certain information can lead to unnecessary calculations. Parallel DBSCAN algorithm is decoded and executed for spark cluster on Google cloud platform (GCP).

SCbs-DBSCAN obtains the optimum DBSCAN parameters using statistical analysis to enhance DBSCAN clustering performance and cost-based R-trees to optimize partitioning and merging for efficient cost time. Finally, the basic overall performance and partition changes, one for each type section, in addition to Spark's parallel programming capabilities. we have proved the scalability of the optimised parallel DBSCAN.

The desired results by comparing different parallel DBSCAN and serial DBSCAN using Basic benchmark clustering UCI standard datasets and large different scales of data.

First and foremost, we use Spark's parallelism to experiment with a better storage structure. Notice the decrease in time costs and the increase in space efficiency. Second, we can identify the advantages of partitioning brought by R-tree when we investigate the process of optimising partition performance, we discovered statistics fitted to various partitions to obtain DBSCAN optimum parameters which enhance clustering performance.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

The paper's background work, conceptualization, methodology, dataset collection, implementation, result analysis and comparison, preparation and editing of the draft, and visualization have been done by the first author. The supervision, review of work, and project administration have been done by the second author.

References

[1] R. V. Kranenburg, “The Internet of things: A critique of ambient technology and the all-seeing network of RFID”, *Institute of Network Cultures*, Vol. 4, No. 6, 2006.

[2] L. D. Xu, W. He, and S. Li, “Internet of Things in Industries: A Survey”, *IEEE Trans. Ind. Informatics*, Vol. 10, No. 4, pp. 2233–2243, Nov. 2014.

[3] H. Bangui, M. Ge, and B. Buhnova, “Exploring Big Data Clustering Algorithms for Internet of Things Applications”, In: *Proc. of the 3rd International Conference on Internet of Things, Big Data and Security*, pp. 269–276, 2018.

[4] A. S. Shirkhorshidi, S. Aghabozorgi, T. Y. Wah, and T. Herawan, “Big Data Clustering: A Review”, In: *Proc. of the 14th International Conference on Computational Science and Its Applications*, pp. 707–720, 2014.

[5] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering", *ACM Comput. Surv.*, Vol. 31, No. 3, pp. 264–323, Sep. 1999.

[6] J. Apostolakis, "An Introduction to Data Mining", *Structure and Bonding*, Vol. 134, pp. 1–35, 2011.

[7] J. MacQueen, "Some methods for classification and analysis of multivariate observations", In: *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, 1967.

[8] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH", *ACM SIGMOD Rec.*, Vol. 25, No. 2, pp. 103–114, Jun. 1996.

[9] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "WaveCluster: a wavelet-based clustering approach for spatial data in very large databases", *VLDB J. Int. J. Very Large Data Bases*, Vol. 8, Nos. 3-4, pp. 289-304, Feb. 2000.

[10] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", In: *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[11] N. Xie, Z. Miao, J. Wang, and Q. Zhang, "Adaptive DBSCAN Algorithm Based on Sample Density Gradient", *J. Phys. Conf. Ser.*, Vol. 1229, p. 012056, May 2019.

[12] M. M. A. Patwary, D. Palsetia, A. Agrawal, W. Liao, F. Manne, and A. Choudhary, "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure", In: *Proc. of 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, Nov. 2012.

[13] A. Zhou, S. Zhou, J. Cao, Y. Fan, and Y. Hu, "Approaches for scaling DBSCAN algorithm to large spatial databases", *J. Comput. Sci. Technol.*, Vol. 15, No. 6, pp. 509–526, Nov. 2000.

[14] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data", *Front. Comput. Sci.*, Vol. 8, No. 1, pp. 83–99, Feb. 2014.

[15] M. Noticewala and D. Vaghela, "MR-IDBSCAN: Efficient Parallel Incremental DBSCAN algorithm using MapReduce", *Int. J. Comput. Appl.*, Vol. 93, No. 4, pp. 13-18, May 2014.

[16] Z. Dafir, Y. Lamari, and S. C. Slaoui, "A survey on parallel clustering algorithms for Big Data", *Artif. Intell. Rev.*, Vol. 54, No. 4, pp. 2411–2443, Apr. 2021.

[17] W. Xiao and J. Hu, "A Survey of Parallel Clustering Algorithms Based on Spark", *Sci. Program.*, Vol. 2020, pp. 1-12, Sep. 2020.

[18] S. Kamaruddin, V. Ravi, and P. Mayank, "Parallel Evolving Clustering Method for Big Data Analytics Using Apache Spark: Applications to Banking and Physics", In: *Proc. of the 5th International Conference on International Conference on Big Data Analytics*, pp. 278–292, 2017.

[19] F. Huang et al., "Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the Spark platform", *Remote Sens.*, Vol. 9, No. 12, 2017.

[20] D. Han, A. Agrawal, W. Liao, and A. Choudhary, "Parallel DBSCAN Algorithm Using a Data Partitioning Strategy with Spark Implementation", In: *Proc. of 2018 IEEE International Conference on Big Data (Big Data)*, pp. 305–312, Dec. 2018.

[21] Y. Gong, R. O. Sinnott, and P. Rimba, "RT-DBSCAN: Real-Time Parallel Clustering of Spatio-Temporal Data Using Spark-Streaming", In: *Proc. of 2018 18th International Conference on Computational Science*, pp. 524–539, 2018.

[22] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, Feng, S. and J. Fan, "MR-DBSCAN: an efficient parallel density-based clustering algorithm using mapreduce", In: *Proc. of* 2011 *IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS), IEEE*, pp. 473–480, 2011.

[23] I. Cordova, and T. S. Moh, "DBSCAN on resilient distributed datasets", In: *Proc. of 2015 International Conference on High Performance Computing & Simulation (HPCS)*, pp.531–540, 2015.

[24] G. Chen, Y. Cheng, and W. Jing, "DBSCAN-PSM: an improvement method of DBSCAN algorithm on Spark", *Int. J. High Performance Computing and Networking*, Vol. 13, No. 4, pp. 417–426, 2019.

[25] X. Lu, Y. Wang, J. Yuan, X. Wang, K. Fu, and K. Yang, "A Parallel Adaptive DBSCAN Algorithm Based on k-Dimensional Tree Partition", In: *Proc. of 2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pp. 249–256, 2020.

[26] Y. H. R. et al., "Massively scalable density based clustering (DBSCAN) on the HPCC systems big data platform", *IAES Int. J. Artif. Intell.*, Vol. 10, No. 1, p. 207, 2021.

[27] J. Han, M. Kamber, and J. Pei, *Data mining concepts and techniques*, Morgan Kaufmann Publishers Inc. Third Edition, 2012.
[28] J. S. Damji, B. Wenig, T. Das, and D. Lee, *Learning Spark: Lightning-Fast Data Analytics*, O'Reilly Media, Inc. 2020.
[29] https://cloud.google.com/dataproc/docs
[30] Clustering basic benchmark. http://cs.joensuu.fi/sipu/datasets/
[31] F. Pedregosa, G. Varoquaux, Gramfort, "Scikit-learn: machine learning in Python", *Journal of Machine Learning Research*, Vol. 12, No. 10, pp. 2825–2830, 2011.