Egyptian Informatics Journal 21 (2020) 131–138

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Detection outliers on internet of things using big data technology

Haitham Ghallab *, Hanan Fahmy, Mona Nasr

Department of Information Systems, Helwan University, Helwan, Cairo, Egypt

A R T I C L E I N F O

Article history:
Received 3 July 2019
Revised 3 November 2019
Accepted 10 December 2019
Available online 26 December 2019

Keywords:
Internet of things
IoT
Big data
Data quality
Outliers Detection
DBSCAN
RDDs

A B S T R A C T

Internet of Things (IoT) is a fundamental concept of a new technology that will be promising and significant in various fields. IoT is a vision that allows things or objects equipped with sensors, actuators, and processors to talk and communicate with each other over the internet to achieve a meaningful goal. Unfortunately, one of the major challenges that affect IoT is data quality and uncertainty, as data volume increases noise, inconsistency and redundancy increases within data and causes paramount issues for IoT technologies. And since IoT is considered to be a massive quantity of heterogeneous networked embedded devices that generate big data, then it is very complex to compute and analyze such massive data. So this paper introduces a new model named NRDD-DBSCAN based on DBSCAN algorithm and using resilient distributed datasets (RDDs) to detect outliers that affect the data quality of IoT technologies. NRDD-DBSCAN has been applied on three different datasets of N-dimensions (2-D, 3-D, and 25-D) and the results were promising. Finally, comparisons have been made between NRDD-DBSCAN and previous models such as RDD-DBSCAN model and DBSCAN algorithm, and these comparisons proved that NRDD-DBSCAN solved the low dimensionality issue of RDD-DBSCAN model and also solved the fact that DBSCAN algorithm cannot handle IoT data. So the conclusion is that NRDD-DBSCAN proposed model can detect the outliers that exist in the datasets of N-dimensions by using resilient distributed datasets (RDDs), and NRDD-DBSCAN can enhance the quality of data exists in IoT applications and technologies.

1. Introduction

Nowadays, mobile devices, computers, microprocessors, and other electronic devices become essential in our daily life, and each device is manufactured with the ability to connect to the internet for specific reasons, which lead to creating a suitable environment to develop and implement Internet of Things (IoT) applications and technologies [1,2]. IoT is facing many challenges and open issues which include hardware, security, privacy, heterogeneity, virtualization, and data analysis challenges.

This paper focuses on IoT's data analysis issues; more specifically data quality issues. And since data quality issues include uncertainty problems, noise, outliers, inconsistency, and missing values, then the goal of this paper is the outliers' detection challenges of data quality. Many clustering algorithms are used to detect outliers, and in this research density-based spatial clustering of applications with noise (DBSCAN) algorithm is used [3]. DBSCAN has several benefits that make it special when dealing with clusters and outliers than other clustering algorithms, these benefits are: its ability to discover arbitrarily shaped clusters, sensitive to the existence of outliers and noise, and finally its input parameters are simple unlike other clustering algorithms, since that in DBSCAN it is not required for the users to enter the number of clusters as an input parameter, because clusters are determined after execution. DBSCAN is also well known among researchers and developers as it's heavily cited in many publications, and in 2014 it was awarded the 2014 SIGKDD test of time award [4].

DBSCAN was developed in the first place to deal with databases in a single machine, and that's caused several challenges according to the current situation of large datasets that need to be distributed across multiple nodes and to be processed in parallel [5,6]. As shown in Table 1. MapReduce is a widely used paradigm for scaling algorithms, Dai and Lin [7], and Luo and Mao [8] have proposed DBSCAN-MR and MR-DBSCAN algorithms respectively. Also an

* Corresponding author.
E-mail addresses: haitham.ghallab@fci.helwan.edu.eg (H. Ghallab), hanan.fahmy@fci.helwan.edu.eg (H. Fahmy), drmona_nasr@fci.helwan.edu.eg (M. Nasr).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

Table 1
Scaled DBSCAN algorithms.

Name	Architecture	Dimensions	Distance function supported
MR-DBSCAN (2011) [8]	MapReduce	high	Euclidean
DBSCAN-MR (2012) [7]	MapReduce	high	Euclidean
PARDICLE (2014) [15]	MPI	high	Euclidean
DBCURE-MR (2014) [16]	MapReduce	high	Euclidean
RDD-DBSCAN (2015) [11]	Apache Spark	Low	Euclidean
NG-DBSCAN (2016) [9]	MapReduce	High	Arbitrary Symmetric

algorithm called NG-DBSCAN [9] has been presented and these algorithms are a variations of DBSCAN that allow the algorithm to run on the top of the Hadoop framework which is the implementation of MapReduce paradigm, but Zaharia, et al. has noticed that one of the limitations and drawbacks of MapReduce is that all the actions and transformations that occur during the parallel processing of the algorithms have happened through the file system, which causes high latency and high computational cost. Although these costs are acceptable for algorithms that not make many iterations over the same data, but this is not the best solution for iterative algorithms including DBSCAN, and to solve this limitation Zaharia, et al. has proposed a solution which is the abstraction for in-memory computing: resilient distributed datasets (RDDs) [10].

Resilient distributed datasets (RDDs) processes a large amount of data by making use of the RAM, by allowing its actions and transformations to use the cache memory to reduce the computational cost and to make the computations more faster. Cordova and Moh [11] have presented a new algorithm called RDD-DBSCAN to overcome the drawbacks of MapReduce with DBSCAN and to be processed in parallel and in a distributed way using RDDs. Cordova and Moh used apache spark which considered to be an implementation of RDDs to implement RDD-DBSCAN.

On the other hand, there are a set of researches that have used MapReduce paradigm with other techniques and approaches rather than the clustering techniques, and these researches focus on building an anomaly detection engine for IoT applications such as the anomaly detection engine (ADE) which is used to detect anomalies in IoT smart applications by using time-series models [12]. And also, Nesa, Ghosh, and Banerjee have used another technique that depends on statistical learning models to detect outliers of IoT [13]. Also, Hasan, Islam, Zarif, and Hashem have compared the performance between different machine learning approaches such as Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and Artificial Neural Network (ANN), and this comparison focuses on presenting the most suitable machine learning approach that can be used to detect the anomalies in IoT sensors in IoT sites [14].

One of the limitations of RDD-DBSCAN is that it works only with datasets of low dimensions, as one of the phases of RDD-DBSCAN is called the partitioning phase which required that the dataset must be presented in a two-dimensional representation to be partitioned otherwise the partitioning collapses [11]. So in this paper, a new model called NRDD-DBSCAN is proposed to detect outliers exists in IoT applications by scaling DBSCAN horizontally and allow DBSCAN to be processed across multiple nodes and in a distributed fashion using resilient distributed datasets (RDDs), also NRDD-DBSCAN could be applied to datasets of high dimensions and it has been implemented to solve the low dimensionality limitation of RDD-DBSCAN.

The research is divided into: the first part is a background of the study; the second part is the proposed model, the third part regarding the evaluation and results, while the last part is the limitations and future work.

2. Background

There are some key concepts that need to be illustrated regarding the architecture of IoT and how the objects talk and communicate with each other through the internet. In addition to describing how DBSCAN is scaled and how it could detect outliers from such a massive heterogeneous generated data.

2.1. Internet of things (IoT)

IoT was first introduced in 1999 by Kevin Ashton [17] and it was defined as a group of smart objects connected together via radio frequency identification (RFID) technology but the exact definition of IoT is still in developing and forming process. The design of IoT architecture consists of different layers, and these layers are:

1) *Perception layer*: is the physical layer or hardware layer that includes smart objects equipped with sensors, actuators, and microprocessors that generate and collect information and sensing the statuses of things [18,19].
2) *Network layer*: allow things to communicate, talk, and share data among each other through wireless or wired network infrastructure and be aware of their surroundings, and then data is aggregated, collected and send to the service layer [18,19].
3) *Service layer*: is the middleware layer used to monitor, create, and manage services that are required by applications and users [18,19].
4) *Interface layer*: it allows exchange, communication, and events processing between heterogeneous things regardless of the hardware platform and different devices standard, it allows the interaction between users and applications [18,19].

2.2. Resilient distributed datasets (RDDs)

IoT requires millions of objects equipped with sensors to generate and collect data, but this data is so big and complex to be processed by traditional applications and causes paramount issues and challenges in data analysis, data storage, capturing data, querying, information security, privacy, searching, and visualization, etc., as IoT is dealing with big data [20]. There are many tools, approaches, and methodologies [21] introduced to process, store, and analyze big data. Such these tools are Hadoop MapReduce and resilient distributed datasets (RDDs) [22,23]. MapReduce perform reliable computations on a large amount of data through three main steps: map, shuffle and reduce, but one of the major limitations of MapReduce is that it is not efficient when implementing iterative algorithms that require several passes over the same data because MapReduce writes most of the data to disk after each map, shuffle and reduce operations and that causes the computations to take much time and the performance of an iterative algorithm becomes worse as the number of passes over the same data increases [24,25].

Then resilient distributed datasets (RDDs) [10] is proposed to solve this issue, RDDs maybe think as an alternative to MapReduce and it has four main features which are: distributed collection of data, fault-tolerant, parallel operations and its ability to use many data sources not just the hadoop distributed file system (HDFS), also RDDs does not require to store the files in HDFS so it perform operations faster than MapReduce as most of the data is cached in the memory after each transformation and operation, and it can spill over the disk if the memory is filled, it is also better in memory management than other approaches as it applies fault-tolerance without data replication. So RDDs become a more suitable environment to host any iterative algorithms than other environments. So in this research, the RDDs approach will be used to implement the NRDD-DBSCAN model to detect outliers.

2.3. Partitions, chunks and clusters

Since the amount of data generated during IoT operations is enormous and to process such data by using DBSCAN algorithm, then DBSCAN is scaled horizontally and data is distributed and split into several partitions as in Eq. (1).

$$D = p1 \cup p2 \cup p3 \cdots \cup pn \tag{1}$$

where D refers to the whole data and $p_{1,\,2,\,3\,\ldots\,n}$ refers to the partition. Furthermore, these partitions are processed across multiple nodes, and each single node apply DBSCAN on the partition assigned to it independently, where DBSCAN loops through each point in the partition and starts to create clusters, where each cluster involve a set of points that share common characteristics, but if there are some points does not belong to any of the clusters exist in the partition then these points are outliers as in Eq. (2).

$$P = (c1 \cup c2 \cup c3 \cdots \cup cn) \cup (o1 \cup o2 \cup o3 \cdots \cup on) \tag{2}$$

where P refers to a single partition and $c_{1,\,2,\,3\,\ldots\,n}$ refers to the cluster, while $o_{1,\,2,\,3\,\ldots\,n}$ refers to the outlier if it exists. And each cluster includes a set of data points that share the same characteristics as in Eq. (3).

$$C = x1 \cup x2 \cup x3 \cdots \cup xn \tag{3}$$

where C refers to a single cluster and x1, x2, x3, and xn refer to the data points that share the same traits. Finally, notice that the term chunk is a synonym of the term partition.

2.4. DBSCAN algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm; DBSCAN depends on two parameters: (MinPts) the minimum number of points in the neighborhood which required to create a cluster, (ϵ) the radius within a point. There are some key concepts regarding DBSCAN algorithm which are:

1) *Core point*: point p $\in$ X is a core point when the number of neighbors within its radius (ϵ) >= MinPts.
2) *Border point*: point p $\in$ X is a border point when p $\in$ c1 cluster and p is not a core point.
3) *Outlier:* point p$\in$ X is an outlier when p is neither a core point nor border point and not belongs to any cluster.
4) *Directly reachable*: point q $\in$ X is directly reachable to p when p is a core point and q is inside point p's neighborhood.
5) *Density reachable*: point y $\in$ X is density reachable to p, when p and q are core points and q is directly reachable to p, and y is directly reachable to q and y is not directly reachable to p.

As one of the benefits of DBSCAN is that it's not required to pass the number of clusters as a parameter. So DBSCAN starts with a set of unlabeled points X={x1, x2, x3 … xn} and the output will be a set of labeled points X={x1, x2, x3 … xn} where xn has a flag of core, border or outlier, and in case of core and border it will be assigned to cluster identifier, and in case of an outlier, it will be assigned to -1. Many equations can be used to calculate the distance between points to determine the neighbors of a certain point, the equation that is used in this research is the Euclidean Distance Matrix (EDM) [26].

3. The proposed model NRDD-DBSCAN

3.1. Overview

DBSCAN algorithm cannot be applied to big data and it needs to be scaled and configured so it can be applied across multiple nodes in parallel and in a distributed way, so NRDD-DBSCAN model is proposed for detecting outliers and implemented using RDDs, the model is applicable for n-dimensions and implemented using apache spark.

NRDD-DBSCAN consists of three phases: data reduction and allocation, local clustering and data aggregation and renaming as shown in Fig. 1. And the phases of NRDD-DBSCAN proposed model is described in Algorithm 1.

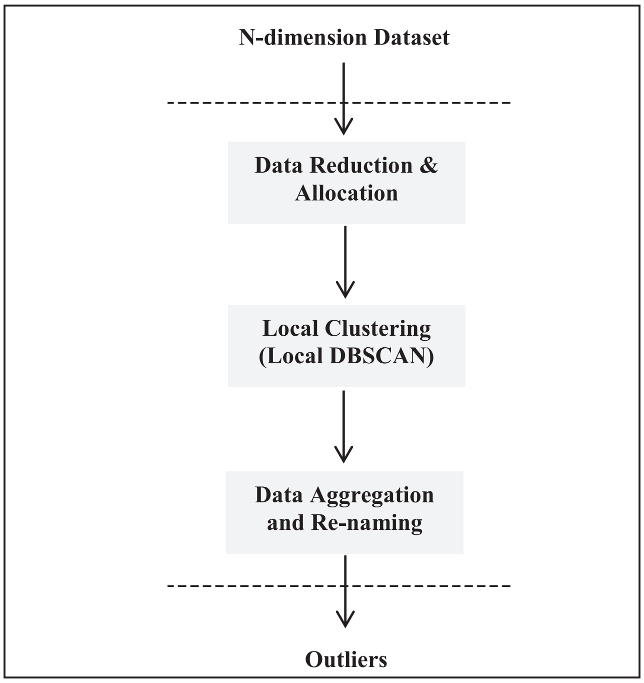

N-dimension Dataset
Data Reduction & Allocation
Local Clustering (Local DBSCAN)
Data Aggregation and Re-naming
Outliers

Fig. 1. NRDD-DBSCAN Phases.

Algorithm 1 The proposed NRDD-DBSCAN algorithm

Input: A set of points X = {x1, x2, x3 …. xn} where xn is an unlabeled point, ϵ is the radius within point, and MinPts is the minimum number of points required to create cluster, rowsNumber is the number of rows required to fit the node memory, nodesNumber is the number of working nodes.

Output: A set of points O = {o1, o2, o3 …. on} where on is an outlier.

```
1. label edPartitions $\leftarrow \emptyset$
2. partitions $\leftarrow$ evenlyChunks(X, rowsNumber, nodesNumber)
3. foreach part $\in$ partitions do
4.    {pn} $\leftarrow$ DBSCAN(part, $\epsilon$, MinPts)
5.    labeledPartitions $\leftarrow$ labeledPartitions $\cup$ {pn}
6. endfor
7. aggregatedPartitions $\leftarrow$
   aggregations(labeledPartitions, $\epsilon$, MinPts)
8. renamedPoints $\leftarrow$ renaming(aggregatedPartitions)
9. outliers $\leftarrow$ all $-$ 1valueinrenamedPoints
```

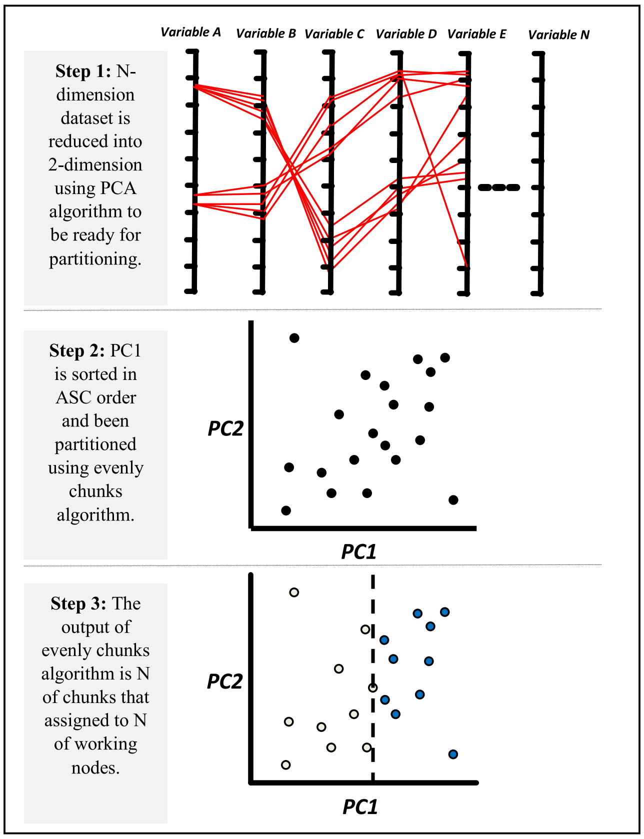

Variable A
Variable B
Variable C
Variable D
Variable E
Variable N
Step 1: N-dimension dataset is reduced into 2-dimension using PCA algorithm to be ready for partitioning.
Step 2: PC1 is sorted in ASC order and been partitioned using evenly chunks algorithm.
PC2
PC1
Step 3: The output of evenly chunks algorithm is N of chunks that assigned to N of working nodes.
PC2
PC1

Fig. 2. Phase one: Data Reduction & Allocation.

1) Phase one: Data reduction and allocation

This phase is divided into two sub-phases: data reduction and data allocation. The data allocation process required that the data points are represented into a scheme of two dimensions to be clustered as shown in Fig. 2. The reason for this representation is that NRDD-DBSCAN uses a two-dimension representation for an efficient partitioning scheme. Then the first step of this phase is to reduce the dimensionality of the dataset into 2D dataset, PCA algorithm has been used for this mission as principal component analysis (PCA) [27,28] is a powerful tool to reduce the dimensionality of a linear highly correlated n-dimension dataset but classical PCA algorithm cannot be applied to big data because of memory and storage barriers [29], then the implemented PCA algorithm of apache spark APIs (pyspark) [22] for big data is used in this phase.

Data allocation starts to load the 2d-dataset and divided it into a number of evenly chunks but take in consideration to avoid creating partitions that exceed the available memory space, forcing RDDs to be loaded from disk and negating the speed benefits that RDDs provide which is the ability to make multiple passes over data that is already loaded into memory.

Algorithm 2 starts with the input which is the n-dimension unlabeled dataset X= {x1, x2, x3, x4 .... xn} and since the allocation process required a two-dimension representation to partition the X dataset into evenly chunks or partitions. Then PCA [27,28] algorithm takes two parameters: n-dimension dataset and the number of dimensions required for partitioning which is two-dimension reducedDimensionalityData←PCA(X, 2).

The research used the implemented PCA APIs provided by pyspark [22] to work with big data analysis and to reduce the dimensionality of X, after the reduction process PC1 has been sorted in ASC order to begin the allocation and partitioning phase, the sortedData is now ready to be partitioned into a number of

Algorithm 2 Evenly chunks

Input: A set of unlabeled points X= {x1, x2, x3 ... xn}
Where xn is an unlabeled point of n-dimensions, rows Number is the number of rows required to fit the node memory, nodesNumber is the number of the working nodes.

Output: A set of chunks C= {c1, c2, c3 .... cn} where cn is a chunk or partition where each partition contain a number of points that's can fit the size of the node space memory

```
1. chunks ← ∅
2. reducedDimensionalityData ← PCA(X, 2)
3. sortedData ← sortingReducedDataByPC1
   (reducedDimensionalityData)
4. numberOfPointsPerNode ← ceiling
   (sortedData.length ÷ nodesNumber)
5. if numberOfPointsPerNode > rowsNumber then
6.     nodesNumber = sortedData.length ÷ rowsNumber
       and then recalculate, numberOfPointsPerNode
7. end if
8. for n ∈ nodesNumber do
9.     {c} ← splitPointsFromSortedData
       (numberOfPointsPerNode)
10.    chunks ← chunks ∪ {c}
11. return chunks
```

evenly chunks, since rowsNumber is a parameter responsible to determine the size of chunk and to ensure that the size of the chunk can be fit the node's space memory and not exceed it, and nodesNumber is a parameter responsible for determining the number of partitions and help to calculate the number of points per node, then a loop will be started to split the sortedData into number of parts (nodesNumber) or chunks and each chunk contain numberOfPointsPerNode points then assign each part to an individual node to start local clustering.

2) Phase Two: Local Clustering

After Data reduction and allocation phase, A set of partitions C = {c1, c2, c3 .... cn} have been generated where cn contains a set of unlabeled points, these partitions are ready to be assigned to a number of nodes where each node will apply DBSCAN algorithm to the chunk assigned to it, since each node is working independently and each partition does not have knowledge about other partitions, Then the result of DBSCAN in each partition will be a set of points P = {p1, p2, p3 ... pn} where pn is a labeled point with a flag of core, border or outlier, and in case of core or border it will be assigned to cluster identifiers, But these cluster identifiers are unique only to their partitions and that leads to different clusters in different partitions will have the same cluster identifiers. And that does not mean that both clusters represent the same thing.

To solve this issue, each point must be labeled by both cluster identifier and partition identifier as (PnCn) where Pn is the partition number and Cn is the cluster number, in case of outliers the flag value is (-1), then the result of Local Clustering is a set of partitions where each partition have a set of points Pn, and Pn has a flag of core, border or outlier, and if the result is core or border the value will be (PnCn) and in case of outliers the value will be (-1).

3) Phase three: Data Aggregation and Renaming

There are two main tasks regarding this phase: data aggregation and data renaming tasks, data aggregation phase's inputs are a set of partitions P= {p1, p2, p3 ... pn}, where pn includes a set of

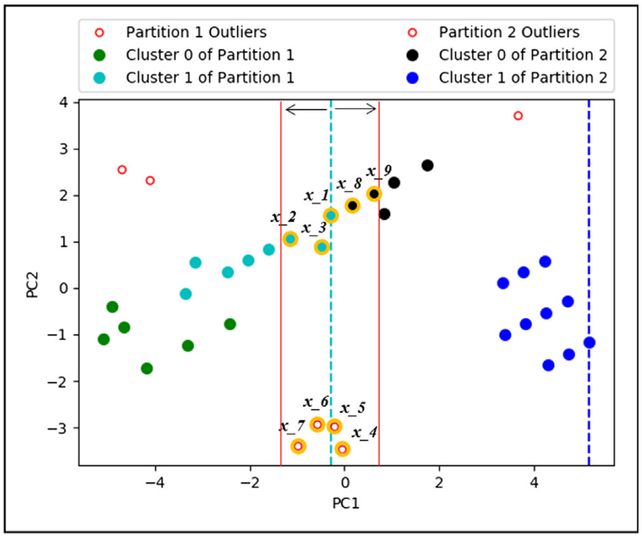


Fig. 3. Data Aggregation.

labeled points xn, where xn is either a core, border or outlier point, in case of outlier the value will be (-1) and in case of core or border points the result will be (PnCn) where P1C0 means a point belongs to partition (1) and inside this partition it belongs to cluster (0), Now it's time to aggregate all these separated partitions into just one, but the aggregation process faces a major issue which is the edge points clustering issue, the edge points is a set of points located at the borders of a partition, the edge points may be an outlier or belong to a cluster but it faces the possibility that it may belong to another cluster beyond the border.

According to Fig. 3, ifϵ is 1 and MinPts is 2 then x_1 which belongs to P1C1 also belongs to P2C0, also x_6 and x_7 which appears to be an outliers but actually they belong to a cluster that includes x_5 and x_4 which seems to be also an outliers in P2, aggregation process starts to go through each partition respectively except the last partition, it starts with the first partition P1 and get the largest PC1 valued point which will be x_1 which consider being the last point of P1 and also x_1 is the location where startLine is going to be set up.

The first step is to get all the edge points by using the startLine, then subtracting and adding one epsilon from startLine to goBackOneEpsilon and goForwardOneEpsilon then all the points (borderPoints) between them are the only points capable to share more than one cluster between the two partitions P1 and P2, Then foreach point x_n inside borderPoints starts to check its neighbors (by using Euclidian Distance Matrix) and if the number of neighbors is more than or equal to MinPts then for each neighborPoint belong to neighbors if the neighborPoint's partition is not the same as the x_n point's partition then there are four cases may occurred: the first one if a point is an outlier (ex. x_6, x_7) and the directly reachable neighborPoint is also an outlier (ex. x_4, x_5) then x_6, x_7, and x_4, x_5 will be PiCj where i is a new unique partition value and j is a new unique cluster value, the second case if a point is an outlier and its directly reachable neighbor is in PnCn then the point will be PnCn as its neighbor, the third case if the point is PnCn and its directly reachable neighbor is an outlier then the neighbor will be PnCn as its core point, the last condition if a point is PnCn and its directly reachable neighbor is Pn + 1Cn then all the points that belong to Pn + 1Cn will become PnCn because after aggregation all the points that belong to Pn + 1Cn will become density reachable to PnCn, for example, if x_1 belongs to P1C1 and x_8 belongs to P2C0 then x_8 will be reassigned to P1C1 along with all points that belong to the same cluster as x_8, data aggregation described in Algorithm 3.

Algorithm 3 Data Aggregation

Input: A set of partitions P= {p1, p2, p3 … pn} where pn is a labeled point, ϵ is the radius within a point, and MinPts is the minimum number of points required to create a cluster.

Output: A set of points Y= {y1, y2, y3 … yn} where yn is a labeled points with a flag of core, border or outlier,

in case of outlier the flag will be (-1) and in case of border and core point the flag will be (PnCn) where Pn is partition number and Cn is the cluster number.

```
1. foreach part ∈ P except the last partition do
2. startLine ← lastElement(part)
3. goBackOneEpsilon ← startLine − ε
4. goForwardOneEpsilon ← startLine + ϵ
5. borderPoints ← GetPointsbetween(goBackOneEpsilon,
   goForwardOneEpsilon)
6. foreach point ∈ borderPoints do
7.     neighbours ← GetNeighbors(point, ϵ)
8.     if length(neighbors) ≥ MinPts then
9.       for neighborPoint ∈ neighbours do
10.          if partitionOf(neighborPoint) ! ==
        partitionOf(point) or (clusterOf(neighborPoint)
        == −1 and clusterOf(point) == −1) then
11.          if clusterOf(point) == −1 and
             clusterOf(neighborPoint) ! = −1 then
12.            clusterOf(point) ← clusterOf(neighborPoint)
13.            partitionOf(point) ← partitionOf(neighborPoint)
14.          else if clusterOf(neighborPoint) == −1
             andclusterOf(point) ! = −1 then
15.          clusterOf(neighborPoint) ← clusterOf(point)
16.          partitionOf(neighborPoint) ← partitionOf(point)
17.          else, if, clusterOf(neighborPoint) == −1
             and clusterOf(point) == −1 then
18.          clusterOf(point) ← newuniqueclustervalue
19.          partitionOf(point) ← newuniquepartitionvalue
20.          clusterOf(neighborPoint) ← clusterOf(point)
21.          partitionOf(neighborPoint) ← partitionOf(point)
22.        else
23.          pointsOfNeighborCluster ← GetPointsBelongTo
             (clusterOf(neighborPoint), partitionOf(neighborPoint))
24.          foreach p pointsOfNeighborCluster do
25.            clusterOf(p) ← clusterOf(point)
26.            partitionOf(p) ← partitionOf(point)
27.          endFor
28.          endif
29.        endif
30.      endFor
31. endif
32. endFor
33. endFor
34. aggregatePoints ← p1 ∪ p2 ∪ p3... ∪ pn
35. return aggregatePoints
```

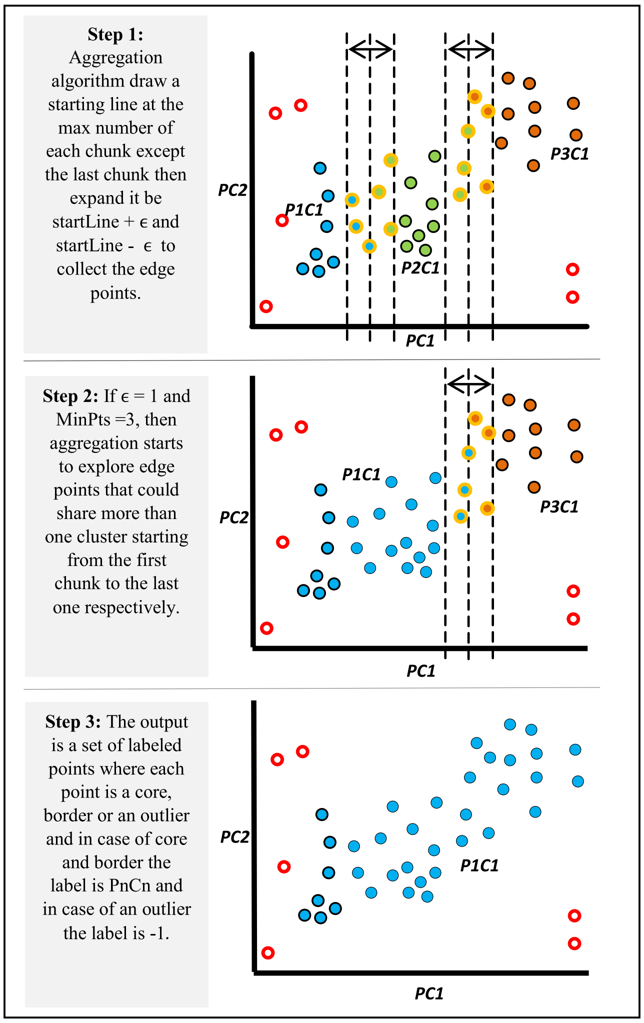
Then the result of Data Aggregation process will be that all the partitions have been combined into one and each point is either an outlier which its value is (-1), or a core or border point which its value is (PnCn) and after data aggregation there are no points that share more than one cluster as shown in Fig. 4. The data renaming process starts to rename the point's label from PnCn to Cn and the outliers will still −1, as shown in Algorithm 4.

The result of the data renaming process is that the points are labeled using ***Cn*** instead of ***PnCn***.

4. Evaluation

NRDD-DBSCAN is implemented using apache spark which is the implementation of resilient distributed datasets (RDDs) and has been written using Python and has been bound to apache spark using spark python APIs (Pyspark) [22].

Three experiments have been made to evaluate NRDD-DBSCAN, the first experiment is made by using a non-synthetic dataset that



Step 1: Aggregation algorithm draw a starting line at the max number of each chunk except the last chunk then expand it be startLine + ϵ and startLine - ϵ to collect the edge points.

Step 2: If ϵ = 1 and MinPts =3, then aggregation starts to explore edge points that could share more than one cluster starting from the first chunk to the last one respectively.

Step 3: The output is a set of labeled points where each point is a core, border or an outlier and in case of core and border the label is PnCn and in case of an outlier the label is -1.

Fig. 4. Data Aggregation sub-phase.

Algorithm 4 Data Re-naming

Input: A set of points X = {x1, x2, x3 …. xn} where xn is a labeled point of core, border with a value of PnCn (Pn is partition identifier and Cn is cluster identifier), or outlier with value −1.

Output: A set of points X = {x1, x2, x3 …. xn} where xn is a labeled point of core, border with value of Cn (cluster identifier), or outlier with value −1.

```
1. numberOfPartitons ← maxOf(partitionsOf(X))
2. numberOfClusters ← maxOf(clustersOf(X))
3. newClusterIdentifier ← uniquevalue
4. **foreach** partitionIdentifier
   range(1, numberOfPartitons) **do**
5.   **foreach** clusterIdentifier range(0, numberOfClusters) **do**
6.      filteredPoints ← filteredPointsBy(X,
        clusterIdentifier, partitionIdentifier)
7.      **if** lengthOf(filteredPoints) > 0 **then**
8.         clusterOf(filteredPoints) ← newClusterIdentifier
9.         newClusterIdentifier ← newuniquevalue
10.     **endif**
11.  **endFor**
12. **endFor**
13. **return** X
```

was constructed by adding elevation information to a 2D road network in North Jutland, Denmark [30] with 434,874 instances and four attributes, while the second experiment has been made by using a synthetic dataset of one million entries, two features, three centers and cluster standard deviation of 0.4, finally the third experiment has been made by using synthetic dataset of one million entries, 25 features, three centers and cluster standard deviation of 0.4.

The synthetic dataset allowed observing and evaluating the behavior and performance of NRDD-DBSCAN under different conditions, and the synthetic dataset is generated by using make_-blobs method of scikit-learn's samples generator utility [31]. In order to verify the correctness of NRDD-DBSCAN in detecting outliers, evaluation metrics will be used to evaluate the performance of the clustering algorithm, regarding the Adjusted Rand Index, Adjusted Mutual Information, Homogeneity, Completeness, and V-measure which requires the ground truth labels labels_true to be known, and based on [32].

The first experiment used a sample of 100 instances from the dataset described above and the given parameters are $MinPts = 3, \epsilon = 1$,labels_true and labels_pred, where labels_true is the ground truth class assignments of the samples and it's calculated by using scikit-learn's implementation of DBSCAN [33], and labels_pred is the results of NRDD-DBSCAN. The results of DBSCAN are applied after excluding OSM_ID column then (LONGITUDE, LATITUDE and ATTITUDE) columns are standardized and scaled to overcome the curse of dimensionality issues but in fact, it is not required to scale the data in the existing dataset as data units are the same and the number of attributes is only four attributes. The results show that the points are divided into four clusters (0, 1, 2 and 3) and the outliers are identified by −1, and then the labels_true values are the cluster column. And to calculate the labels_pred that are determined by using NRDD-DBSCAN, then an environment has been set up of a cluster of two virtual machines of Linux based system running locally; each machine has 2 CPUs and base memory (RAM) of 4096 MB with 20 GB hard disk. The results show that the points are divided into four clusters (100, 200, 300 and 400) and the outliers are identified by −1, and then the labels_pred values are the cluster column. Now all the parameters required calculating Adjusted Rand Index, Adjusted Mutual Information, Homogeneity, Completeness and V-measure are exists, then Homogeneity of NRDD-DBSCAN is 1.0, Completeness is equal 1.0, V-measure is equal 1.0, Adjusted Rand Index is equal 1.0 and finally Adjusted Mutual Information is equal 1.0, the mathematical formulation is described in [32] and these results prove that both results of DBSCAN and NRDD-DBSCAN are identical in case of three dimensions dataset as shown in Fig. 5.

The second experiment regarding the two features synthetic dataset described above, used a sample of 1000 instances and the

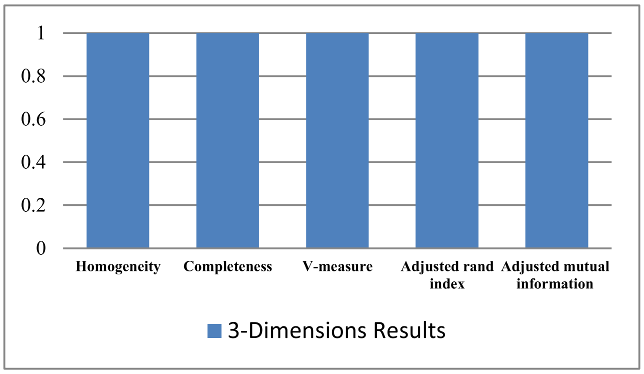


Fig. 5. NRDD-DBSCAN vs DBSCAN [3] in 3-D.

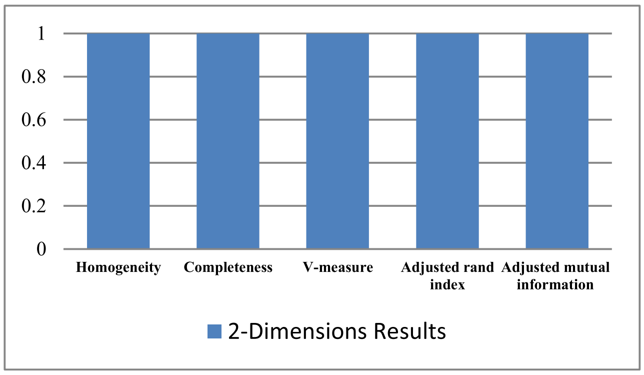


Fig. 6. NRDD-DBSCAN vs DBSCAN [3] in 2-D.

given parameters of the experiment are $MinPts = 3, \epsilon = 0.5$, labels_true and labels_pred, where labels_true is the ground truth class assignments of the samples and it's calculated by using scikit-learn's implementation of DBSCAN [33] and labels_pred is the results of NRDD-DBSCAN, the results are Adjusted Rand Index = 1.0, Adjusted Mutual Information = 1.0, Homogeneity = 1.0, Completeness = 1.0 and V-measure = 1.0 and that's proof that both results of NRDD-DBSCAN and DBSCAN are identical in case of two dimensions dataset as shown in Fig. 6.

The third experiment regarding the 25 features synthetic dataset described above, used a sample of 1000 instances and the given parameters of the experiment are $MinPts = 3, \epsilon = 0.5$, labels_true and labels_pred, where labels_true is the ground truth class assignments of the samples and it's calculated by using make_blobs method of scikit-learn's samples generator utility [31]. And labels_pred is the results of NRDD-DBSCAN, the results are: the estimated number of clusters = 3, Homogeneity = 1.0, Completeness = 0.985, V-measure = 0.992, Adjusted Rand Index = 0.996 and Adjusted Mutual Information = 0.985 as shown in Fig. 7.

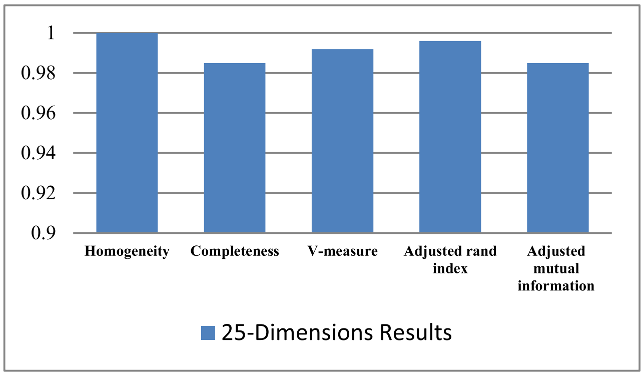


Fig. 7. NRDD-DBSCAN vs Gaussian blobs clusters [31] in 25-D.

Table 2
RDD-DBSCAN Vs. NRDD-DBSCAN.

	RDD-DBSCAN		NRDD-DBSCAN	
	2-D	25-D	2-D	25-D
Adjusted Rand Index	1.0	–	1.0	0.996
Adjusted Mutual Information	1.0	–	1.0	0.985
Homogeneity	1.0	–	1.0	1.0
Completeness	1.0	–	1.0	0.985
V-measure	1.0	–	1.0	0.992

As shown in Table 2. another comparison has been made between the results of RDD-DBSCAN implemented by I. Cordova and T. S. Moh [11] and the results of the proposed model NRDD-DBSCAN regarding the 2D and the 25D datasets, the comparison shows that both RDD-DBSCAN and NRDD-DBSCAN return identical results to the original DBSCAN algorithm when work with datasets of 2D but RDD-DBSCAN collapses when working with datasets of high dimensions, while NRDD-DBSCAN can work with datasets of high dimensions such as the datasets of 3D and 25D explained above.

5. Conclusion, limitations, and future work

NRDD-DBSCAN is introduced to resolve the issue of low dimensionality of RDD-DBSCAN introduced by I. Cordova and T. S. Moh [11] which has been implemented to scale DBSCAN horizontally using resilient distributed datasets RDDs. In this research, Three experiments have been made to evaluate the performance of NRDD-DBSCAN using 2d, 3d and 25 dimensions datasets, the 2d and 3d experiments prove that the results of scaled DBSCAN represented by NRDD-DBSCAN is identical to the original DBSCAN, while the 25d experiments show the performance and correctness of the scaled algorithm by comparing its results with make_blobs method of scikit-learn's samples generator utility results.

Finally the purpose of this research is to detect outliers and noises that affect the data quality of IoT technologies and applications, the proposed model NRDD-DBSCAN is implemented to deal with datasets of high dimensions in an efficient way, The steps of NRDD-DBSCAN have been discussed in details along with the algorithms used in implementation and with a visualized description of the details and how it works and communicate, and how these steps can be translated to an actual implementation, and verify the correctness of the scaled algorithm.

The main limitation of NRDD-DBSCAN model is that the data reduction and allocation phase uses principal component analysis to reduce the dimensionality of the datasets but PCA performs better when reducing the dimensionality of a linear highly correlated n-dimension data which constrained the model to handle non-linear data, so the future work is to improve the model to handle all types of data including linear highly correlated n-dimension datasets and non-linear n-dimension datasets.

Declaration of Competing Interest

The authors declare that they have no conflict of interest.

References

[1] Liu J, Yan Z. Fusion-An aide to data mining in Internet of Things. Information Fusion 2015;23(8):1–2.
[2] National Intelligence Council, Disruptive Civil Technologies — Six Technologies with Potential Impacts on US Interests Out to 2025— Conference Report CR 2008–07, April 2008, Available at www.dni.gov/nic/NIC_home.html.
[3] Ester M, Kriegel H-P, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD 1996;96(34):226–31.
[4] (2014, August 18) 2014 SIGKDD Test of Time Award. [Online]. Available: http://www.kdd.org/blog/2014-sigkdd-test-time-award.
[5] Chen M, Gao X, Li H. Parallel dbscan with priority r-tree. In: Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. IEEE; 2010. p. 508–11.
[6] Arlia D, Coppola M. Experiments in parallel clustering with dbscan. In: Euro-Par 2001 Parallel Processing. Springer; 2001. p. 326–31.
[7] Dai B-R, Lin I-C. Efficient map/reduce-based dbscan algorithm with optimized data partition. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE; 2012. p. 59–66.
[8] He Y, Tan H, Luo W, Mao H, Ma D, Feng S, Fan J. Mrdbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE; 2011. p. 473–80.
[9] Lulli A, Dell'Amico M, Michiardi P, Ricci L. In: Proceedings of the VLDB Endowment. p. 157–68.

[10] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association; 2012. p. 2.
[11] Cordova I, Moh T. DBSCAN on resilient distributed datasets. IEEE 2015:531–40.
[12] Mohamudally N, Mohaboob M. Building an anomaly detection engine (ADE) for IoT smart applications. Elsevier 2018;134:10–7.
[13] N Nesa, T Ghosh, I Banerjee, Outlier detection in sensed data using statistical learning models for IoT, in: IEEE Wireless Communications and Networking Conference (WCNC), 2018.
[14] Hasan M, Islam M, Zarif I, Hashem M. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. Elsevier 2019;7:1–14.
[15] Patwary M, Satish N, Sundaram N, Manne F, Habib S, Dubey P. PARDICLE: parallel approximate density-based clustering. IEEE 2014:560–71.
[16] Kim Y, Shim K, Kim M, Lee J. DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce. Elsevier 2014;42:15–35.
[17] Ashton K. (22 June 2009). That 'Internet of Things' Thing". Retrieved 9 May 2017, Available at http://www.rfidjournal.com/articles/view?4986.
[18] Li S, Xu LD, Zhao S. The internet of things: a survey. Elsevier 2014;54(15):243–59.
[19] He W, Xu LD, Li S. Internet of Things in Industries: A survey. IEEE 2014;10(4):1–11.
[20] Ciobanu RI, Cristea V, Dobre C, Pop F. Big data platforms for the internet of things. Springer; 2014. p. 3–34.
[21] G. Luo, X. Luo, T. F. Gooch, L. Tian and K. Qin, "A Parallel DBSCAN Algorithm Based On Spark," IEEE International Conferences on Big Data and Cloud Computing (BDCloud), pp.548-553, 2016.
[22] spark.apache. (2018, June, 22). [Online]. Available: http://spark.apache.org/docs/2.2.0/api/python/index.html.
[23] Gandomi A, Haider M. Beyond the hype: Big data concepts, methods, and analytics. Elsevier 2015;35(2):137–44.
[24] Wikipedia. (2015, April, 5) Scalability — Wikipedia, the free encyclopedia. Accessed 22-July-2004. [Online]. Available: http://en.wikipedia.org/wiki/Scalability.
[25] Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun ACM 2008;51(1):107–13.
[26] Dokmanic I, Parhizkar R, Ranieri J, Vetterli M. Euclidean Distance Matrices: Essential Theory, Algorithms and Applications. IEEE 2015;32(6):1–17.
[27] Jolliffe I. Principal component analysis. New York: Springer-Verlag; 1986.
[28] Lee YK, Lee ER, Park BU. Principal component analysis in very high-dimensional spaces. StatisticaSinica 2012;22:933–56.
[29] Zhang T, Yang B. Big data dimension reduction using PCA. In: IEEE International Conference on Smart Cloud. p. 152–7.
[30] UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+%28North+Jutland%2C+Denmark%29.
[31] (2018, September 1) Dataset loading utilities. [Online]. Available: http://scikit-learn.org/stable/datasets/
[32] (2018, September 2) Clustering performance evaluation. [Online]. Available: http://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation.
[33] 8, September 2) Clustering. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html