# Massively scalable density based clustering (DBSCAN) on the HPCC systems big data platform

**Yatish HR[1], Shubham Milind Phal[2], Tanmay Sanjay Hukkeri[3], Lili Xu[4], Shobha G[5], Jyoti Shetty[6], Arjuna Chala[7]**
[1,2,3,5,6] RV College of Engineering, Bangalore, Karnataka, India
[4,7] HPCC Systems LexisNexis Risk Solutions, USA

**Article Info**

***Article history:***

Received Aug 1, 2020
Revised Jan 9, 2021
Accepted Feb 7, 2021

***Keywords:***

Big data
Data mining
Density based clustering
Distributed computing
HPCC systems
Machine learning

**ABSTRACT**

Dealing with large samples of unlabeled data is a key challenge in today's world, especially in applications such as traffic pattern analysis and disaster management. DBSCAN, or density based spatial clustering of applications with noise, is a well-known density-based clustering algorithm. Its key strengths lie in its capability to detect outliers and handle arbitrarily shaped clusters. However, the algorithm, being fundamentally sequential in nature, proves expensive and time consuming when operated on extensively large data chunks. This paper thus presents a novel implementation of a parallel and distributed DBSCAN algorithm on the HPCC systems platform. The algorithm seeks to fully parallelize the algorithm implementation by making use of HPCC systems optimal distributed architecture and performing a tree-based union to merge local clusters. The proposed approach* was tested both on synthetic as well as standard datasets (MFCCs Data Set) and found to be completely accurate. Additionally, when compared against a single node setup, a significant decrease in computation time was observed with no impact to accuracy. The parallelized algorithm performed eight times better for higher number of data points and takes exponentially lesser time as the number of data points increases.

***Corresponding Author:***

Yatish H R
Department of Computer Science and Engineering
R.V College of Engineering
Mysore Rd, RV Vidyaniketan, Post, Bengaluru, Karnataka 560059, India
Email: yatishhr.cs16@rvce.edu.in

## 1. INTRODUCTION

Clustering, or the grouping of data into clusters, is one of the most fundamental techniques in dealing with large chunks of unlabeled data. It involves grouping the data into meaningful subclasses, such that the inter-class distances are maximized, and the intra-class distances are minimized. There are four major domains of clustering algorithms, namely: hierarchy-based, partitioning-based, density-based and grid-based [1] DBSCAN, or density based spatial clustering of applications with noise, falls under the category of density-based clustering algorithm. This algorithm is based on the premise that for every data point in the cluster, its neighborhood within a given radius(eps) has to contain a minimum number of points(minpts). Thus, for a given threshold, the density of the neighborhood for every point should exceed this threshold. When dealing with large samples of data, performing this clustering in a sequential manner is time consuming and often expensive. Thus, the proposed paper presents a distributed parallel DBSCAN algorithm, in order to overcome these performance bottlenecks without any impact to accuracy. The proposed algorithm is implemented on the HPCC systems [2].

HPCC systems is an open-source, lightweight and powerful big data management plat- form, which serves as an end-to-end Data Lake management solution. Its key advantages arise from its scalability, performance and usability. The platform serves as an alternative to existing big-data platforms such as apache hadoop, apache spark and data torrent RTS [3]. The platform is supported by its underlying programming language ECL, which is implicitly parallel and declarative in nature, and provides several constructs to simplify parallel compute operations. The general HPCC systems architecture is shown in Figure 1.

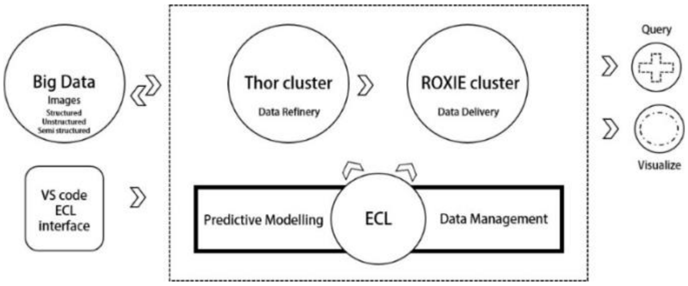


Figure 1. HPCC systems architecture

HPCC systems is further supported by a pair of powerful data engines: THOR and ROXIE. THOR serves as the data refinery engine, and gives the user control over data transformations. It also facilitates optimal operational capacity on mixed schema data. ROXIE serves as the search engine that facilitates high-speed real-time queries through interfaces such as REST, SOAP and XML. It is responsible in greatly reducing the latency associated with querying. The remainder of the paper is organized as follows. Section 2 analyses the existing state of the art for distributed parallel clustering algorithms. Section 3 details both the traditional DBSCAN algorithm as well as the novel parallel DBSCAN algorithm. Section 4 discusses the experiments carried out and the results obtained. Finally, section 5 provides a conclusion to the work as well as propositions for future work.

## 2. LITERATURE REVIEW

A study of the state-of-the-art techniques reveals certain pre-existing methods that seek to improve the performance of the DBSCAN algorithm. The authors in [4-5] showed that the DBSCAN algorithm can be solved in O(nlogn) for Euclidean 2D spaces. However, this method cannot be directly applied on massive datasets distributed across multiple nodes for higher dimensions. There were efforts presented by authors in [6] which presented an approximate DBSCAN in O(nlog(n)) time. The authors [7] also achieved the same but in O(n) time. However, none of the approaches talk about paralellization required in big data applications.

Some of the other techniques make use of GPU' s and various concepts of parallelism to achieve the requisite level of performance. Other proposals to improve the performance include the use of R*-Tree structure [8], KDTREE algorithm [9], special data partition algorithm or disjoint set data structure [10-13]. The implementation of these techniques however, is performed on a single physical system. Due to limitations on the number of cores, and memory of a single system the amount of scalability and performance boost that can be achieved using them is limited.

When working with large real-world datasets the use of a distributed system for clustering is more appropriate [14-17]. A popular strategy for implementing the distributed DBSCAN algorithm is the usage of the disjoint set data structure. This approach was proposed in the PDSDBSCAN algorithm by Patwary *et.al.* [18-19]. The algorithm makes use of a bottom- up structure to construct the clusters as a collection of hierarchical trees. In this approach the clustering is performed in 2 phases. In the first phase local clustering is performed via the regular DBSCAN algorithm across each node. In the second phase the global union of clusters across nodes is performed using the union-find operation. PDSDBSCAN was shown to significantly outperform the previous approaches used to parallelize DBSCAN. Speedups up to a factor of 25.97 was achieved when using 40 cores on datasets containing several hundred million high-dimensional points. PDSDBSCAN is implemented using both OpenMP and MPI.

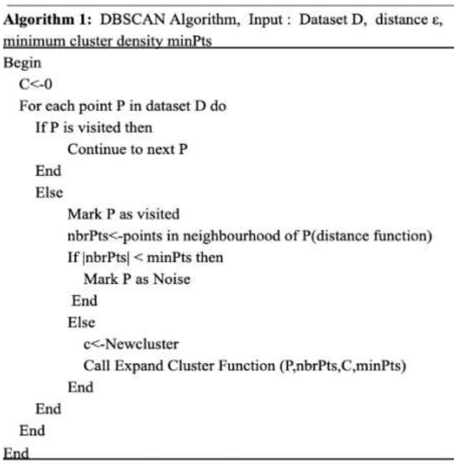
An improvement to the above algorithm was proposed by Hu *et al.* [20] wherein the authors made use of the distributed file system of the Kunpeng system. In this algorithm data was distributed across different workers and message passing interface (MPI) communication pattern was used to communicate the data between the workers. Further the large overhead associated with MPI was overcome using a novel parameter server frame- work. PS-DBSCAN was show to outperform the MPI-based PDSDBSCAN-D with a 2-10 times speedup on communication efficiency in both real-world and synthetic datasets, and the speedup was found to increase with the increase in the number of processor cores and the dataset scale. In RP-DBSCAN [21] the authors made use of pseudo random partitioning together with a two-level cell dictionary. RP-DBSCAN was implemented using 48 cores (12 Azure Ma- chines) on an Apache Spark system. RP-DBSCAN was found to achieve an almost perfect load balance among data splits in local clustering and did not duplicate points among them. Hence RP-DBSCAN dramatically outperformed the state-of-the-art parallel DBSCAN algorithms by up to 180 times. Furthermore, only RP-DBSCAN was able to handle large data sets (362 GB) whereas the other algorithms could not.

Although the usage of disjoint set data structure for DBSCAN is popular in distributed systems several other methods such as the convex-hull method are used as well [22]. In the convex- hull method a convex hull enclosing the points within a cluster is created locally across nodes. In the merge operation, the convex hulls of the local clusters are merged based on the amount of overlap. However, this algorithm is not capable of handling concave shapes and hence does not correctly capture the essence of the underlying points. In the algorithm [23] the authors propose a grid based disjoint set algorithm for solving the problem efficiently. However due to the fact that distribution of data by Thor among nodes is random, it is not possible to apply the algorithm directly. These existing algorithms are not a straightforward fit into the HPCC Big Data Platform and require modifications. Hence there is a need for a better algorithm which can exploit the components of HPCC such as Thor and Roxie in order to gain massive performance improvements during parallel execution of the DBSCAN algorithm.

## 3. METHODOLOGY

### 3.1. DBSCAN algorithm

The DBSCAN algorithm [24] is non-parametric spatial clustering algorithm. The main concept of the clustering algorithm is that for each cluster the number of points within eps distance is greater than certain minimum points i.e.., threshold density. The pseudo code for DBSCAN algorithm is elucidated in Algorithm 1. The algorithm starts with a point p ∈ D and retrieves the neighboring points in the eps-neighborhood of itself. If the retrieved neighborhood contains at least minpts points, then a new cluster, C is added. The algorithm then finds all points in X, that are reachable from x (neighboring point) and adds them to the cluster C. If the eps-neighborhood of x has less than minpts, then x is marked as noise. The pseudo code can be summarized in Algorithm 1.



```
Algorithm 1: DBSCAN Algorithm, Input : Dataset D, distance ε,
minimum cluster density minPts
Begin
  C<-0
  For each point P in dataset D do
    If P is visited then
        Continue to next P
    End
    Else
        Mark P as visited
        nbrPts<-points in neighbourhood of P(distance function)
        If |nbrPts| < minPts then
          Mark P as Noise
        End
        Else
          c<-Newcluster
          Call Expand Cluster Function (P,nbrPts,C,minPts)
        End
    End
  End
End
```

Algorithm 1. DBSCAN algorithm

The time complexity of Algorithm 1 is O (n^2), where n is the number of points in X. But if R* trees or K-D trees are used to obtain the nearest neighbors the time complexity is reduced to O (n*log n). The

DBSCAN algorithm is a highly sequential algorithm. This makes it computationally inefficient task when applied to large amounts of data, especially on big data platforms. Hence there is a need to parallelize the algorithm for achieving better efficiency in such big data systems. The following section gives the overall procedure for the same.

### 3.2. Parallel DBSCAN algorithm

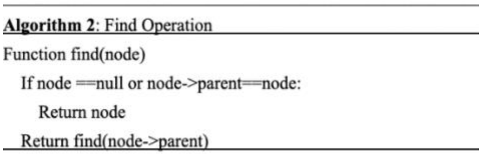
The proposed methodology provides a scalable solution to the DBSCAN clustering on HPCC big data platform. This is achieved by decomposing the clustering algorithm into three stages namely-spraying of data, local clustering phase and global merging of results. Each of these stages are explained in sections:

#### 3.2.1. Spraying of data

In this stage the data points are assigned global unique ids and distributed across different nodes in a cluster by Thor engine. The distribution by the Thor engine ensures that the data points are distributed evenly across all the nodes. Each of the local nodes then sort the data points by their unique ids and send the data to local clustering stage. These unique ids do not change till the end of all the stages and they identify each data point uniquely. One more important point to note is that the algorithm does not depend on the nature of distribution of data across the nodes.

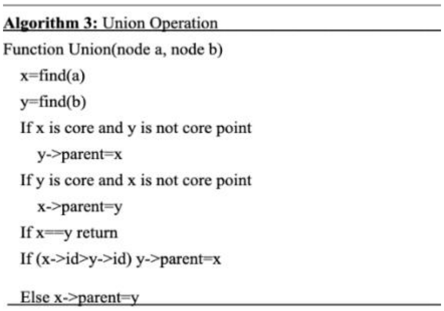
#### 3.2.2. Local clustering

In this phase each node performs DBSCAN clustering operations on the data points that are local to the node in the HPCC cluster. The local DBSCAN algorithm uses disjoint set operations namely-UNION and FIND for performing local DBSCAN clustering. Each set (final cluster) is uniquely represented by highest core point. The FIND operation is used to identify the parent i.e.., highest core point, for each point (node) in the tree. The FIND operation is summarized in Algorithm 2. The union operation merges two trees to form a bigger tree based on rem's algorithm. The union operation is performed on a core point and other core or a non-core point. The union operation is summarized in Algorithm 3.



```
Algorithm 2: Find Operation
Function find(node)
  If node ==null or node->parent==node:
    Return node
  Return find(node->parent)
```

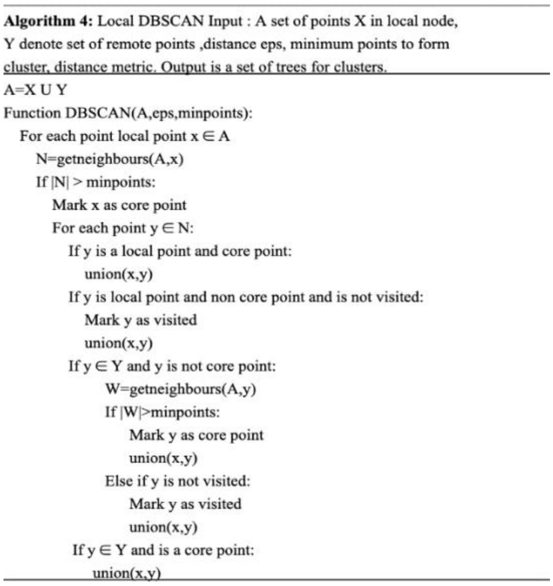
Algorithm 2. Find operation



```
Algorithm 3: Union Operation
Function Union(node a, node b)
  x=find(a)
  y=find(b)
  If x is core and y is not core point
    y->parent=x
  If y is core and x is not core point
    x->parent=y
  If x==y return
  If (x->id>y->id) y->parent=x
  Else x->parent=y
```

Algorithm 3. Union operation

Initially each data point is represented as a node containing the fields and parent id. The parent of each data point is made to point to itself i.e.., p(x)=x and the points are marked as non-core point. Each cluster is uniquely represented by the parent id (highest core point of that cluster). The pseudo code for local DBSCAN clustering is given in Algorithm 4. Let X denote the set of data points that are in the local node and Y denote set of data points that are present on a remote node. The points that are in the eps neighborhood of each $x \in X$ in the set $X \cup Y$ is found. The neighborhood points can be found by using numerous distance metrics like Euclidean, Manhattan, and Chebyshev's distance functions. If the number of points in the neighborhood is greater than the minimum number of points, then it is marked as a core point. Each point y in the neighborhood of x can either be local to that node or in a remote node.



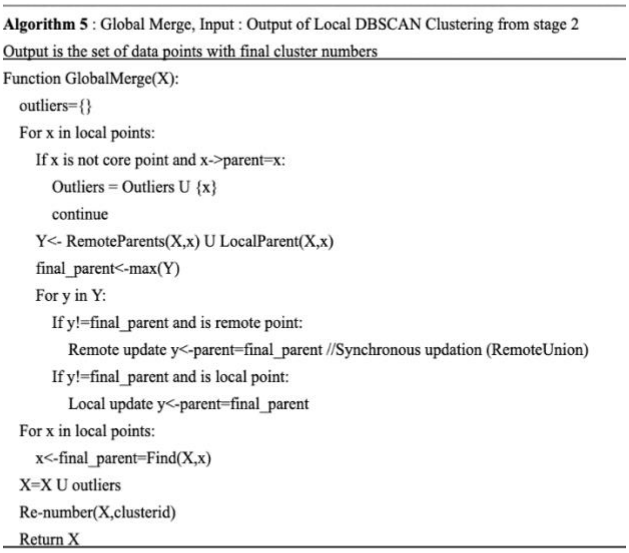
```
Algorithm 4: Local DBSCAN Input : A set of points X in local node,
Y denote set of remote points ,distance eps, minimum points to form
cluster, distance metric. Output is a set of trees for clusters.
A=X U Y
Function DBSCAN(A,eps,minpoints):
  For each point local point x ∈ A
    N=getneighbours(A,x)
    If |N| > minpoints:
      Mark x as core point
      For each point y ∈ N:
        If y is a local point and core point:
          union(x,y)
        If y is local point and non core point and is not visited:
          Mark y as visited
          union(x,y)
        If y ∈ Y and y is not core point:
            W=getneighbours(A,y)
            If |W|>minpoints:
                Mark y as core point
                union(x,y)
            Else if y is not visited:
                Mark y as visited
                union(x,y)
        If y ∈ Y and is a core point:
          union(x,y)
```

Algorithm 4. Local DBSCAN

If y is in local node then there can be two cases: (a) y is core point, (b) y is non-core point i.e.., a border point. If condition (a) is true Union (x, y) is performed. If condition (b) is true then its marked as visited and Union (x, y) is performed. If y is a remote point then y can be either a core point or a non-core point. If the latter condition is true then the neighborhood of y is found. If the neighborhood does not contain minimum points then y is a border point else it is marked as a core point and union (x, y) is performed. If the point y is core then union (x, y) operation is performed and the process is continued for other points. Therefore, at the end of the algorithm it can be noted that the points which are noisy have single trees (singleton trees) with its parent pointing to itself. If the point is a border point then it is assigned to the highest core point of the cluster of its nearest neighbor. If the point is a core point then the Union operation assigns it to parent with highest core point. The time complexity of the local DBSCAN operation is O($n^2$). The above algorithm merges points to its clusters (trees) without any particular ordering of data-points. The output mapping of datapoint to the parent ids is input to the next stage as a global ttream of data points so that each data point is returned one by one from the Thor.

### 3.2.3. Global merge

In this phase, the trees formed in the previous stage are merged to obtain the global clusters. A merge of trees across nodes happens when a point belongs to more than one tree in different nodes. After the trees are merged the final clusters are obtained which are represented by their highest core point across all nodes. This can be achieved by using similar approaches of union and find operations of disjoint set structure as described before. The pseudo code is elucidated in Algorithm 5. Initially the output mapping of stage 2 (only core points and local points that node) is distributed back among the nodes as well as non outlier points inorder to find the ultimate id. Each data point has two attributes - local parent id (local to the node) and global parent id (global/ultimate id). In each node, the remote parents of each data point in remote nodes are found (by having a local lookup table). In cases such as a highest core point being present in another node, the trees are merged (changing the ids) and the union operation results in the highest core point to that cluster (by changing the parent ids). Hence at the end of the algorithm each cluster is represented by highest core point as cluster id. Since the entire computation is done parallelly, an increased speedup is achieved. After this, clusters are renumbered by assigning 0 to the outliers found in stage two and 1 based indexing to the rest of points based on the order of their occurrences in dataset. The final result is returned to ECL Watch for the user as table.



**Algorithm 5** : Global Merge, Input : Output of Local DBSCAN Clustering from stage 2

Output is the set of data points with final cluster numbers

```
Function GlobalMerge(X):
  outliers={}
  For x in local points:
    If x is not core point and x->parent=x:
      Outliers = Outliers U {x}
      continue
    Y<- RemoteParents(X,x) U LocalParent(X,x)
    final_parent<-max(Y)
    For y in Y:
      If y!=final_parent and is remote point:
        Remote update y<-parent=final_parent //Synchronous updation (RemoteUnion)
      If y!=final_parent and is local point:
        Local update y<-parent=final_parent
  For x in local points:
    x<-final_parent=Find(X,x)
  X=X U outliers
  Re-number(X,clusterid)
  Return X
```

Algorithm 5. Global merge

## 4. EXPERIMENTS AND RESULTS

The Parallel DBSCAN algorithm was implemented in Enterprise Control Language on HPCC systems. Each node in the cluster had 6GB of RAM and 128GB of Hard Disk. The processor used was intel Xeon with 2.4GHz clock frequency. Table 1 elucidates the processor specifications. In order to measure the performance of the parallel DBSCAN algorithm a comparison between single node and multi node (consisting of two nodes and three nodes) clusters was made. Each dataset was run with different eps and minpts. A Higher value of minpts increases the noise counts. The results are tabulated as shown in Table 2.

The parallel DBSCAN implementation and serial DBSCAN implementation were tested separately for conventional single node processing versus the use of a two node HPCC cluster. The results obtained were compared with the output of sklearn implementation of DBSCAN to compare the cluster accuracy. The cluster densities in both cases were found to be identical. The test set consisted of 6 test cases including both synthetic datasets and real dataset (frogs' MFCC dataset*[25]). The test sets had up to 20 dimensions. Also, one more point to note is that the data is distributed uniformly among the nodes. The data points in disjoint sets (representing each cluster) may or may not be in the same node. Hence the experiments conducted does not assume any particular distribution of data points among the worker nodes and no further redistribution is required at any stage of algorithm. The outputs indicate that substantial improvements in time complexity of the parallel algorithm than the serial version (single node).

The tabulated results obtained are depicted graphically in Figure 2. We can observe that there is no significant improvement in the execution time for lower number of points for higher number of nodes. It is in fact slightly higher due to the fact that the time required to distribute data points is not compensated well by the time gained due to parallelization. However, it is clear that for larger number of data points as the number of nodes increases, substantial speedup is obtained. The results clearly indicate that the multi node setup outperforms the single node setup in all cases. When the number of points is less, there is no significant improvement of parallel DBSCAN over sequential DBSCAN. As the number of data points increases the parallel algorithm performs way better (4x better for 30000 points compared to two node cluster) than its serial counterpart. The code for the same was merged to HPCC systems repository and can be found on Github.

Table 1. Processor specifications for each node

| Specification | Value |
|---|---|
| Architecture | x86_64 |
| CPU op-mode(s) | 32-bit, 64-bit |
| Byte Order | Little Endian |
| Model Name | Intel Xeon |
| CPU GHz | 2.4 |
| Core (s) | 6 |

Table 2. Results of run time of different datasets with various values of eps and minpts for serial and parallel algorithm

| Size | Epsdistance | Minpts in a cluster | Single node time (in s) | Time on two nodes (in s) | Time on three nodes (in s) |
|---|---|---|---|---|---|
| 4800 | 0.2 | 2 | 16.35 | 14.5 | 15.86 |
| 6000 | 0.3 | 9 | 35.2 | 22.246 | 23.471 |
| 7200* | 0.3 | 10 | 53.5 | 44.426 | 45.63 |
| 9000 | 0.35 | 10 | 112.8 | 50.573 | 53.642 |
| 14300 | 0.4 | 20 | 535.74 | 213.922 | 203.184 |
| 30000 | 0.4 | 20 | 3924.72 | 964.616 | 727.33 |
| 50000 | 0.5 | 30 | 24948.69 | 5124.24 | 3266.462 |

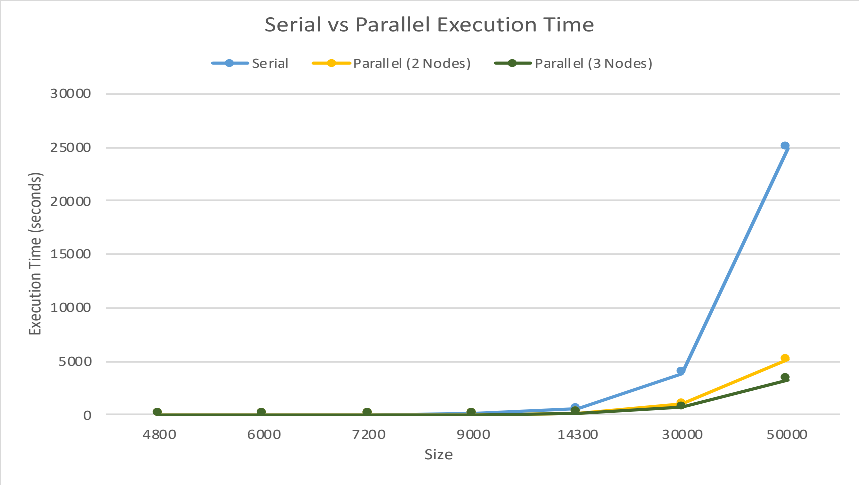


Figure 2. Comparison between serial DBSCAN and parallel DBSCAN on HPCC systems

## 5. CONCLUSION

As the number of big data applications are increasing in the recent years there is a need for big data platforms which enable faster data processing, and HPCC's Platform is a suitable alternative. HPCC platform supports cross platform developments in languages like C++, python, which makes it to develop applications at a faster pace. The thor and roxie components of HPCC platform enables faster data ingestion and data query across multiple nodes which makes it efficient in implementing machine learning algorithms. Also, the platform parallelizes the sequential algorithms across multiple nodes efficiently. This makes it possible for highly sequential and computation intensive algorithms like DBSCAN to be implemented at a scale. The experiments demonstrated highlight the fact that the platform can divide the computation across nodes efficiently. In the future, the existing functionality of the DBSCAN algorithm on HPCC can be used in various data analytics applications especially in areas of geo-spatial clustering's.

## ACKNOWLEDGEMENTS

We would like to thank Lexis-Nexis and R.V. College of Engineering for actively sup- porting this research work. The interfacing documentation and the set of clusters used to test the system were provided by Lexis-Nexis. R.V. College of Engineering provided all the necessary infrastructure support for the project

## REFERENCES

[1] Bano, Saima & Khan, Naeem, "A Survey of Data Clustering Methods," *International Journal of Advanced Science and Technology*. 113, 2018, doi:10.14257/ijast.2018.113.14.

[2] A.M. Middleton and A. Chala (Lexis Nexis Risk Solutions), "HPCC Systems®: Introduction to HPCC (High-Performance Computing Cluster)," May 2011.

[3] Tanmay Sanjay Hukkeri, Shobha G, Shubham Milind Phal, Jyothi Shetty, Yatish H R, and Naweed Mohammed, "Massively Scalable Image Processing on the HPCC Systems Big Data Platform," *In Proceedings of the 3rd International Conference on Software Engineering and Information Management (ICSIM '20)*. Association for Computing Machinery, New York, NY, USA, pp. 26-31, 2020, DOI:https://doi.org/10.1145/3378936.3378978.

[4] Ade Gunawan, "A faster algorithm for DBSCAN," Master's thesis, Eindhoven University of Technology, 2013.

[5] Mark de Berg, Ade Gunawan, and Marcel Roeloffzen, “Faster DB-scan and HDB-scan in Low-Dimensional Euclidean Spaces,” *International Journal of Computational Geometry & Applications*, vol. 29, no. 1, pp. 21-47, 2019, doi: 10.1142/S0218195919400028.

[6] Danny Z. Chen, Michiel Smid, and Bin Xu, “Geometric Algorithms for Density-Based Data Clustering,” *International Journal of Computational Geometry & Applications*, vol. 15, no. 3, pp. 239-260, 2005, doi: 10.1007/3-540-45749-6_28.

[7] Junhao Gan and Yufei Tao, “On the Hardness and Approximation of Euclidean DBSCAN,” *ACM Trans. Database Syst*., vol. 1, no. 1, 2016.

[8] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, “The r*-tree: an efficient and robust access method for points and rectangles,” *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, vol. 19, no. 2, pp. 322-331, 1990, doi: 10.1145/93605.98741.

[9] M. B. Kennel, “KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space,” institute for Nonlinear Science, University of California, 2004.

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” *in Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, vol. 1996. AAAI Press, pp. 226-231, 1996.

[11] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: an efficient data clustering method for very large databases,” *in ACM SIGMOD Record*, vol. 25, no. 2, pp. 103-114, ACM, 1996.

[12] D. Arlia and M. Coppola, “Experiments in parallel clustering with DBSCAN,” *in Euro-Par 2001 Parallel Processing. Springer, LNCS*, pp. 326-331, 2001.

[13] M. Chen, X. Gao, and H. Li, “Parallel DBSCAN with priority r-tree,” *in Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. IEEE*, pp. 508-511, 2010, doi: 10.1109/ICIME.2010.5477926.

[14] Y. Fu, W. Zhao, and H. Ma, “Research on parallel DBSCAN algorithm design based on mapreduce,” *Advanced Materials Research*, vol. 301, pp. 1133-1138, 2011, DOI: 10.4028/www.scientific.net/AMR.301-303.1133.

[15] Markus Götz, Christian Bodenstein, and Morris Riedel, “HPDBSCAN: Highly Parallel DBSCAN,” *In Proc. Workshop on Machine Learning in High-Performance Computing Environments*, no. 2, pp. 1-10, 2015, doi: 10.1145/2834892.2834894.

[16] G. Luo, X. Luo, T. F. Gooch, L. Tian and K. Qin, “A Parallel DBSCAN Algorithm Based on Spark,” *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Atlanta, GA, USA, pp. 548-553, 2016, doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.85.

[17] D. Han, A. Agrawal, W. Liao and A. Choudhary, “A Novel Scalable DBSCAN Algorithm with Spark,” *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, USA, pp. 1393-1402, 2016, doi: 10.1109/IPDPSW.2016.57.

[18] M. M. A. Patwary, D. Palsetia, A. SAgrawal, W. Liao, F. Manne and A. Choudhary, “A new scalable parallel DBSCAN algorithm using the disjoint-set data structure,” *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, 2012, pp. 1-11. doi: 10.1109/SC.2012.9.

[19] M. Patwary, J. Blair, and F. Manne, “Experiments on union-find algorithms for the disjoint-set data structure,” *in Proceedings of the 9th International Symposium on Experimental Algorithms. Springer, LNCS* 6049, pp. 411-423, 2010, doi: 10.1007/978-3-642-13193-6_35.

[20] Hu, Xu & Huang, Jun & Qiu, Minghui & Chen, Cen & Chu, Wei, “PS-DBSCAN: An Efficient Parallel DBSCAN Algorithm Based on Platform of AI (PAI), *arXiv:1711.01034v1 [cs.DB] 3 Nov 2017*, 2017.

[21] Song, Hwanjun & Lee, Jae-Gil, “RP-DBSCAN: A Superfast Parallel DBSCAN Algorithm Based on Random Partitioning,” *SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data,* 2018, DOI: 10.1145/3183713.3196887.

[22] Llort, Germán & Gonzalez, Juan & Servat, Harald & Gamblin, T. & Gimenez, Judit & Labarta, Jesús, “Distributed tree-based implementation of DBSCAN cluster algorithm for on-line performance analysis, *Preprint submitted to Parallel Computing*, 2016.

[23] Yiqiu Wang, Yan Gu, and Julian Shun, “Theoretically-Efficient and Practical Parallel DBSCAN. *In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2555-2571, 2020.

[24] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, “A Density-based Algorithm for Discovering Clusters a Density-basedvAlgorithm for Discovering Clusters in Large Spatial Databases withvNoise. *In International Conference on Knowledge Discovery and DatavMining (KDD)*, pp. 226-231, 1996.

[25] Dua, D. and Graff, C., “UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2019.