European Journal of Interdisciplinary Research and Development
Volume-08 Oct. - 2022
Website: www.ejird.journalspark.org ISSN (E): 2720-5746

IMPROVEMENT THE QUALITY OF IOT BIG DATA: AN OUTLIER DETECTION APPROACH

Mshtaq Talib Mahdi Ali1
Department of Physiology, Hammurabi College of Medicine,
University of Babylon, Babylon, Iraq
mailto:mushtaq.talib@uobabylon.edu.iq

Abstract
With the advent of the Internet, many concepts have been introduced in our technological life. One of the common and promising concepts that have attracted research communities is the Internet of Things (IoT). This concept assumes that objects (e.g., devices, sensors, processors, appliances, etc.) around us are connected and communicated with each other as a single network. The quality of data exchanged and its uncertainty are considered the main challenges that face developers when designing IoT models. This is due to the large-scale data generated by network objects that leads to redundancy, noise, and inconsistency in the collected data, which, in turn, yield a variety of issues. Moreover, IoT network is considered heterogeneous since different types of devices and applications are gathered to generate complex-considered data that is difficult to analyze. This data may follow anomalous behavior that leads to having abnormal data points, which impact the quality of data. The literature includes a lot of works that deal with the aforementioned issues. However, most of the approaches struggle the complexity and accuracy. Therefore, this work suggested a DBSCAN-based approach using resilient distributed databases in distinguishing abnormal/outlier data and maintain the IoT data quality. Three datasets are examined in the proposed approach, namely, 2D, 3D, and 25D. The results of the proposed approach are benchmarked with the literature. The findings showed the proposed approach outperformed the benchmarking in terms of addressing the low dimensionality and handling the large-scale data. Moreover, the proposed approach can accurately distinguish the abnormal/outlier data and improve the quality of data using resilient distributed datasets.

Keywords: IoT, Big data, Cloud Computing, Abnormal/Outlier Detection.

I. Introduction

Most of today's devices and technologies are designed with the ability to connect to the Internet. In this case, the infrastructure of the Internet of Things (IoT) has become true and adoptable [1][2]. Furthermore, collecting high-quality data from IoT objects is not an easy task to be performed. In this context, the IoT straggles issues related to many aspects such as data analysis issues. This is due to the heterogeneity of the IoT environment that leads to having outlier and abnormal data. In the literature, several approaches have been developed for overcoming this issue. Therefore, this work comes to overcome this issue and maintain high-quality data through detecting outlier data points using a "*Density-Based Spatial Clustering of Applications with Noise (DBSCAN)*" algorithm. The reason behind using the DBSCAN is the ability to detect particular shaped clusters. Also, it is able to sense outlier and noisy data. In addition to the aforementioned, DBSCAN is considered a simple algorithm and easy to implement with minimum cost consumed as well as it does not need user actions [3] [4]. This algorithm was designed to be suitable to work on a single machine. However, it struggles to deal with large-scale data and the situation when it is needed to migrate data to multiple terminals within a network [5][6]. According to the literature, the MapReduce technique is one of the most common approaches that has been used for scaling algorithms in the literature such as the works of [7] and [8]. Moreover, the authors of [9] suggested a method called "NG-DBSCAN", which was based on the MapReduce technique and DBSCAN algorithm that permitted running the algorithm over the Hadoop. However, the researchers in [10] found that MapReduce struggles with the issues that its actions and processes are performed through the file system. This specific issue leads to a gain high delay and high computational cost. To mitigate this issue, the authors of [10] suggested an approach that used abstraction for in-memory computing. This kind of approach is called Resilient Distributed Datasets (RDDs).

In RDDs, huge data is processed by utilizing the main memory (RAM). The actions, processing, and transformations tasks take the benefit of the cache memory aiming to speed up the computations and minimize the cost. This kind of algorithm has been intensively used in the literature. For example, the researchers in [11] developed a new method called RDD-DBSCAN of which the goal was to address the issues of MapReduce with DBSCAN. Also, they aimed to implement a real parallel environment that is able to distribute the loads using RDDs.

The MapReduce technique has also been used in developing anomaly detection applications the work under the IoT (e.g., Anomaly Detection Engine (ADE) [12]). The purpose of this application is to detect abnormal behavior of IoT collected data and distinguish outliers using time-series models [12]. Also, advanced statistical models can also be used for this specific purpose such as the work of Nesa et al. [13]. Moreover, machine learning techniques can also be involved in developing a model for detecting anomalous data. Hasan et al. [14] measured the performance of different machine learning algorithms (e.g., "*Support Vector Machine (SVM)*", "*Random Forest (RF)*", "*Logistic Regression (LR)*", "*Decision Tree (DT)*", and "*Artificial Neural Network (ANN)*"). The authors of [14] performed the benchmarking based on the appropriateness of the approach in terms of detecting anomalous data in IoT.

Another limitation that affects the use of RDD-DBSCAN is that it can deal only with low-dimensional data. Hence, the contribution of this work is to overcome this limitation and develop a new approach for detecting outliers in IoT applications. This can be performed by horizontally scaling the DBSCAN algorithm. This feature enables the algorithm to deal with multiple terminals (node) and distribute loads accordingly using RDDs. Another algorithm is involved that is called "*NRDDDBSCAN*" aiming to address the limitation in low dimensionality of "RDD-DBSCAN". Furthermore, the Euclidean approach was also used in the literature to efficiently deal with the high dimensionality of data such as the works [15] and [16] that used "*Message Passing Library Interface (MPI)*" for parallel processing and *Map Reduced* techniques respectively.

This paper is organized as follows: Section two describes the research method followed by the authors and the steps of the approach. Section 3 demonstrates the obtained results using the proposed approach as well as the benchmarking with other approaches in the literature. Also, this section discusses the obtained results and highlights the main points. Section 4 concludes this work and suggests some recommendations to readers with potential future work.

II. Research Method

A. Concepts and Techniques

The term "Internet of Things (IoT)" was first introduced as a group of objects that are connected and communicated using RFID technology [17]. The general architecture of the IoT comprises four layers according to [18] and [19] as follows: *Perception Layer* includes the hardware objects such as sensors, processors, actuators, and/or even smart devices. These objects are used for different purposes such as measuring a particular event within the environment, collect and generating data, or used for data transfer within the IoT network. *Network Layer*, involved in the connections and the communications among IoT objects (e.g., sending and receiving tasks). *Service Layer* is used for monitoring and managing IoT services for users and applications. Finally, the *Interface Layer* coordinates the communications among homogeneous and heterogeneous devices. According to these layers, developers should be aware of which layer they are dealing with during IoT applications development.

On the other hand, as mentioned, a large-scale of data is generated by the IoT devices and this scale is increased over time. This leads to having issues related to the security of data, storage capacity, data analysis, privacy, etc. [20][21]. In the literature, a lot of tools have been developed and used to handle IoT data such as Map Reduce, Hadoop, and RDDs [22][23]. The performance of Map Reduce is affected when it is needed to have iterative operations on the data. This is because Map Reduce writes data on disk after each map, which makes it struggle the shuffling and reducing processes in terms of computation cost [24][25]. Addressing this issue can be performed using the resilient distributed datasets RDDs approach. It has many features such as being fault-tolerant, handle parallel operations, and does not need to store data in Hadoop Distributed Files System, which makes it faster. These features can be considered a powerful tool alternative to Map Reduce. Therefore, this work makes use of the RDDs to design NRDD-DBscan for efficiently detecting IoT data outliers.

Furthermore, the DBSCAN algorithm can be horizontally scaled and the large-scale data is partitioned into many portions (chunks). Given that D is the whole data that is intended to be partitioned, then each portion is termed *CH* where CH_1, CH_2, …, CH_n are all the portions and can be formulated as follows:

$$D = CH_1 \cup CH_2 \cup \ldots CH_n \quad (1)$$

Where *n* is the total number of chunks in D. All these chunks should be processed across multi nodes and each of which should apply the DBSCAN independently. The DBSCAN creates clusters based on the common features that are shared among the data points. Outliers data points are detected *if and only if* there are no shared features in common with other data points and can be formalized as follows:

$$CH = (c_1 \cup c_2 \cup \ldots c_m) \cup (o_1 \cup o_2 \cup \ldots o_l) \quad (2)$$

Where *CH* denotes a particular chunk and the clusters are represented by c_1, c_2, …, c_m, and the outliers are represented by o_1, o_2, …, o_l. The *m* reflects the number of clusters that is not necessary to equal to the number of outliers *l*. It should be mentioned that an outlier is not necessary to exist. Now, each cluster is represented as follows:

$$c = x_1 \cup x_2 \cup \ldots x_k \quad (3)$$

Where *c* denotes a cluster and $x_1, x_2, \ldots., x_k$ are the data points with a total number of *k*. The DBSCAN algorithm is based on two parameters: MinPts and ε. The former represents the minimum number of neighbors (points) that are needed to form a cluster. The latter represents the radius of the cluster. Moreover, the DBSCAN includes concepts that should be well-understood before implementing it. For instance, "*core point*" is a point that belongs to *c* and has neighbors within its radius, "*border point*" is a point in *c* when but it is not a core, "*directly reachable*" is a point that is directly reachable by a *core point* and positioned in its neighbors, "*density reachable*" is a point that is directly reachable by a *core point* and indirectly reachable by another *core point* and these two core points are directly reachable to each other.
One of the advantages of the DBSCAN is that it starts with unlabeled data points and its output is labeled data points. In addition, the detected outliers are flagged with a value of (-1). In this work, determining the neighbors of a particular point can be performed by calculating the distance among points. This step is done using the "*Euclidean Distance Matrix (EDM)*" [26].

B. The Proposed Approach

The proposed NRDD-DBSCAN aims to detect outliers with the support of RDDs. The proposed approach can be applied to n-dimensions and the implementation is performed using Apache Spark. Three phases are used in the proposed approach: 1) data allocation and reduction, 2) clustering and 3) aggregation. After performing these steps, outliers should be detected (if any). The proposed approach (NRDD-DBSCAN) is described in Algorithm 1:

Algorithm 1: Steps of the proposed approach.

Goal: *Outliers Detection in Data Points*

Input: *A set of data points* ($x= x_1, x_2, \ldots., x_k$)

Output: *A set of outlier points* ($o= o_1, o_2, \ldots., o_l$)

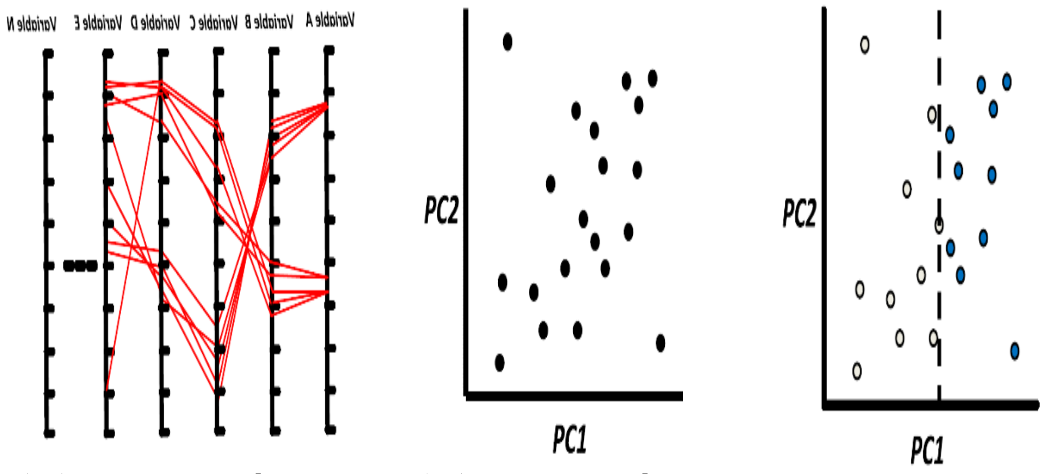
Steps:

```
Step#1:   SET labeled_chunks TO ∅
Step#2:   Data Portions =Evenlychunks (X, Node_number, Row_Number)
Step#3:   FOREACH chunk in Data Portions DO
Step#4:             $p_n$ =DBSCAN (chunk, ε, MinPts)
Step#5:             labeled_chunks = labeled_chunks $\cup p_n$
Step#6:   ENDFOR
Step#7:   AggregatedChunks = aggregation (labeled_chunks, ε, MinPts)
Step#8:   RenamedPoints = renaming (AggregatedChunks)
Step#9:   Outliers = all the (-1) in RenamedPoints
```

Now, the three phases are described in detail as follows.

Data Allocation and Reduction Phase

Two steps are included in this phase, the first one is the data allocation process that needs the data point to be represented in two dimensions aiming to cluster them (see Figure 1). The main reason behind this representation is that NRDD-DBSCAN can efficiently partition the 2D data. Therefore, reducing the data is crucial and can be implemented using the "*Principle Component Analysis (PCA)*" algorithm [27][28]. PCA is one of the most efficient data reduction approaches that are widely used for this specific purpose. However, the classic approach of PCA cannot be implemented due to the high volume of data that consumes a large amount of memory that cannot be handled in regular computer systems [29]. The implementation, therefore, is performed using the PCA algorithm of "*Apache Spark APIs*" (pyspark) [22]. Then, the data allocation procedure starts to load the two-dimensional data and perform a splitting process that divides the data into many evenly-chunks. Here, it is worth mentioning that the number of portions should be compatible, in terms of size, with the available memory space. The reason behind this procedure is to avoid the RDDs to read from the secondary memory and makes the process slower, which is not sufficient for the proposed approach. Algorithm 2 describes the details of the whole process.



(#1) n-D converted into a 2-D dataset using PCA

(#2) PC1 is sorted in ascending order and divided using evenly chunks

(#3) N chunks are assigned to N nodes.

Figure 1: The processes of data allocation and data reduction.

Algorithm 2: The description of evenly chunks.

Goal: *Create N chunks to be distributed to N nodes within the network.*

Input: *A set of unlabeled data points* ($x= x_1, x_2, …., x_k$)

Output: *A set of chunks* ($c= c_1, c_2, …., c_l$), each chunk contains data points that fit the node's memory space.

Steps:

```
Step#1:   SET chunks TO ∅
Step#2:   Reduce_Dimensionality = PCA (X,2)
Step#3:   Sorted_Data = AscendingSort(PC1)
Step#4:   #_of_Points_Per_Node = Ceiling (Sorted_Data.Length / Nodes_Number)
Step#5:   IF (#_of_Points_Per_Node > Rows_Number) THEN
              Nodes_Number = Sorted_Data.Length / Rows_Number
              CALCULATE #_of_Points_Per_Node
          ENDIF
Step#6: FORALL n in Nodes_Number DO
              c = Split_Points(Sorted_Data)
Step#7: Chunks = Chunks ∪ c
Step#8:  RETURN Chunks
```

Local Clustering Phase

In the previous phase, a set of chunks is created where each of which includes unlabeled data points. The chunks are now ready to be assigned to nodes that, in turn, will use the DBSCAN independently. The output of applying the DBSCAN is labeled data points. These points are labeled as a border data point, core data point, or outlier data point. The first two types of points will be joined to the same or different clusters based on the number of features in common among the points.

Data Aggregation and Renaming Phase

The aggregation process takes the chunks as its input, which includes labeled data points of one of the three defined types (core, border, or outlier). As mentioned, all the outliers are flagged with a value of (-1), while core and border points are flagged with their portion and chunk. For instance, if a data point is flagged with P_1c_0, it means that this data point belongs to portion 1 and chunk 0. The main issue in the aggregation process is when dealing with edge points. These kinds of points are positioned on the border of portions, which makes it a candidate to be an outlier point. Based on Figure 2, if the value of MinPts is 2 and the value of radius is 1, then the data point *x_1* belongs to P_1c_1, but it also belongs to P_2c_0. Similarly, the data points *x_6* and *x_7* have almost seemed like outliers, but they belong to the same cluster of *x_4* and *x_5*. Therefore, the aggregation process will process each portion and starts with portion 1 by obtaining its PC1 which is x_1 (the last point in portion 1). The position of x_1 is considered the startline of that portion. The startline of all the portions is used to detect the edges of these portions. The process starts with subtracting ("*goBackOneEpsilon*") and adding ("*goForwardOneEpsilon*") one epsilon from the startline points. After that, all the in-between points are able to share more than one cluster. Then, using the Euclidian Distance Matrix is applied for each data point aiming to determine the neighbors. Here, when the number of neighbors is greater than or equal to the MinPts, four cases should be considered for each neighbor point as described in detail in Algorithm 3. After completing the aggregation process, each point in the data is assigned to a cluster (see Figure 3). The renaming process is performed after the aggregation and aims to rename the labels of the points. In other words, instead of using the notion P_nC_n, the renaming process makes it in the form of c_n as described in Algorithm 4.

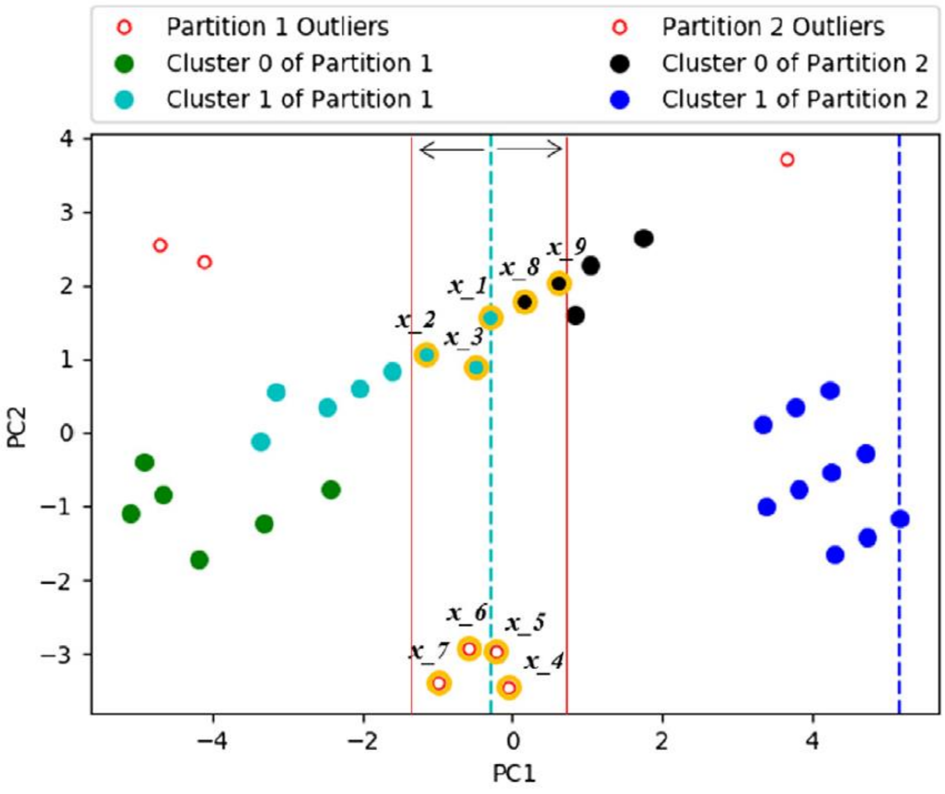
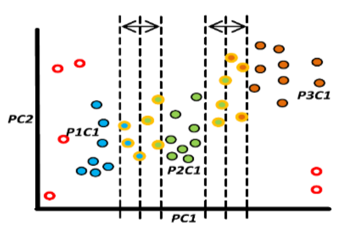
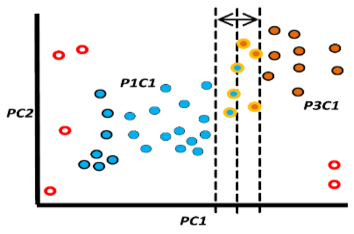


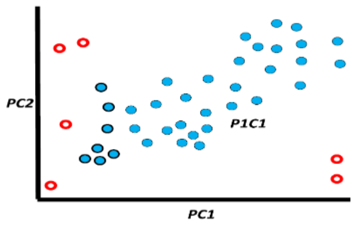
Figure 2: The process of data aggregation.



(#1) drawing startline for each chunk and expand it to be (+ε and -ε) aiming to gather the edge points.



(#2) When ε=1and MinPts=3, then exploring the edges that share more than one cluster for all the chunks.



(#2) Obtain the output of labeled data points and these labels can be core, border, or outlier.

Figure 3: Process steps of data aggregation in the proposed approach.

Algorithm 3: The process of data aggregation.

Goal: *Aggregating data points.*

Input: *A set of labeled data points* ($P= P_1, P_2, …., P_n$), the radius ε, and MinPts

Output: *A set of points* ($Y= y_1, y_2, …., y_l$) where *y* is labeled points with flags of border, core, and outlier.

Steps:

```
Step#1:   FOREACH PART in (P) – Last portion DO
Step#2:   Startline = LastElement(PART)
Step#3:   goBackOneEpsilon = startline - ε
Step#4:   goForwardOneEpsilon = startline + ε
Step#5:   borderPoints = GetPoints between (goBackOneEpsilon, goForwardOneEpsilon)
Step#6:   FOREACH P in borderPoints DO
      neighbors=GetNeighbors(P,ε)
      IF neighbors.length >= MinPts THEN
         FOREACH neighborPoints in neighbors DO
            IF portion(neighborPoint) !==portion(P) OR (cluster(neighborPoint)==(-1) & cluster(P) ==(-1)) THEN
            IF cluster(P) ==(-1) & cluster(neighborPoint) !==(-1) THEN
                  Cluster(P)=cluster(neighborPoint)
                  portion(P)=portion(neighborPoint)
            ELSE
                  IF Cluster(neighborPoint)==(-1) & cluster(P)!==(-1) THEN
                        Cluster(neighborPoint)=cluster(P)
                        Portion(neighborPoint)=portion(P)
                  ELSE
                        IF cluster(neighborPoint) ==(-1) & cluster(P) ==(-1) THEN
                              NEW_CLUSTER=CreateNewClusterID()
         NEW_PORTION=CreateNewPortion()
                              Cluster(P)= NEW_CLUSTER
                              portion(P)= NEW_PORTION
                              cluster(neighborPoint)=cluster(P)
                              portion(neighborPoint)=portion(P)
                        ELSE
                              Points_neighbor_cluster=GetPointsOF(cluster(neighborPoint),
portion(neighborPoint))
                              FOREACH P in (PonitsOfNeigborCluster) DO
                                    Cluster(P)=cluster(neighborPoint)
                                    Portion(P)=portion(neighborPoint)
                              ENDFOR
                        ENDIF
                  ENDIF
               ENDIF
            ENDFOR
         ENDIF
         ENDFOR
Step#7: aggregatePoints=$P_1 \cup P_2 \cup …… P_n$
Step#8: RETURN aggregatePoints
```

Algorithm 4: The process of renaming data points.

Goal: *Renaming the aggregated data points*

Input: *A set of labeled data points* ($X= x_1, x_2, …., x_k$)

Output: *A set of points* ($X= x_1, x_2, …., x_k$), where core and border points are renamed as c_n (Cluster_ID) and ourliers as (-1).

Steps:

```
Step#1:   #OfPortions=max(portion(x))
Step#2:   #OfClusters=max(cluster(x))
Step#3:   NEW_CLUSTER=uniqueValue
Step#4: FOREACH portion in range(1, #OfPortions) DO
            FOREACH Cluster_ID in range(0, #OfClusters) DO
                filteredPoints=filteredPointsBy(x, cluster_ID, Portion_ID)
                IF (filteredPoints.length > 0) THEN
                    Cluster(filteredPoints)= NEW_CLUSTER
                    NEW_CLUSTER=New_uniqueValue
                ENDIF
            ENDFOR
        ENDFOR
Step#5: REUREN x
```

III. Results and Discussions

A. Datasets Used

The proposed NRDD-DBSCAN was implemented using "*Apache Spark*" which simulated the RDDs using Python programming language (pyspark API) [22]. The evaluation of the proposed approach was based on performing three main experiments as follows:

- Experiment_1: it was performed using a dataset ("*non-synthetic*") [30] of four attributes and 434,874 instances.
- Experiment_2: it was performed using a dataset ("*synthetic*") of one-million records, three centers, and two features.
- Experiment_3: it was performed using a dataset ("*synthetic*") of one-million records, three centers, and twenty-five features

The use of two synthetic datasets in the experiments enabled to measure the performance of the proposed approach. It should be mentioned that the two datasets were created using the "*make_blobs method of scikit-learn's samples*" [31]. The evaluation measurements used in this work are "*Adjusted Rand Index (ARI)*", "*Adjusted Mutual Information (AMI)*", "*Homogeneity (Ho)*", "*Completeness (Co)*", and "*V-measure (Vm)*". Using these measurements needed to know "*label_true*" as ground truth labels [32].

B. Experimental Results

Experiment_1: In this experiment, a sample of 100 instances was used with parameters values of $\varepsilon=1$ and $MinPts=3$. Also, "*labels_true*" and "*labels_pred*" were used, where the former was the "*ground truth*" of the samples as presented in [33]. The latter represented the results of the proposed approach. Practically, the DBSCAN is applied after excluding the "*OSM_ID*" column. After that, the columns of "*LONGITUDE*", "*LATITUDE*", and "*ATTITUDE*" were normalized and scaled aiming to address the *n-dimensionality* issue. The findings revealed four clusters (c0, c1, c2, and c3), while the outliers were defined as (-1) labels as well as the cluster column reflected the values of "*labels_ true*". Now, to compute the "*labels_pred*" defined by NRDD-DBSCAN, the simulation environment was prepared as a cluster of two "*Virtual Machines (VM)*" of Linux. Where each VM has two processors and the main memory of 4096 MB and secondary storage of 20 GB. Based on the obtained results, the points were clustered into four clusters (100, 200, 300, and 400) and (-1) for the outliers, and the cluster column was the values of "*labels_pred*". The results of the measurements described in the previous section were as follows:

Table 1: The results of the evaluation measurements in Experiments_1 for a 3-D dataset.

ARI	AMI	Ho	Co	Vm
1.0	*1.0*	*1.0*	*1.0*	*1.0*

These results proved that the results obtained from DBSCAN and the proposed are similar in the case of a 3-D dataset.

Experiment_2: The second experiment used synthetic datasets and a sample of 1000 instances, the values of the parameters were; ε=0.5 and MinPts 3. The values of "*labels_pred*" were the results of NRDD-DBSCAN. Interestingly, similar results of Experiment_1 were obtained in terms of the measurements used (see table 2). This proved that both results of "*DBSCAN*" and the scaled one were in agreement in the case of 2-D datasets.

Table 2: The results of the evaluation measurements in Experiments_2 for a 2-D dataset.

ARI	AMI	Ho	Co	Vm
1.0	*1.0*	*1.0*	*1.0*	*1.0*

Experiment_3: The third experiment was performed on 25 features synthetic dataset. A sample of 1000 instances were considered in this experiment and the parameters were *ε=0.5* and *MinPts 3*. The "*labels_true*" was the ground truth class and computed using the "*make_blobs method of scikit-learn's samples*" generator utility [31]. The "*labels_pred*" was the results of NRDD-DBSCAN, which represented the estimated number of clusters which was 3. Table 3 presents the results of Experiment_3 I term of the measurements used.

Table 3: The results of the evaluation measurements in Experiments_3 for a 25-D dataset.

ARI	AMI	Ho	Co	Vm
0.996	0.985	*1.0*	0.985	0.992

Based on Tables 1, 2, and 3, a comparison was performed for the performance of the three experiments. It can be observed that both RDD-DBSCAN and NRDD-DBSCAN with 2-D datasets provided similar results to the original version of the DBSCAN. However, the RDD-DBSCAN struggled to deal with high-dimensional datasets. On the other hand, the NRDD-DBSCAN has the ability to deal with high dimensional datasets and produce efficient results as observed in this work when dealing with 3-D and 25-D datasets.

IV. Conclusions

This work suggested an approach for two main purposes, namely, addressing the issue of low dimensional datasets and outlier detection. The proposed NRDD-DBSCAN approach was examined using three experiments. These experiments were also utilized to evaluate the performance of the proposed approach against other approaches in the literature. The NRDD-DBSCAN used 2-D, 3-D, and 25-D datasets. The 2-D and 3-D datasets reflected identical results for both NRDD-DBSCAN and its original version (DBSCAN). On the other hand, the 25-D dataset, in the experiments, was successfully implemented by the proposed approach. As mentioned, this work aimed to detect noisy and outlier data points that impact the quality of data exchanged in the IoT. The suggested approach was modeled to efficiently deal with high-dimensional datasets. The results of the proposed approach are promising and can be adopted by IoT developers and maintain data quality. However, this work has a main drawback, which is the use of PCA for dimensionality reduction that makes the proposed approach struggle dealing with nonlinear data. Therefore, as future work, there is an important area of improvement in this work, which is developing the approach to be able to handle nonlinear and highly correlated n-dimensional data.

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 highdimensional 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://scikitlearn.org/stable/datasets/
32. (2018, September 2) Clustering performance evaluation. [Online]. Available: http://scikit-learn.org/stable/modules/clustering.html#clusteringperformance- evaluation.
33. 8, September 2) Clustering. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html