Research Journal of Applied Sciences, Engineering and Technology 16(2): 77-87, 2019
DOI:10.19026/rjaset.16.6003
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2019 Maxwell Scientific Publication Corp.
Submitted: December 27, 2018 Accepted: February 17, 2019 Published: March 15, 2019

Research Article

Distributed Anomaly Detection Over Big Data

Mohamed Sakr, Walid Atwa and Arabi Keshk
Faculty of Computers and Information, Shebeen El Kom, Menofia, 32511, Egypt

Abstract: This study aims to solve the problem of detecting anomalies in big data. A border-based Gird Partition (BGP) algorithm was proposed. The BGP algorithm focuses on calculating the Local Outlier Factor (LOF) for big data in a distributed environment. It splits the data into intersected subsets, then allocates these subsets to the slave nodes in a distributed environment. Some parts of these subsets are replicated between slave nodes. The slave nodes calculate the LOF for each subset that it owns. The splitting of the data between the slave nodes is done in grid-based without considering the size of the data that will be assigned to every slave node. The BGP algorithm results in un-balanced distribution of the subsets between slave nodes. To overcome this problem a modification on the BGP algorithm is proposed to take in consideration the size of the data that will be assigned to every slave node. The modified algorithm called Balanced boarder-based Gird Partition algorithm (BBGP). BBGP splits the data between the slave node equally. So that all the slave nodes will do balanced processing for calculating the LOF for the data. In the end, we evaluate the performance of the two algorithms through a series of simulation experiments over real data sets.

Keywords: Anomaly detection, big data, distributed environment, local outlier factor, outlier detection

INTRODUCTION

Nowadays there is a huge growth in the area of Big Data. A lot of information flows on the internet and a lot of data are generated every day in our life. For many social websites (e.g., Facebook and twitter), a huge amount of information is generated every day. This information must be processed and filtered to detect the suspicions one from them. Suspicions information may be noise or wrong data or maybe up normal one. The process of detecting these types of information are called outlier detection. According to (Hawkins, 1980), "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism". Many other studies and definitions have been proposed for outlier detection e.g., top-n outlier (Ramaswamy *et al*., 2000), DB-outlier (Knox and Ng, 1998) and density-based outlier (Breunig *et al*., 2000).

There exist lots of algorithms for the outlier detection. Most of them focus on the centralized processing of the data. Due to the huge increase in data generated every day, this data will take a lot of time for processing and detecting outliers. There are many applications that time will be very critical for them (e.g., Credit card Fraud detection). So, there is a need for processing this data in a distributed environment, not a centralized one.

This study focuses on the problem of processing data in a distributed environment for detecting outliers. We use density-based outlier (Breunig *et al*., 2000) for detecting outliers. Density-based outlier has advantages over Some other algorithms. Other algorithms label every tuple as an outlier or not (Ramaswamy *et al*., 2000) (Knox and Ng, 1998), on the other hand, density-based algorithms measure the degree of being an outlier for a tuple p with respect to its neighbors. For measuring this degree for p we calculate its Local Outlier Factor (LOF). LOF represents the degree of a tuple to be outlier *w.r.t* its neighbors. Many real-world applications proved that measuring the degree of outlierness for p is more meaningful that marking p as an outlier or not. There exist two studies in solving the problem of distributed outlier detection (Lozano and Acufia, 2005) (Bai *et al*., 2016). The authors in (Lozano and Acufia, 2005) analyzed the process of calculating the LOF and find that the step to calculate the KDNeighbors (which is the matrix that contains the elements of dataset D with its respective k-neighbors) is the most exhaustive step that takes time and processing power. They proposed a master-slave solution for this problem that offloads all the KDNeighbors calculations to the slaves to calculate it. The authors in Bai *et al*. (2016) also proposed an algorithm that also solves the problem of calculating LOF using distributed environment. They divide all the datasets into grids and

Corresponding Author: Mohamed Sakr, Faculty of Computers and Information, Shebeen El Kom, Menofia, 32511, Egypt
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

sends these grids to be processed by the slave nodes. Their algorithm needs a communication between all the slaves to handle all the neighbor tuples.

In this study, we proposed practical approaches for calculating the LOF for tuples in a distributed environment which can be summarized as follows:

We proposed a border-based Gird Partition (BGP) algorithm which is based on the GBP on Bai *et al*. (2016), this algorithm partition the dataset into subsets and add a border to each subset of the dataset. The tuples in these borders are replicated in each adjacent subset.

We modified our BGP algorithm and proposed a Balanced boarder-based Gird Partition algorithm (BBGP). BBGP algorithm partition the tuples between slaves in a balanced way to distribute the workload to all the slaves equally and utilize the resources in every slave.

We evaluate the performance of the proposed approaches through a series of simulation experiments over real data sets. The experimental results show that our proposed approaches give better results for calculating the LOF for the tuples in comparing with Bai *et al*. (2016).

Many approaches for outlier detection were proposed, depending on the type of model they learn (non-parametric model or statistical) (Rajasegarar *et al*., 2008) Non-parametric techniques have a lot of types of data clustering, Distance-based, Density-based and rule-based approaches. data clustering approaches aim to find groups of similar data points where each group of data points is well separated. This approach was first intended to cluster a group of points but after that, it was used to detect outliers e.g., Aggarwal *et al*. (2003), Cao *et al*. (2006) and Guha *et al*. (2003). Distance-based approaches were proposed to detect outliers in a group of points based on the distance between points. Points out of the distance are supposed to be outliers e.g., Knox and Ng (1998). These approaches have an important problem with finding the outliers in a group of points with multi distances. Density-based approaches solve the problem of the distance based approaches by finding the outliers depend on the density of each group of points. Each point calculates its density depends on the other points near to it. The author in Breunig *et al*. (2000) proposed a number that measures the degree of each point of being an outlier this number is called LOF (local outlier factor). Some techniques improve the detection accuracy of LOF by changing the way k-NNs are computed (Tang, *et al*., 2002) so that it can cover a wider range of outlier types and finding a symmetric neighborhood relationship (Jin *et al*., 2006).

For the problem of distributed outlier detection there exist two studies (Lozano and Acufia, 2005; Bai *et al*., 2016). In Lozano and Acufia (2005) the authors proposed a distributed algorithm for calculating LOF. They analyzed the LOF algorithm and find that the most exhaustive step that requires time is the step of calculating *KDNeighbors* (which is the matrix that contains the elements of dataset D with its respective *k-neighbors*). For this reason, they attempt to paralyze this step. The architecture is composed of master and slaves, the master distributes the data to all the slave. Each slave calculates *KDNeighbors* for his local data and sends the results to the master. The master collects the partial *KDNeighbors* and finds the *KDNeighbors* matrix, then computes the reachability and LOF. Clearly, this approach is not suitable for distributed outlier detection on large-scale data, because all the data are combined in the last step and processed to compute the LOF for each tuple (centrally on the master node). The master node becomes a bottleneck when the data is large. In Bai *et al*. (2016) the authors also proposed a distributed algorithm for calculating LOF. Their architecture is also composed of master and slaves. They proposed Gird-Based Partition algorithm (GBP) for data portioning between slaves. In GBP, the author first splits the whole dataset into isometric grids considering the dimensionality of the dataset *d*. for each dimension the author splits this dimension into several isometric segments (the number of segments is denoted by *s*). Then, the whole space is split into s^d grids. The author set *s* as the smallest number that satisfies $s^d \geq |N|$ where $|N|$ is the number of slave nodes.

They propose also the Distributed LOF Computing method (DLC). Each slave generates LOF for most of the tuples in it. The boarder tuples are then transmitted between slaves to calculated its LOF efficiently then all the LOFs for the tuples are sent back to the master node. This approach has some drawbacks in two parts, first in the Gird-Based Partition algorithm the authors partition the data to the slaves without taking in consideration the data distribution which makes the slaves unbalanced, some slaves have a lot of data with respect to the others. Second, the cross-border tuples are sent over the network many times which consumes network and every time a new tuple is sent to the salve the neighbors for this tuple recalculate the LOF for them which consumes time and processing power.

Recently, there also emerge some outlier detection algorithms for special purposes, such as high dimensional data (Aggarwal and Yu, 2001), streaming data (Kontaki *et al*., 2011) and uncertain data (Aggarwal and Yu, 2008).

MATERIALS AND METHODS

Local Outlier Factor (LOF): In this section, we will use the density based outlier detection algorithm to detect the outlier tuples. To measure the outlierness of each tuple we will use the LOF algorithm. The process of calculating the LOF takes a lot of time and processing power. We will use a distributed environment solution to distribute the processing between many nodes. So that they calculate the LOF in

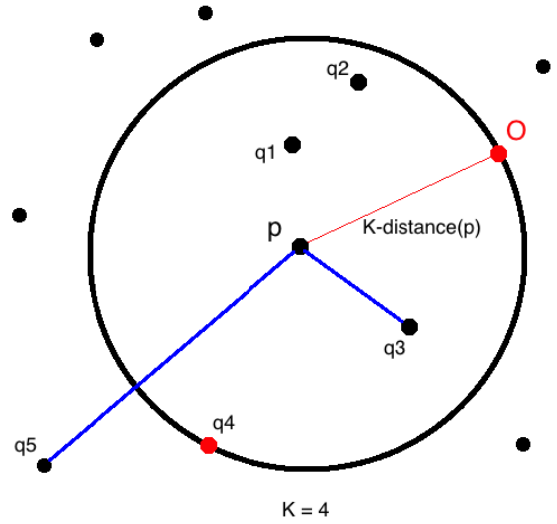

Fig. 1: The k-distance of point p given k = 4

parallel which will reduce the time. We will first describe how we will calculate the LOF of tuples then we will give in detail the system flow for the distributed calculation of the LOF in a distributed environment in the next section.

Given a dataset *D* in d-dimensional space (the size of *D* is |*D*|), a tuple *p* is donated as *p* = *{ p[1],p[2],…p[d]}*. The distance between two tuples *p,q* is:

$$\mathrm{dis}(p,q) = \sqrt{\sum_{i=1}^{d}(p[i]-q[i])^2} \quad (1)$$

before we describe how the LOF is calculated there must be some definitions that need to be explained.

Definition 1: (*k-distance* of tuple *p*) Given a positive integer *k*, the *k-distance* of a tuple *o* (denoted as $dis_k(o)$) is the furthest distance among the *k-nearest neighbors* of a data point *p* as shown in Fig. 1.

K-distance (*p*) is defined as the distance *dis*(*p,o*) between *p* & *o* such that:

for at least *k* objects $q \in D\backslash\{p\}$ it holds that $d(p,q) \leq d(p,o)$

for at most *k*-1 objects $q \in D\backslash\{p\}$ it holds that $d(p,q) < d(p,o)$

Definition 2: (*k-distance* neighborhood of a tuple). Given a positive integer *k*, the *k-distance* neighborhood of a tuple *p* is a set of *k-nearest* neighbors *i.e.,* the data points closer to *p* than *k-distance(p)* $Neigh_k(p) = \{ q | dis(q,p) \leq dis_k(p)$ and $q \neq p \}$. Which in Fig. 1 the $Neigh_k(p)$ is the points *{q1, q2, q3, q4, O}*. notice that the number of points in the $Neigh_k(p)$ may be larger than *K* as shown in Fig. 1 this happens because *dis(p,o) = dis(p,q4)*

Definition 3: (reachability distance of a tuple *o w.r.t. p*). Given a positive integer *k*, the $Rdis_k(o,p)$ is either the radius of the neighborhood of *p* if *o* in the neighborhood of *p* or the real distance from *o* to *p*:

$$Rdis_k(o,p) = max\{dis_k(p), dis(o,p)\}$$

As shown in Fig. 1 the $Rdis_k(q3,p) = dis_k(p)$ as point *q3* is in the neighborhood of *p*. on the other hand the $Rdis_k(q5,p) = dis(q5,p)$ as point *q5* is not in the neighborhood of *p*.

Definition 4: (local reachability density of a tuple). For a given parameter *k*, the local reachability density of a tuple p is defined as:

$$\mathrm{LRD_k}(p) = 1/\frac{\sum_{o \in \mathrm{Neigh_k}(p)} \mathrm{Rdis_k}(p,o)}{|\mathrm{Neigh_k}(p)|}$$

The $LRD_k(p)$ is the inverse of the average of the reachability distances of *p w.r.t.* the tuples in $Neigh_k(p)$.

Definition 5: (local outlier factor of a tuple). Given an integer *k*, the local outlier factor of a tuple *p* is:

$$\mathrm{LOF_k}(p) = \frac{\sum_{o \in \mathrm{Neigh_k}(p)} \frac{\mathrm{LRD_k}(o)}{\mathrm{LRD_k}(p)}}{|\mathrm{Neigh_k}(p)|}$$

The local outlier factor of a tuple *p* is the average of the ratio of the local reachability density of *p* and those of *p*'s *k-distance* neighbors. As Breunig *et al.* (2000) describes each tuple will be given a degree that of being outlier using the LOF instead of mark each tuple as an outlier or not. The higher the LOF mean that this tuple is supposed to be more outlier than the tuples with the lower LOF.

System flow: In this section, we will describe how we will calculate the LOF in a distributed environment. First, we will describe the components of the system after that we will go into detail about how the system calculates the LOF for each tuple.

We designed a distributed environment that consists of one master node and slave nodes. The master node is responsible for partitioning the dataset and distribute it between slave nodes. Each slave node will compute the LOF for its portion of the dataset and sends the results back to the master node. There exist many algorithms that make the master node processes the results sent from the slaves to finish computing the LOF like in Lozano and Acufia (2005). On the other hands in our architecture, the master node is only responsible for partitioning the dataset between slave nodes and schedule the processing between them. And all the LOF computations are done in the slave nodes.

We first split the dataset into subsets and assign each subset to a slave node. To do so We propose a

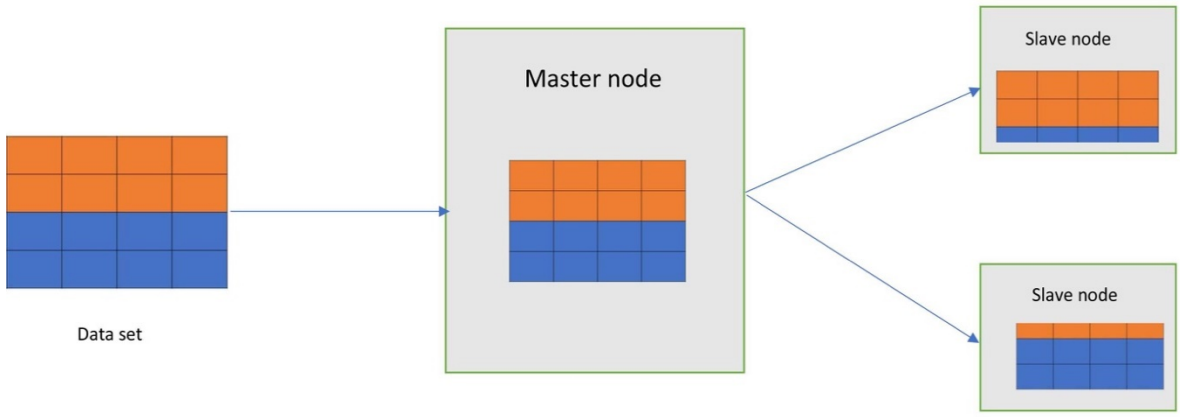

Fig. 2: System flow

border-based Gird Partition (BGP) algorithm which is based on the GBP on Bai *et al*. (2016), this algorithm partition the dataset into subsets and add a border to each subset of the dataset. The tuples between this border and the subset boarder are replicated in both subsets. After that, each subset is allocated to a slave node which calculates the LOF for this subset only. Finally, each slave node sends its results back to the master node. As shown in Fig. 2 the data set consists of 4 rows 2 orange and 2 blue rows, we divide the rows between two slave nodes. 2 rows for first slave node (orange) and the other 2 rows for the other slave node. A border was added for each subset (orange and blue) the border width is half of the row height. So, in Fig. 2 we will send the first slave node 2 orange rows and 0.5 blue row and the other slave node 2 blue rows and 0.5 orange row.

Border-based Gird Partition (BGP): The BGP is based on the old GBP in Bai *et al*. (2016). In our BGP algorithm, we continue from the step that determines the number of grids. Then we divide the dataset into isometric grids. Each grid has a border in its dimension *i*. We propose a border. This border *b* is for each dimension *i* in the dataset. The size of the border is a percentage of the width of the grid in the dimension *i* ($i \in [1,d]$). After determining the border width, each grid will now have a new virtual border in dimension *i* which is its original border plus the proposed border size. Each grid now will be allocated to a slave node. As shown in Fig. 2 if the boarder is 25% of grid width. So, when we divide the 4 rows into two slave nodes there must be 2 grids. Each grid will be 2 row (gird in orange and grid in blue). The border width is 25% of the grid so each grid will take 25% of the other grid i.e., 0.5 row. grid 1 will become 2 orange rows and 0.5 blue row and grid 2 will be 2 blue rows and 0.5 orange row as shown in algorithm 1. In algorithm 1 we first calculate the new safe border for each grid, then we add all the tuples between the original border and the new border to the grid (lines 1-8). After that we sort the girds in *g* according to the number of tuples in *g* in descending order, taking in consideration that the tuples in *g* are not only the tuples below original border but also the tuples between the new border and the original border (line 9). Then we go through all the sorted girds and allocate each grid g randomly to all the slave nodes till all the slave nodes are allocated with grids. (lines 10-13).

For the remaining grids *g*, we calculate the average number of tuples per slave node and initialize a set *N'* that contains all the slave nodes with a number of tuples less than or equal (lines 14-15). After that, we select the slave node with the largest number of grids that are adjacent to that grid g from N' and allocate g to this slave node (lines 16-17). We repeat this process (line 13-19) till all the girds are allocated to slave nodes.

After finish allocating the tuples in the grids to the slave nodes using algorithm 1, Each slave node starts to process its tuples and calculate the LOF for each tuple. After each node finish calculating the LOF for the tuples it sends the result back to the master node.

As shown in Fig. 3 we have a dataset with *n* tuples and two slaves. The BGP algorithm will partition the dataset into 4 isometric grids. Each slave node will be allocated a number of grids for this example *g1* will be allocated to a slave node and *g2, g3, g4* will be allocated to the other slave node according to the algorithm 1. Given the border percentage, for example, we have two percentage one in the red color and the other in the blue color. We have three choices one to process the tuples without using any border. The second one is to apply the red border and the last one is to apply the blue border.

Algorithm 1: Border based Grid partition
input: Grid set *G*, dataset *N*, Boarder percentage *b*
output: Grids set allocated to slave nodes
1 for each gird *g* in *G* do
2 for each dimension *i* in *d* do

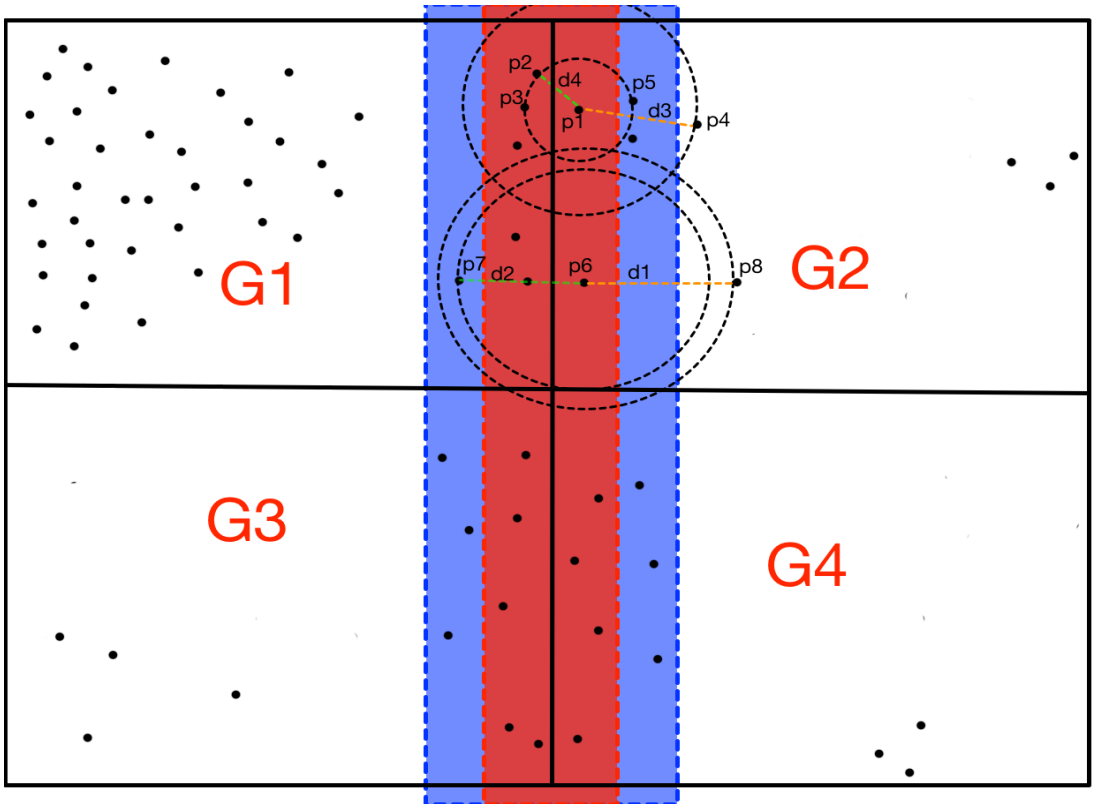

Fig. 3: BGP Example |N | = 2, K = 3

3 bw ← the border width for dimension i
4 calculate new border for g in *i*
5 update *g* new border in *i*
6 end
7 add tuples between the original grid border and the new border to *g*
8 end
9 sort grids in *G* according to the number of tuples (tuples below safe border) in *g* in descending order
10 for each grid g in G do
11 if there exist slave node with no grid then
12 randomly choose a slave node with no grids and allocate g to it
13 else
14 ε ← the average number of tuples per salve node
15 initialize a slave node set *N'* that contain all the slave nodes that has number of tuples less than or equal ε
16 *n* ← select the slave node with the largest number of grids that are adjacent to *g*
17 allocate g to n
18 end
19 end

First, if we didn't apply any border, then when we calculate the $dis_k(p1)$ it will be *d3*. As it is the distance that can include at least 3 tuples (k = 3). If we apply our border to be the red border then when we calculate the $dis_k(p1)$ it will be *d4*. As it is the distance that includes at least 3 tuples (k = 3). The distance changed after we apply our border to be the red border as points *p2, p3* were added to the neighbors of *p1* and they are near to *p1* than *p4*. So that the *d2* is smaller than *d3* as we added extra tuples *(p2,p3)* from the red border. On the other hand, for the point *p6* if we apply the red border the $dis_k(p6)$ will be *d1* but if we increase our border and apply the blue border the $dis_k(p6)$ will be *d2* as *p7* is now in the neighbors of *p6* and it is near to *p6* than *p8.*

Balanced Border-Based Gird Partition algorithm (BBGP): *In* the BGP we notice that the portioning of the dataset is done using the girds concept that the author in Bai *et al*. (2016) proposed. One characteristic of this grids is that they are isometric which mean that all the girds must be with the same dimensions. In fact, this characteristic may lead to unbalanced subsets in the slave nodes. For example, if we have four grids and 2 slave nodes and the dataset is distributed to be most of it in grid *g1* and the other three grids are almost empty. Then when we allocated our grids to the two slave nodes one slave node will be allocated with *g1* and the other slave node will be allocated with the other three empty grids. If we calculate the workload in the two slaves we will found that the two slaves are not balanced and one of them is underutilized which is against the characteristics of the parallel computing. As shown in Fig. 3 we notice that most of the points are in grid *g1*. And the other three

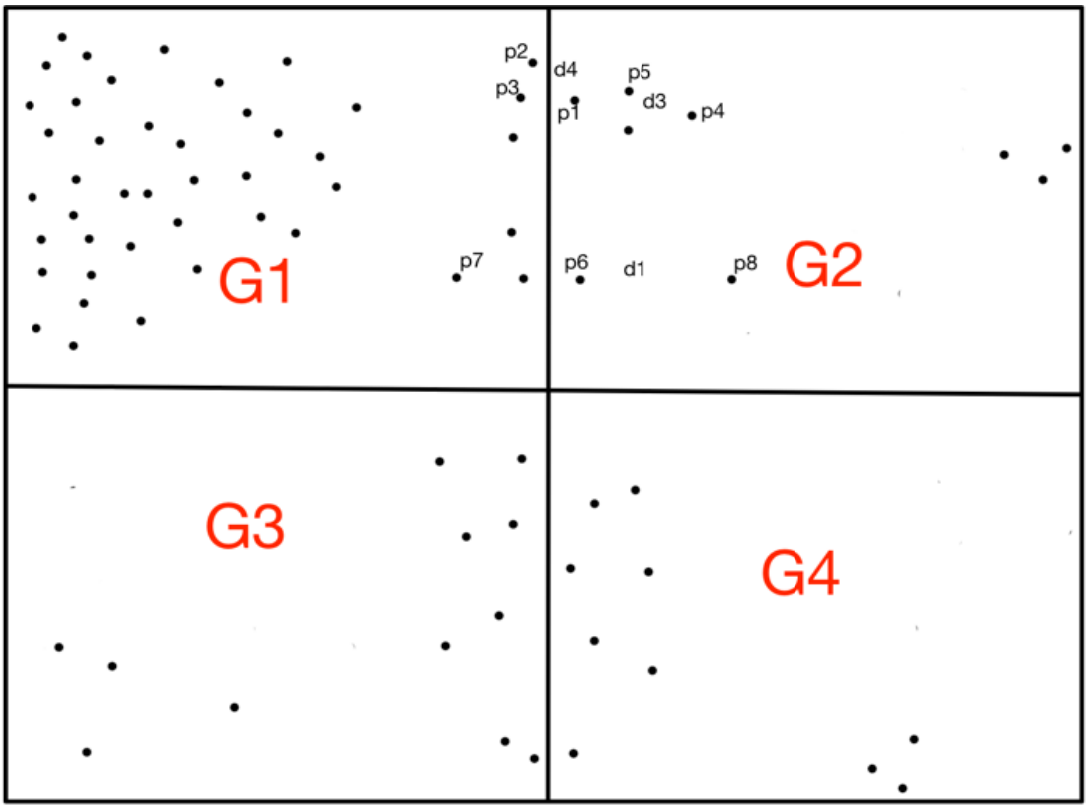

Fig. 4: Non-balanced grids old GBP

grids are almost empty. Grid *g1* will be allocated to the slave node and the other three grids will be allocated to the other slave node. If we checked the load balancing between them we will find the number of tuples allocated to slave 1 to the other slave is 100:20.

Algorithm 2: Balanced border-based Gird Partition algorithm (BBGP)
input: number of grids in each dimension NG, dataset N, Border percentage b
output: Grids set allocated to slave nodes
1 for each dimension i in NG do
2 t1, t2 <- the smallest and the largest tuple in dimension i
3 pg <- the points ng in dimension i that divides the space between t1, t2 such that all the spaces have the same number of tuples, the last space may have less tuples than the other spaces
4 g <-create a grid in dimension i with pg
5 add g to G
6 End
7 call Algorithm 1 with G, N, b

To solve this problem, we proposed the Balanced boarder-based Gird Partition algorithm (BBGP) shown in algorithm 2. In BBGP we first loop through all the dimensions *i* in the number of grids in each dimension, for example, we have 2 grids in *x* dimension and 2 grids in dimension *y* (lines 1-6). In each dimension *i* we select the two tuples *t1* and *t2* that has the smallest and most value in this dimension (line 2). Now we need to divide the space between *t1* and *t2* into *n* spaces that are equal to the number of grids in this dimension. For example, divide this space into two spaces. These spaces must have the same number of tuples as possible (line 3). We create a grid in the dimension *i* with the spaces divided in line 3 then we add this grid to the grid set *G* (line 4-5). At the end we call our algorithm 1 and input to it the grid set *G*, data set *N* and the border percentage *b*. after we apply algorithm 2 it will result into dividing the dataset into a number of grids, these grids have the same number of tuples. Then when allocating these grids to the slave nodes each slave node will have the same number of tuples to process which will lead to balancing the load between all the slave nodes. As shown in Fig. 4 this is the old GBP where the grids are not balanced but in Fig. 5 the grids are balanced.

RESULTS AND DISCUSSION

This section presents the experimental results of the boarder-based Gird Partition (BGP) and Balanced boarder-based Gird Partition algorithm (BBGP). In addition a comparison between the proposed methods and the GBP+DLC algorithm in Bai *et al*. (2016) are

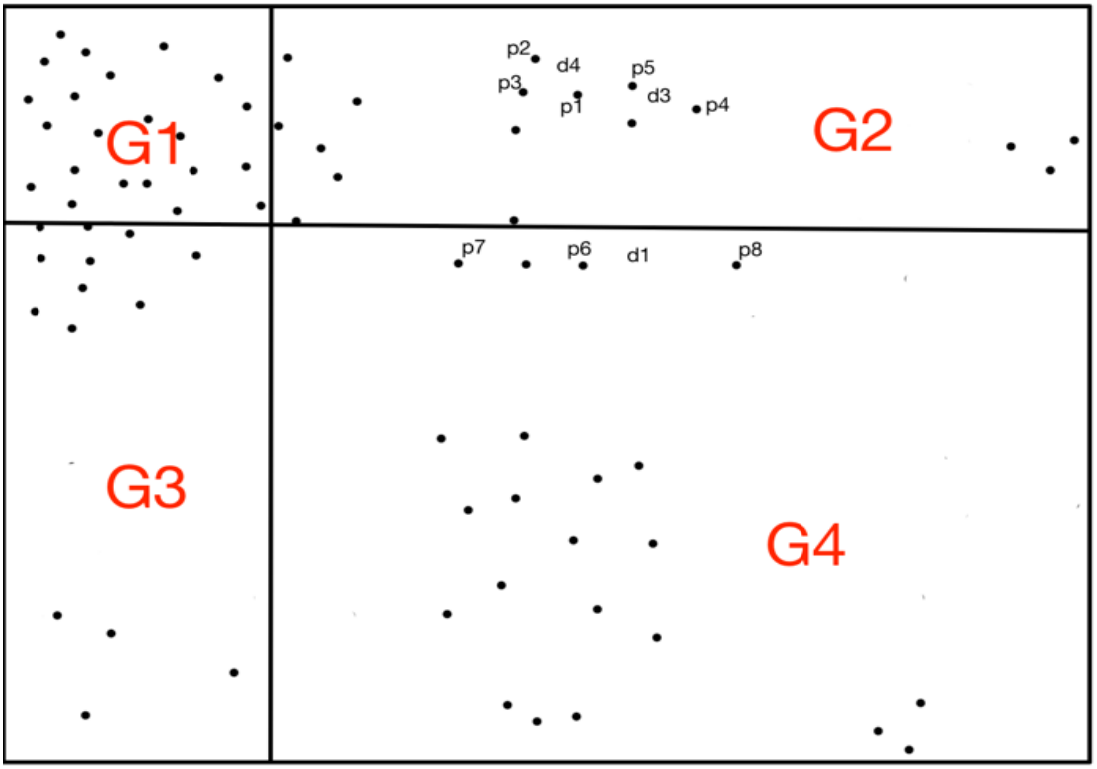

Fig. 5: Balanced grids new BBGP

presented. All the algorithms were implemented in JAVA programming language and we use kryonet (Esoteric, n.d., 2018) a java library that is used for TCP and UDP client/server network communication. All the algorithms were deployed in a cluster that consists of one master node and two slave nodes. The master node has a 2.5 GHz Intel Core i5 CPU, 8G memory DDR3 and 250G SSD hard disk. The other two slave nodes one has 2.5 GHz Intel Core i7 CPU, 4G memory and 1T hard disk and the other slave has 2.5 GHz Intel Core i5 CPU, 8G memory and 1T hard disk.

To evaluate the proposed algorithms three criterias were considered:

- Time taken to calculate the LOF
- Size of data that was transmitted over the network
- Accuracy in calculating the right LOF for each tuple

The calculation of LOF was done based on Breunig *et al*. (2000). All the results were demonstrated using k = 5. Six boarder percentages of grid width were demonstrated to evaluate the proposed algorithms (0.001, 0.005, 0.01, 0.1, 0.2, 0.25).

Two real datasets Shuttle and Covtype (each database is 10,000 tuple) were used to evaluate the performance for all the algorithms. These datasets are obtained from the Machine Learning database repository at UCI (http://archive.ics.uci.edu/ml/) (Dheeru and Karra Taniskidou, 2017).

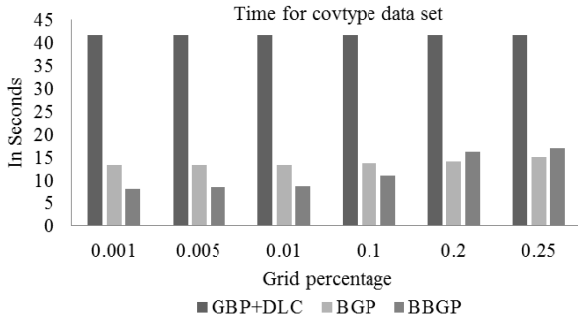

Fig. 6: Processing time for GBP+DLC, BGP and BBGP for covtype data set

Time taken to calculate the LOF: Figure 6 and 7 illustrates that for covtype dataset and shuttle dataset, BGP algorithm takes much smaller time than that of GBP+DLC algorithm in Bai *et al*. (2016). In addition, BBGP algorithm takes a much smaller time than BGP algorithm and GBP+DLC algorithm.

The GBP+DLC algorithm takes time in tuples transmission for the cross-grid tuples over the network but in BGP there is no tuples transmission over the network. That's why the BGP takes less time than the GBP+DLC algorithm.

The BGP and GBP+DLC algorithm take more time than the BBGP algorithm as when partitioning the tuples across the girds (4 grids as mentioned in Fig. 4) most of the tuples about 70% of them were distributed

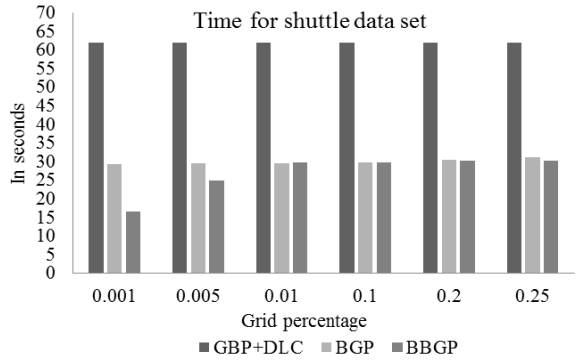

Fig. 7: Processing time for GBP+DLC, BGP and BBGP for shuttle data set

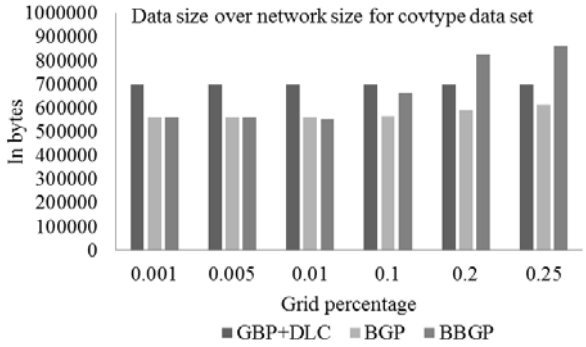

Fig. 8: Size of data over the network for GBP+DLC, BGP and BBGP for CovType data set

in one grid which is processed by only one slave. While slave A is processing the 70% of tuples, the other slave B is processing only 30% of them. Slave B after finishing processing the 30% of tuples it waits idle till slave A finish it's processing which takes a lot of time and don't consume all the resources. On the other hand, the BBGP algorithm takes less time as when portioning the tuples each one of the four grids has almost the same number of tuples. So, every slave has almost the same number of tuples which utilize all the resources.

For the time and boarder percentage we notice that when we increase the border the time for processing is increased a little (or stable in comparing with the total number of tuples to be processed) as there are extra redundant tuples that will be processed.

Size of data that was transmitted over the network: Figure 8 illustrates that, the data size transmitted over the network for the covtype dataset for the GBP+DLC algorithm is more than the size of the data for the BGP and BBGP algorithms. But when we reach border percentage of 0.2 the data size of the BBGP algorithm is more than the BGP and GBP+DLC algorithm this is because that number of redundant tuples increases when increasing the border percentage.

For the shuttle dataset the data size transmitted over the network is illustrated in Fig. 9. The size of the

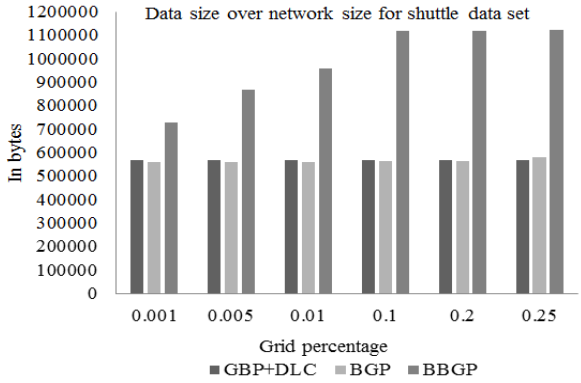

Fig. 9: Size of data over the network for GBP+DLC, BGP and BBGP for shuttle data set

data transmitted over the network for BBGP algorithm is always much bigger than the other two algorithms. That's because of the distribution of the tuples around the boarders is high in compared to covType dataset.

We notice that the data size over the network for our BBGP algorithm is larger than the data for GBP+DLC. But the time for processing the data for the BBGP is less than the time for the GBP+DLC algorithm. This is due to that the GBP+DLC algorithm takes time to calculate the k-distance neighbors and sends a request to the slaves which has these neighbors. Then the slaves send these tuples. After that, it takes time to calculate the LOF for these tuples. This request and response take time which is more than the processing time for the LOF for each tuple. That's why the data size over the network is larger for the BBGP algorithm but the time is smaller than the GBP+DLC algorithm.

Accuracy in calculating the right LOF for each tuple: In evaluating the accuracy of the algorithms, we calculated the LOF for the tuples in both datasets covtype and shuttle. The GBP+DLC algorithm calculated the LOF for all the tuples correctly as any tuple that is needed in the slave it will be sent to it from the other slave so its accuracy is 100%.

For the BGP and BBGP algorithm, each slave node sends a LOF value for each tuple it has. The tuples replicated in both slaves have two values one from each slave node. The master node has the choice to select the LOF with the ssmallest value, biggest or average value between the two slaves. For each tuples, we subtracts the LOF of our algorithm and that of the original LOF and found that for each border of the six borders (0.001, 0.005, 0.01, 0.1, 0.2, 0.25) the master can selects LOF with the minimum, maximum or average value.

BGP algorithm: For the covtype dataset, if the master selects the LOF with the minimum value, about 90.5-91.5% of the tuples are equal to zero and about 98.5-98.7% of the tuples are less than or equal to 0.1. If the

Table 1: Accuracy of the algorithms for the two datasets

		BGP					
		Min.		Max.		Avg	
Data set	Border	= 0	<0.1	= 0	<0.1	= 0	<0.1
Covtype	0.001	91.52	98.71	91.52	98.71	91.52	98.71
	0.005	91.52	98.71	91.52	98.71	91.52	98.71
	0.01	91.52	98.71	91.52	98.71	91.52	98.71
	0.1	91.36	98.69	91.33	98.62	91.17	98.71
	0.2	90.56	98.6	90.91	98.53	89.95	98.58
	0.25	90.5	98.58	90.74	98.44	89.72	98.52
Shuttle	0.001	98.12	99.88	98.12	99.88	98.12	99.88
	0.005	98.12	99.88	98.12	99.88	98.12	99.88
	0.01	98.12	99.88	98.12	99.88	98.12	99.88
	0.1	98.12	99.88	98.13	99.89	98.12	99.88
	0.2	98.12	99.88	98.13	99.89	98.12	99.88
	0.25	98.12	99.88	98.13	99.89	98.12	99.88
		BBGP					
		Min		Max		Avg	
Data set	Border	= 0	<0.1	= 0	<0.1	= 0	<0.1
Covtype	0.001	88.25	98.48	88.25	98.47	88.25	98.47
	0.005	88.26	98.49	88.26	98.47	88.26	98.47
	0.01	88.25	98.49	88.25	98.46	86	98.31
	0.1	87.1	98.41	87.14	98.13	86	98.31
	0.2	87.01	98.45	87.03	98.16	85.79	98.28
	0.25	86.98	98.36	87.29	98.27	86.02	98.32
Shuttle	0.001	76.81	92.39	76.4	92.14	75.43	92.24
	0.005	77.36	92.53	77.16	92.41	76.65	92.42
	0.01	77.24	92.54	77.12	92.44	76.56	92.47
	0.1	77.42	92.55	77.32	92.52	76.92	92.53
	0.2	77.43	92.55	77.31	92.52	77.05	92.53
	0.25	77.49	92.55	77.34	92.52	77.05	92.53

Min.: Minimum; Max.: Maximum

master selects the LOF with the maximum value, about 90.7-91.5% of the tuples are equal to zero and about 98.4-98.7% of the tuples are less than or equal to 0.1. If the master selects the LOF with the average value, about 89.7-91.5% of the tuples are equal to zero and about 98.5-98.7% of the tuples are less than or equal to 0.1.

For the shuttle dataset, if the master selects the LOF with the minimum value, about 98.12% of the tuples are equal to zero and about 99.88% of the tuples are less than or equal to 0.1. If the master selects the LOF with the maximum value, about 98.12% of the tuples are equal to zero and about 99.88% of the tuples are less than or equal to 0.1. If the master selects the LOF with the average value, about 98.12% of the tuples are equal to zero and about 99.88% of the tuples are less than or equal to 0.1 as shown in Table 1.

BBGP algorithm: For the covtype dataset, if the master selects the LOF with the minimum value, about 86.9-88.25% of the tuples are equal to zero and about 98.3-98.5% of the tuples are less than or equal to 0.1. If the master selects the LOF with the maximum value, about 87-88.25% of the tuples are equal to zero and about 98.1-98.5% of the tuples are less than or equal to 0.1. If the master selects the LOF with the average value, about 85.7-88.25% of the tuples are equal to zero and about 98.2-98.4% of the tuples are less than or equal to 0.1

For the shuttle dataset if the master selects the LOF with the minimum value, about 76.8-77.4% of the tuples are equal to zero and about 92.5% of the tuples are less than or equal to 0.1. If the master selects the LOF with the maximum value, about 76.4-77.3% of the tuples are equal to zero and about 92.1-92.5% of the tuples are less than or equal to 0.1. If the master selects the LOF with the average value, about 75.4-77% of the tuples are equal to zero and about 92.2-92.5% of the tuples are less than or equal to 0.1 as shown in Table 1.

This can be summarized in Fig. 10 and 11 for both the datasets and the two algorithms (BGP and BBGP) using the two comparison techniques mentioned above (zero and below 0.1).

CONCLUSION

This study focuses on the problem of calculating the outliers for big data sets in a distributed environment processed. We discussed the existing algorithms that calculate the outliers from the dataset and mentioned the LOF to represent the degree of the outlierness for each tuple in the data set. We also discussed the existing algorithms for calculating the LOF for large-scale data sets in a distributed

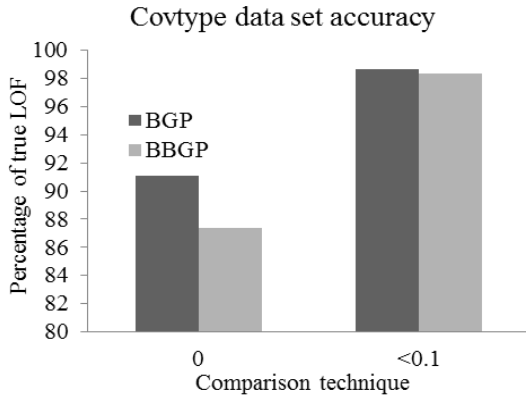

Fig. 10: Percentage of true positive outliers detected for BGP and BBGP algorithm for CovType data set

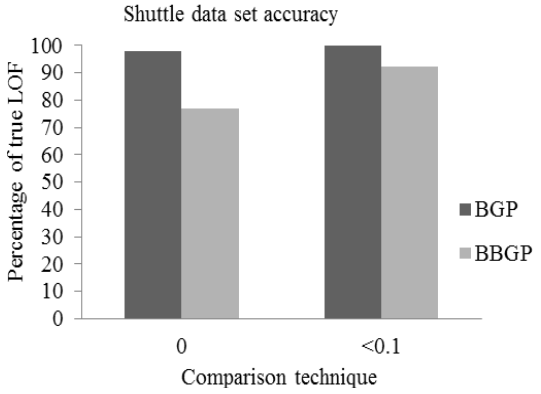

Fig. 11: Percentage of true positive outliers detected for BGP and BBGP algorithm for shuttle data set

environment and mentioned its weakness. Then we propose two algorithms boarder-based Gird Partition (BGP) and Balanced boarder-based Gird Partition algorithm (BBGP). BGP uses a safe border to copy tuples between grids to be processed with each grid and discussed the impact of border size change on the results. BBGP algorithm solves the problem of unbalanced grids distributed between processing nodes. Finally, we evaluate our algorithms by time, data size over network and accuracy of the output over two real data sets shuttle and covtype. The evaluation results show that our two algorithms BGP and BBGP take less time than GBP+DLC algorithm. And the accuracy of our algorithms in calculating LOF is about 90-92% for BGP and 87-88.3% for BBGP.

REFERENCES

Aggarwal, C.C. and P.S. Yu, 2001. Outlier detection for high dimensional data. Proceeding of the 2001 ACM SIGMOD International Conference on Management of Data, pp: 37-46.

Aggarwal, C.C. and P.S. Yu, 2008. Outlier detection with uncertain data. Proceeding of the 2008 SIAM International Conference on Data Mining, pp: 483-493.

Aggarwal, C.C., J. Han, J. Wang and P.S. Yu, 2003. A framework for clustering evolving data streams. Proceeding of the 29th International Conference on Very Large Data Bases (VLDB '03), 29: 81-92.

Bai, M., X. Wang, J. Xin and G. Wang, 2016. An efficient algorithm for distributed density-based outlier detection on big data. Neurocomputing, 181: 19-28.

Breunig, M.M., H.P. Kriegel, R.T. Ng and J. Sander, 2000. LOF: Identifying density-based local outliers. Proceeding of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00), pp: 93-104.

Cao, F., M. Ester, W. Qian and A. Zhou, 2006. Density-based clustering over an evolving data stream with noise. Proceeding of the 6th SIAM International Conference on Data Mining, pp: 328-339.

Dheeru, D. and E. Karra Taniskidou, 2017. UCI Machine Learning Repository.

Esoteric, n.d. 2018. Kryonet. [Online] Retrieved form: https://github.com/EsotericSoftware/kryonet (Accessed on: January 1, 2018)

Guha, S., A. Meyerson, N. Mishra, R. Motwani and L. O'Callaghan, 2003. Clustering data streams: Theory and practice. IEEE T. Knowl. Data En., 15: 515-528.

Hawkins, D.M., 1980. Identification of Outliers. Chapman and Hall, London, Vol. 11.

Jin, W., A.K.H. Tung, J. Han and W. Wang, 2006. Ranking outliers using symmetric neighborhood relationship. In: Ng, W.K., M. Kitsuregawa, J. Li and K. Chang, (Eds.): Advances in Knowledge Discovery and Data Mining. PAKDD, 2006. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 3918: 577-593.

Knox, E.M. and R.T. Ng, 1998. Algorithms for mining distance-based outliers in large datasets. Proceeding of the 24th International Conference on Very Large Data Bases (VLDB '98), pp: 392-403.

Kontaki, M., A. Gounaris, A.N. Papadopoulos, K. Tsichlas and Y. Manolopoulos, 2011. Continuous monitoring of distance-based outliers over data streams. Proceeding of the IEEE 27th International Conference on Data Engineering, pp: 135-146.

Lozano, E. and E. Acufia, 2005. Parallel algorithms for distance-based and density-based outliers. Proceeding of the 5th IEEE International Conference on Data Mining (ICDM'05), pp: 4.

Rajasegarar, S., C. Leckie and M. Palaniswami, 2008. Anomaly detection in wireless sensor networks. IEEE Wirel. Commun., 15(4): 34-40.

Ramaswamy, S., R. Rastogi and K. Shim, 2000. Efficient algorithms for mining outliers from large data sets. Proceeding of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00), pp: 427-438.

Tang, J., Z. Chen, A.W.C. Fu and D.W. Cheung, 2002. Enhancing effectiveness of outlier detections for low density patterns. In: Chen, M.S., P.S. Yu and B. Liu (Eds.), Advances in Knowledge Discovery and Data Mining. PAKDD, 2002. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2336: 535-548.