*Amany AbdElSamea Saeed*. International Journal of Engineering Research and Applications*
*www.ijera.com*
*ISSN: 2248-9622, Vol. 14, Issue 6, June, 2024, pp: 109-117*

RESEARCH ARTICLE OPEN ACCESS

# DBSCAN Clustering Algorithm for Efficient Container Allocation in Cloud Computing Environment

Amany AbdElSamea Saeed*
**(Computers and Systems Department, Electronics Research Institute, Cairo, Egypt)*

**Abstract**
The deployment, management, and scalability of applications in cloud computing environment have all been revolutionized using containerization by encapsulating the applications and their dependencies into lightweight, portable units called containers. Although containers have many advantages, such as isolation, consistency across environments, and quick deployment, effective resource allocation is still a significant barrier in dynamic and heterogeneous cloud systems. Clustering techniques are crucial for cloud computing environments because they efficiently allocate and regulate resources to satisfy varying demands of workload. Clustering algorithms aggregate related workloads or containers into clusters allowing for more efficient resource utilization and better performance isolation. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a popular clustering algorithm that effectively discovers clusters of arbitrary shapes in spatial data while effectively handling noise. DBSCAN is very versatile and suitable to a wide range of datasets because it does not need a set quantity of clusters, in contrast to other standard clustering techniques. In order to improve load balancing and reduce resource execution times while simultaneously increasing resource utilization rates, this paper proposes the DBSCAN clustering algorithm for containers. The experimental findings demonstrate that the suggested algorithm performs better than FCFS in terms of response time, and throughput.
***Keywords -*** Containers, Clustering Techniques, Resource Allocation, Machine Learning, Cloud Computing

---

Date of Submission: 08-06-2024 Date of acceptance: 22-06-2024

---

## I. Introduction

Container-based virtualization [1] has gained popularity recently as a lightweight, portable, and scalable method for virtualizing applications that make cloud management easier. Containers virtualize at the operating system level, removing the requirement for a separate guest operating system for every container, in contrast to traditional virtualization, which runs multiple virtual machines (VMs) on a single physical server. In order to ensure consistency across many contexts and minimize compatibility difficulties, containers first wrap applications and their dependencies into self-contained entities. These small, lightweight, and portable units ensure consistency and dependability across many contexts by encapsulating all the components an application needs to run, such as libraries, dependencies, runtime, and code. The host operating system kernels are shared by containers and use less system resources to operate; they provide better resource utilization than virtual machines (VMs). Containers support scalability and resource optimization by enabling applications to be efficiently packed and deployed across distributed infrastructures. They enable organizations to quickly adapt to variations in workload without overprovisioning hardware resources by facilitating the rapid scaling of resources to meet fluctuating demand.

Container scheduling [2] is an essential part of resource optimization and management in cloud computing environment. It involves the allocation of containers to available resources in order to maximize efficiency and meet performance requirements. The requirement for efficient container scheduling techniques [3] has grown in significance to guarantee the seamless functioning of cloud-based systems. The dynamic and unpredictable nature of contemporary cloud workloads may not be well-suited for static or rule-based approaches, which are frequently used in traditional approaches to container allocation. Static allocation algorithms [4] may result in under- or overprovisioning of resources in

dynamic environments where workload patterns change quickly, increasing costs and lowering performance. Furthermore, rule-based allocation techniques may find it difficult to achieve the best possible resource utilization and performance isolation in multi-tenant systems with varying application requirements. Clustering techniques are important for resource allocation in cloud computing systems since they effectively allocate and manage resources to satisfy varying demands of workload. Cluster analysis technique [5] is extensively employed in various study domains such as cloud computing, image processing, artificial intelligence, and machine learning. Clustering is an unsupervised machine learning technique. Inferential data sets lacking labeled output variables are used to make conclusions in the unsupervised learning approach. Clustering is the technique of dividing data sets into an established number of groups so that the data points inside a cluster have comparable characteristics. A cluster is just an arrangement of data points with the least amount of inter-cluster distance possible. Clustering is used to segregate the groups with identical traits.

Clustering is grouped into two types Hard, and Soft Clustering. One data point can belong to only one cluster in hard clustering. On the other hand, the result of soft clustering is a probability likelihood of a data point being in each of the pre-specified number of clusters. There are three main data clustering methods, Partition-based clustering [6], [7], Hierarchical clustering [8], [9], and Density-based clustering [10], [11], [12]. All clustering techniques, at their core, follow similar methodologies. First, this work computes the similarities, and uses that information to batch or group the data points.

This work will first focus on highlighting the main difference between partition-based clustering specifically k-mean clustering algorithm [13] and density-based clustering specifically DBSAN clustering algorithm. It is crucial to evaluate these variables and choose the clustering algorithm that most closely fits the unique specifications and limitations of the container assignment. Deciding whether DBSCAN or K-means is preferable for

Table 1. The difference among k-means and DBSCAN clustering algorithms

| Parameter | K-means Clustering | DBSCAN Clustering |
|---|---|---|
| Cluster Shape | The resulting clusters must all have the same feature size and be roughly spherical or convex in appearance. | The resulting clusters may not all have the same size features and are shaped arbitrarily. |
| Number of clusters | The number of clusters that are specified has an impact on K-means clustering. | There is no need to provide the number of clusters. |
| Datasets Handling | For larger datasets, K-means clustering is more effective. | High dimensional datasets are not effectively handled by DBSCAN Clustering. |
| Outliers | K-means In datasets with noise and outliers, clustering performs poorly. | Noise in datasets and outliers are effectively handled via DBSCAN clustering. |
| Number of clusters | It requires one parameter Number of clusters (K) | Two parameters are needed: Minimum Points (M) and Radius (R). |
| Densities variation | K-means clustering algorithm remains unaffected by variations in data point density. | Data points with different densities or sparse datasets are not well suited for DBSCAN clustering. |
| Pros | - The fastest approach based on centroid<br>- Scalability for huge data sets<br>- Minimize intra-cluster variance metrics | - Resistant to outliers<br>- It can manage a variety of shaped and sized clusters.<br>- It is not necessary to indicate the number of clusters |
| Cons | - Suffers when the data contains noise.<br>- It is impossible to identify outliers.<br>It minimizes intra-cluster variance, although it still has a local minimum issue.<br>- Unsuitable for datasets including non-convex shapes<br>- Difficult to determine the ideal k value | - Extremely sensitive to the two parameters, MinPts and epsilon<br>- Data sets with significant variability in densities are difficult for DBSCAN to cluster. |
| Use cases | Even cluster size, flat geometry, general purpose clusters | Uneven cluster sizes, non-flat geometry |

allocating containers rely on the specific characteristics of the workload and the goals of the allocation process. The decision between DBSCAN and K-means for container allocation ultimately comes down to computational considerations, workload characteristics, and the desired degree of cluster formation flexibility. Table 1 shows the distinction between the DBSCAN and k-means clustering. K-means is a partitioning technique that works well in situations when clusters are well-separated and generally spherical in shape. It is also simple to apply. If there is a need for a simple and computationally efficient clustering method and the workload characteristics are largely similar across containers, K-means could be helpful in container allocation.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [14] is a widely used clustering technique that effectively handles noise while finding clusters of any forms in spatial data. DBSCAN is especially well-suited for datasets where the number of clusters is unknown or fluctuates, as it does not necessitate predetermined cluster counts, in contrast to other partitioning techniques such as K-means. Two parameters are used to group together closely packed data points: MinPts, which is the lowest number of points required to form a dense (core point) region, and epsilon (ε), that is the maximum distance among points in the cluster. DBSCAN is able to discriminate between three types of points: noise points, which are neither core nor border points, reachable from a core point but lacking enough neighbors to form their own cluster, and core points, which have an adequate number of neighbors within ε. DBSCAN is robust in situations with irregularly shaped or sparse clusters because of its hierarchical approach, which enables it to recognize clusters of varied densities and shapes. DBSCAN is a strong clustering technique that performs exceptionally well at finding clusters in noisy, complex geographical data.

DBSCAN is useful when working with datasets that contain a variety of cluster sizes, densities, and forms, as well as when identifying noise spots are necessary. DBSCAN may be useful in the context of container allocation if resource requirements fluctuate or spike unexpectedly, or if the workload characteristics change dramatically throughout containers. In response to these differences, DBSCAN can discover clusters and adjust its identification, which could result in more reliable resource allocation decisions.

The DBSCAN clustering algorithm for containers is proposed in this paper to improve load balancing and reduce resource execution times while increasing the resource utilization rate for containers and VMs. The reason of choosing DBSCAN is that Partitioning methods and hierarchical clustering are appropriate only for small, well-separated clusters. Furthermore, they are also negatively impacted by data noise and outliers. Also Real-world data may have anomalies since Clusters can have any shape and noise can be presented in data. The proposed method's execution time is compared with that of the FCFS and maintains significant improvement of the resource utilization among virtual machines and physical machines. More precisely, the major contribution of this paper might be summed up as follows:

a) Provides a detailed analysis of k-mean clustering algorithm and DBSCAN clustering algorithm
b) Introduces and implements a DBSCAN clustering algorithm for containers
c) Comparing the outcome of the suggested method against that of the FCFS algorithm.

This paper has the following format: Section II presents the related work. Section III identifies the basic DBSCAN clustering technique. Section IV presents the proposed DBSCAN clustering technique for containers. Section V covers the implementation and simulation results. Finally, future work and the conclusion are discussed in Section VI.

## II. Related Work

The most recent research on resource allocation in cloud computing systems using the DBSCAN clustering method is reviewed in this section. S.M.F D Syed Mustapha et al. [10] [11] suggests a task scheduling technique that achieves great efficiency by utilizing DBSCAN (density-

based spatial clustering). Authors aim to address the shortcoming of current techniques that treat the cloud as a stand-alone entity. In contrast, the current configuration treats the entire data center as a single entity with a variety of resources and comparable performance. By clustering resources with comparable historical performance and shortening the load balancing time with ideal execution time, the DBSCAN algorithm will be used in this work to simultaneously improve the system's performance and resource utilization. According to the results, the suggested algorithm works better in terms of the average start and finish times than the conventional methods (PSO, ACO) algorithms. The paper drawback is that the work didn't consider other objective functions, such as power.

Nahid Gholizadeh et al. [12] speed up the DBSCAN execution speed so that large datasets may be processed by the algorithm in a reasonable amount of time. In order to handle the issue, the K-means++ algorithm was used to apply the first grouping to the data. Next, clustering was done independently in each group using DBSCAN. As a result, the clustering execution speed significantly enhanced and the computational load of DBSCAN execution decreased. Lastly, border clusters were combined if required. The results of using the suggested technique showed that it was able to significantly shorten the DBSCAN execution time (98% in the best-case scenario) without significantly altering the clustering's qualitative evaluation criteria. One benefit of the approach is that it can be used on a single system and doesn't require a lot of hardware resources. Using the suggested approach to address a shortage of hardware resources and powerful equipment would be especially effective. Utilizing the suggested algorithm could also prevent pointless system operations, spare a lot of system resources and time, and eventually lessen the system's depreciation. One drawback is that the suggested method ignores noise in calculations. Quality of clustering will rise if the procedure can be enhanced to incorporate noise.

Nafi Shahriar et al. [13] use four separate platforms (R, Python, Matlab, and Wolfram) to compare the runtime and accuracy of the DBSCAN and K-means algorithms. Their analysis reveals that Matlab executes K-Means more quickly than R, Python, and Wolfram. After further analysis, it was found that Matlab outperformed Python, R, and Wolfram in terms of speed for DBSCAN. The drawback of the paper is that didn't consider the usage the DBSCAN and k-mean for resource allocation in cloud computing environment.

Weipeng Jing et al. [14] suggested a better parallel DBSCAN approach (DBSCAN-PSM), which simplifies the stages involved in data splitting and regional querying. It also realizes algorithm parallelization on the Spark platform by utilizing the pre-construction strategy of KD trees. The experimental results demonstrate a significant increase in DBSCAN's efficiency compared to a stand-alone method based on the Spark platform, which is beneficial for processing large amounts of data. The drawback of the algorithm is that it doesn't make use of the statistical properties of data sets, automatically choosing eps and MinPts values to raise the algorithm's degree of automation. Also it doesn't use various partitioning techniques for various sorts of data sets to increase parallelization even more.

The previous related work neglected to take into account using DBSCAN for container resource allocation in cloud computing environments but this paper provides DBSCAN clustering algorithm for efficient container allocation in cloud computing environment.

## III. Basic DBSCAN clustering algorithm

Clusters are dense regions in the data space that are divided by regions with a lower point density. The DBSCAN method is based on the logical idea of "clusters" and "noise". A minimum number of points must exist in the vicinity of a specific radius for each point within a cluster, according to the primary concept.

### *A. DBSCAN clustering algorithm parameters*

- Epsilon: It is the maximum distance among 2 points for them to be considered neighbors to each other. It define the neighborhood surrounding a

data point, which means that two points are thought to be adjacent when their distances are less than or equal to epsilon. An excessively small epsilon number will be seen as an outlier for a significant portion of the data. If the selection is made at a very high size, the clusters will merge, resulting in the majority of the data points being in the same clusters.

- MinPts: The smallest quantity of neighbors (data points) in an eps radius. A bigger value of MinPts must be selected for larger datasets. MinPts >= D+1 is the general formula for calculating the minimal MinPts, where D is how many dimensions the dataset contains. At least three must be selected as the minimum value for MinPts.

### *B. Data Points Classification*

- Core-Point: If a point contains more than MinPts points within an epsilon, it is considered a core-point.

- Border-Point: A point near a core-point but with fewer points within an eps than MinPts.

- Outlier or Noise: A point that is neither a core-point nor a boundary-point

### *C. DBSCAN Pseudocode*

The DBSCAN technique functions in multiple crucial steps. The algorithm works by defining two parameters MinPts (a Minimum number of Points) and eps (a distance threshold). In order to get every point inside the Eps distance, the method first chooses an arbitrary point at random from the dataset. This point is referred to as a "core point" and a cluster is formed if the total number of points retrieved inside the eps distance zone is more than MinPts. After that, the algorithm gathers all of the points that are within each core point's eps distance and adds them to the cluster. All core points go through this process again until no more points can be added to the cluster. After that, the algorithm moves on to the next unexplored point and keeps going until every point has been reached as shown in Algorithm 1.

DBSCAN starts by choosing a dataset data point that hasn't been visited yet. Based on a predetermined distance threshold ε (epsilon), the algorithm determines the neighborhood of the specified point. The chosen point is designated as a core-point if exists more points in this neighborhood than there are below a predetermined threshold (MinPts).

**Algorithm 1 Basic DBSCAN Clustering Algorithm**

**Input:** Dataset D, eps, and MinPts.

**Output:** The assignment of each receiving datum to a cluster.

```
/Initialization/
Initialize the labels of all the data points to be Unvisited

For each point P in dataset D do
    if label(P) is not Unvisited then
        label(P) ← Visited
    neighbors = get points that are within a
        point's eps distance

    if number of neighbours < MinPts
        Identify a point as noise
    else
        Construct a new cluster
        Include a point in the cluster
        For each neighbor in neighbors
            If neighbor is not visited
            Mark neighbor as visited
            NewNeighbors = retrieve points
        within eps distance of neighbor
            If number of NewNeighbors >
        MinPts
                    Add NewNeighbors to
        neighbors
        end for
        add cluster to clusters
end for
return clusters
```

DBSCAN adds reachable points inside the ε-neighborhood to the cluster iteratively, starting from a core point. If these points are within the epsilon neighborhood of a core point or are core points themselves, they join the same cluster. Border points are those that are within a core point's ε-neighborhood but do not have enough neighbors to be classified as core points in and of themselves. Although border points are a component of the cluster, they don't help it grow. Noise points are points that are neither core nor

within a core point's ε-neighborhood. These points are regarded as outliers as they don't fit into any cluster. Clusters are created as the algorithm goes along by combining core points and the points that border them. The process ends once each point has been examined and categorized into a cluster.

## IV. DBSCAN clustering algorithm for containers in cloud computing environment

**Algorithm 2 DBSCAN Clustering Scheduling Algorithm for Containers in Cloud Computing**

**Input:** VM list, Container list, eps, and MinPts.

**Output:** Allocation of containers on VMs

```
/Cloudsim initialization/

Initialize CloudSim by creating the Datacenter
Broker, containers, virtual machines and cloudlets.
Provide the unallocated container list and
unassigned VM list to the Datacenter broker
Initialize eps and MinPts

/Clustering the containers using DBSCAN /
for each container in containerlist do
        Get the values of MIPS and RAM size of
        each container
end for

for each container C in containerList do
        if C is not Unvisited then
            C ← Visited
        neighbors = get containers that are within
            a point's eps distance

        if number of neighbours < MinPts
            Add C to list of
            Unassigned_Containers
        else
            Create a new cluster
            Container_Cluster
            Add C to cluster Container_Cluster
            For each Container Q in the
            neighbors (neighborhood of C) do
                If Q is not visited
                Q ← Visited
                NewNeighbors = retrieve
            containers within eps distance of Q
                If number of NewNeighbors >
            MinPts
                            Add NewNeighbors to
            neighbors
            end for

            If Q does not currently belong to any
            cluster
             Add Q to cluster Container_Cluster
end for
Add Container_Cluster to list of Container clusters
Container_Clusters
return Container_Clusters

/ Clustering the VMs using DBSCAN
for each VM in VMlist do
        Get the values of MIPS and RAM size of
        each VM
end for
Apply DBSCAN clustering algorithm on the VMs
for each container in containerlist do
        Assign container to VM of appropriate
        cluster
end for
sendNow (container id, virtual machine id)
```

Algorithm 2 presents the pseudocode for the suggested DBSCAN clustering container allocation algorithm in cloud computing environment. DBSCAN could be beneficial if the workload characteristics vary significantly across different containers, or if there are unpredictable spikes or variations in resource requirements. DBSCAN can adapt to these variations and identify clusters accordingly, potentially leading to more robust resource allocation decisions. Firstly, the Datacenter Broker, virtual machines, and cloudlets are created as part of the initialization of the CloudSim simulator. Datacenter Broker receives the list of unallocated containers and unassigned VMs. The Modified DBSCAN clustering algorithm is utilized to categorize the containers. Datacenter broker calculates each virtual machine's processing capacity based on RAM, size, and MIPS and deploys the modified DBSCAN clustering technique on VMs then assign the containers to VM of appropriate cluster.

## V. Implementation and simulation results

This section verifies the modified DBSCAN placement strategy for containers in cloud computing environment, which are subsequently assessed using ContainerCloudSim simulator.

### *A. Implementation Environment*

ContainerCloudSim [15] has the same layered architecture as CloudSim, with the necessary modifications to integrate the container idea. ContainerCloudSim provides containerized cloud data centers, hosts, containers, virtual machines, applications, and their workloads. Both VM-level and container-level container provisioning are offered by the simulator. The percentage of the virtual machine's total processing power which is allotted to every container is defined at the VM level. On the other hand, at the container level, every application service hosted on the container can be allocated a set number of resources. As a finer abstraction of an application service housed in the container, a task unit is thought to facilitate interoperability with CloudSim. In the latest version of the ContainerCloudSim, time shared and space shared provisioning strategies are applied for both levels.

### *B. Parameter setting*

Using ContainerCloudSim, we evaluate our proposed DBSCAN clustering scheduling method for cloud computing containers and compare it with other job scheduling algorithms. Table 2 shows how the suggested algorithm's parameters are set up to give us the optimum performance. We initialize the value of the epsilon to 5 and the MinPts to 15.

***Table 2. ContainerCloudSim Parameter Setting***

| Type | Parameters | Value |
|---|---|---|
| **Containers** | *TYPES*<br>*MIPS*<br>*PES*<br>*RAM*<br>*BW* | 3<br>4658, 9320, 18636<br>1<br>128, 256, 512<br>2500 |
| **Virtual Machine** | *TYPES*<br>*PES*<br>*RAM*<br>*BW*<br>*SIZE* | 4<br>2, 4, 1, 8<br>1024, 2048, 4096, 8192<br>100000<br>2500 |
| **Hosts** | *TYPES*<br>*MIPS*<br>*PES*<br>*RAM*<br>*BW*<br>*STORAGE* | 3<br>37274<br>4, 8, 16<br>65536, 131072, 262144<br>1000000<br>1000000 |

### *C. Experimental results*

A comparison is made between our DBSCAN clustering scheduling technique for cloud computing containers and FCFS scheduling algorithm [17]. DBSCAN is one of the most effective and frequently referenced density-based clustering algorithms. It is thought to be able to detect clusters of random size and shape in sizable datasets tainted by noise with a considerable degree of accuracy. In FCFS scheduling algorithm, Requests are queued in the order that they are received by FCFS, which processes them automatically. FCFS is the most basic CPU scheduling algorithm currently in use.

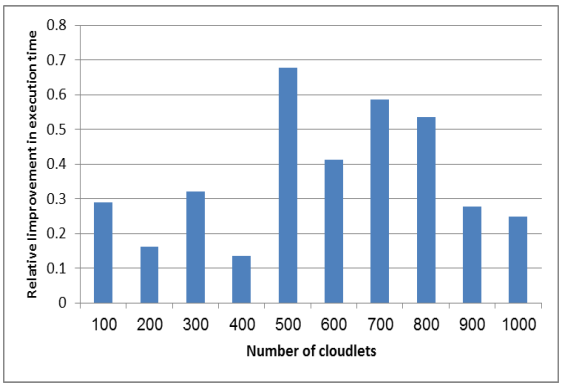


***Fig. 1. Relative improvement in execution time of the proposed DBSCAN algorithm w.r.t FCFS algorithm***

The relative increase in execution time of the suggested DBSCAN algorithm for containers in comparison to the FCFS algorithm is displayed in Fig. 1. It is shown that the proposed method achieves a satisfactory balance of system loads and minimizes the required time. We find that as the number of cloudlets grows, the relative response time rises linearly. It performs best when there are 500 cloudlets because the execution time advantage over the FCFS method is 6%. The relative improvement in execution time is 5.7%, 5.3%, and 4.3% for 700, 800, and 600 cloudlets, respectively. When the number of the cloudlets is 400 the relative improvement in execution time is the worst since it is about 1.3%.

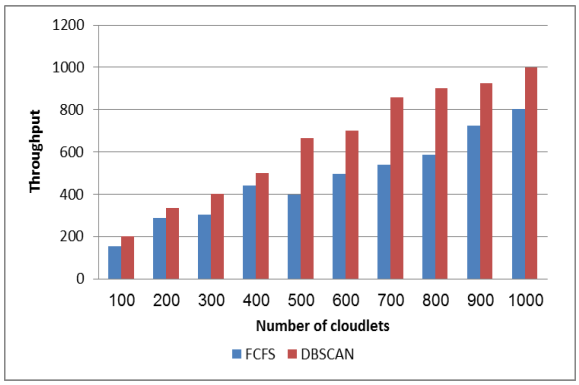


***Fig. 2. Throughput***

In terms of throughput across all cloudlets, Fig. 2 shows that the recommended method performs better than FCFS. While some of the cloudlets we sent were successful, some weren't. Containers can be optimally placed on virtual machines (VMs) with shorter reaction times and faster throughput thanks to the recommended technique, which works best when there are 1000 cloudlets. The algorithm performs worst when the number of cloudlets is small (100 and 200 cloudlets).

## VI. Conclusion

In the context of container allocation, DBSCAN can be used to group containers with similar resource requirements and workload characteristics into clusters, allowing for more efficient resource utilization and better performance isolation. By identifying clusters of containers that share common resource demands, DBSCAN enables containers to be allocated to hosts in a way that minimizes resource contention and maximizes resource utilization. Additionally, DBSCAN can adapt to changing workload patterns and dynamically adjust cluster boundaries in response to fluctuations in resource demand, making it well-suited for dynamic cloud environments where workload patterns are constantly evolving. The DBSCAN algorithm's primary benefit is that datasets do not require predetermination of the number of clusters. Given that the DBSCAN method can accurately and efficiently handle the noise points, it is more useful to identify a group that is surrounded by noise as opposed to another group.

This paper suggests a DBSCAN clustering technique for containers in order to enhance load balancing, decrease resource execution times, and increase resource utilization rates at the same time. The experimental findings demonstrate that, in terms of execution time and throughput, the suggested approach outperforms the FCFS algorithm. According to the experimental findings, the suggested method outperforms alternative algorithms by a margin of 6%. Instead of using simulation in the future, we can implement the suggested DBSCAN allocation technique on an actual platform. In order to optimize the placement of containers on virtual machines, we can also experiment with various machine learning methods.

## REFERENCES

[1] S. Abraham, A. K. Paul, R. I. S. Khan and A. R. Butt, "On the Use of Containers in High Performance Computing Environments," 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), Beijing, China, pp. 284-293, 2020, doi: 10.1109/CLOUD49709.2020.00048.

[2] M. K.Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment", Journal of Cloud Computing, vol. 8, 2019. https://doi.org/10.1186/s13677-019-0131-1

[3] A. M. Hafez, A. Abdelsamea, A. A. El-Moursy, S. M. Nassar and M. B. E. Fayek, "Modified Ant Colony Placement Algorithm for Containers," 2020 15th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, pp. 1-6,2020, doi: 10.1109/ICCES51560.2020.9334671.

[4] K. Shrikant, V. Gupta, A. Khandare, P. Furia,"A Comparative Study of Clustering Algorithm". Intelligent Computing and Networking, Springer, Singapore vol. 301, 2022. https://doi.org/10.1007/978-981-16-4863-2_19

[5] A. Priyadarshini, S. K. Pradhan, S. Pattnaik, S. R. Laha, and B. K. Pattanayak, "Dynamic Task Migration for Enhanced Load Balancing in Cloud Computing using K-means Clustering and Ant Colony Optimization", International Journal on Recent and Innovation Trends in Computing and Communication, vol. 11, no. 7, pp. 156–162 , July, 2023.

[6] G. Muthusamy, S. R. Chandran, "Cluster-based Task Scheduling Using K-Means Clustering for Load Balancing in Cloud Datacenters", Journal of Internet Technology, Vol. 22, No.1, pp. 121–130, 2021.

[7] P. Charles, and V. Alagumalai,"Load Balancing in Cloud Computing Using Agglomerative Hierarchical Clustering Approach", Journal of

Advanced Research in Dynamical and Control Systems, vol. 4, pp. 1720-1723, 2019.

[8] X. Wu, J. Wei, S. Yuan, Z. Chen and X. Wang, "Hierarchical Clustering Algorithm Based on Fast and Uniform Segmentation," *2022 12th International Conference on Cloud Computing, Data Science & Engineering*, Noida, India, pp. 88-93, 2022, doi: 10.1109/Confluence52989.2022.9734143.

[9] A. Fahim, "A varied density-based clustering algorithm", Journal of Computational Science, Vol. 66, 2023, https://doi.org/10.1016/j.jocs.2022.101925.

[10] S.M.FD.S. Mustapha, P. Gupta, "DBSCAN inspired task scheduling algorithm for cloud infrastructure", Internet of Things and Cyber-Physical Systems, vol. 4, pp. 32-39, 2024. https://doi.org/10.1016/j.iotcps.2023.07.001

[11] S.M.FD.S. Mustapha, P. Gupta, " Fault aware task scheduling in cloud using min-min and DBSCAN", Internet of Things and Cyber-Physical Systems, vol. 4, pp. 68-76, 2024. https://doi.org/10.1016/j.iotcps.2023.07.003.

[12] N. Gholizadeh, H. Saadatfar, Hamid, N. Hanafi, "K-DBSCAN: An improved DBSCAN algorithm for big data". The Journal of Supercomputing, vol. 77, pp. 1-22, 2021.https://doi.org/ 10.1007/s11227-020-03524-3.

[13] N. Shahriar, S. M. Akib Al Faisal, M. M. Pinjor, M. A. Sharif Zobayer Rafi, and A. Rahman Sarkar, "Comparative Performance Analysis of K-Means and DBSCAN Clustering algorithms on various platforms," 2019 22nd International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, pp. 1-6, 2019. https://doi.org/10.1109/ICCIT48885.2019.9038535.

[14] W. Jing, C. Zhao, C. Jiang, "An improvement method of DBSCAN algorithm on cloud computing". Procedia Computer Science. vol. 147, pp. 596-604, 2019. https://doi.org/10.1016/j.procs.2019.01.208.

[15] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "ContainerCloudSim: An Environment for Modeling and Simulation of Containers in Cloud Data Centers", Softw. Pract. Exper., pp.1–17, 2010.