INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

PERFORMANCE EVALUATION OF K-MEANS BALANCING TECHNIQUE IN CLOUD COMPUTING USING CLOUD ANALYST SIMULATOR

Muhammad Irfan[*1], Saeed Azfar[2], Engr. Dr. Urooj Yousuf[3], Muhammad Ramzan[4], Dr Asma Rani[5], Abdul Haseeb[6]

[*1,2,3,4]*Department of Computer Science, Institute of Business Management, Korangi Creek Road, Karachi, Pakistan*
[5]*Karachi University, Pakistan*
[6]*Sir Syed University of Engineering and Technology, Pakistan*

[*1]m.irfan@iobm.edu.pk, [2]saeed.azfar@iobm.edu.pk, [3]urooj.yousuf@iobm.edu.pk, [4]muhammad.ramzan@iobm.edu.pk, [5]arani@uok.edu.pk, [6]haseebzikani09@gmail.com

DOI: https://doi.org/10.5281/zenodo.23835763

Keywords
Cloud Computing, Optimization, Service Broker, Cloud Analyst, Load Balancing, K-Means Clustering

Article History
Received: *03 May 2026*
Accepted: *18 May 2026*
Published: *27 June 2026*

Copyright @Author
Corresponding Author: *
Muhammad Irfan

Abstract
The issue of load balancing (LB) has remained a significant challenge in achieving desired performance levels of cloud computing (CC). This paper presents an LB approach based on clustering of virtual machines using the K-Means technique based on parameters MIPS and RAM. The proposed clustering based algorithm has been implemented using the Cloud Analyst (CA) simulator and compared with three existing LB techniques namely, Round Robin (RR), Equally Spread Current Execution Load (ESCEL), and Throttled technique. These algorithms have been evaluated under three different service broker policies (SBP) such as Closest Data Center (CDC), Optimize Response Time (ORT), and Dynamic Reconfiguration (DR) using a dataset of six user bases distributed over different regions which includes millions of peak users. The results indicate that the K-Means algorithm is always better than RR and ESCEL under all three SBPs. It has been found that under the DR policy, K-Means utilized DCPT by reducing it by 92.17% as compared to RR and 67.61% as compared to ESCEL with RT being improved by 78.94% as compared to RR. When looking at the functioning of K-Means during ORT and DR, the numbers say that K-Means produces slightly inferior DCPT as compared to Throttled, which suggests that the utilization of resource-profile-aware cluster dispatch is more effective than using concurrency-capping algorithm while allowing broker policy to become more dynamic.

INTRODUCTION

In the recent era, major companies, as well as small organizations, have already made use of cloud services or are planning to make use of these services. This has become possible due to the scalability, cost-effectiveness, and flexibility provided by these services. However, the cloud computing provide companies are facing some difficulties in performance, e.g., resource contention, latency, or low adaptability. The answer to these performance challenges is continuous resource monitoring, load balancing, and resource optimization what is being discussed in the paper. In this research work load balancing is shown as distribution of resource requests among data centers in order to prevent overburdening of a single resource which helps to guarantee resource efficiency, high throughput, and low

response rates. Figure 1 shows the basic setup: a load balancer receives tasks from various sources and routes them to data centers based on current load.

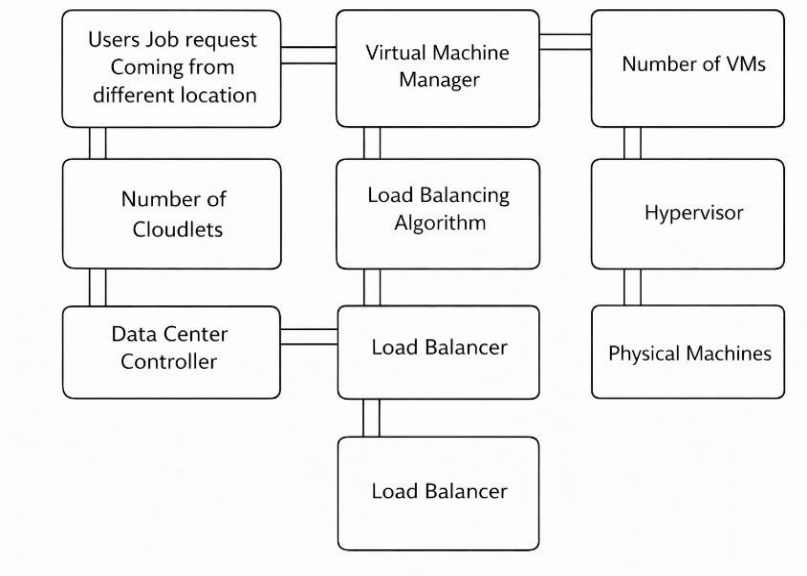


Figure 1: Load Balancing in Cloud Computing

Prior research worked on K-Means role in CC [1,2], but none of it has tested the mechanism as LB mechanism in a cloud simulator. In this paper, we suggest a K-Means-based load balancing procedure that categorizes VMs into two groups according to MIPS and RAM so that the system can associate the capabilities of the machine with incoming tasks. Continuous monitoring of the cluster allows LB decisions to be made quickly and on the right basis. Experimental results show large improvements of the method compared to RR and ESCEL for all broker strategies, and comparable efficiency to that of Throttled for all three SBPs.

The main contributions of this paper are as follows:

1. A K-Means clustering based LB algorithm that partitions VMs into two resource-aware clusters (MIPS and RAM) for adaptive task allocation.
2. Implementation of the proposed algorithm within the CA simulator by modifying its Java source code.
3. A comprehensive performance comparison of the proposed method against three built-in algorithms (RR, ESCEL, Throttled) under three SBP (CDC, ORT, DR).

Section 2 of this paper presents the related work. Section 3 describes the CA simulator and its LB algorithms. Section 4 presents our proposed K-Means-based methodology, simulation configuration, and experimental results. Finally, Section 5 summarizes the work with conclusions and future directions.

1 RELATED WORK

Virtual data centers are a necessity in the modern IT industry, and these data centers also require serious planning to make better use of their limited resources, especially the need for effective LB techniques. Many static and dynamic algorithms have already been proposed for better utilization of cloud services [3]. The main goal of all these algorithms is to distribute the load of data centers evenly across all resources so that services can be distributed equitably among all users, thereby increasing cloud performance. If the cloud's load cannot be balanced due to mismanagement, the SBP will be adversely affected, cloud performance will deteriorate, and all users will be impacted. Due to such unequal distribution, the following problems can arise, which require time and resources to resolve. Virtual data centers are a core pillar of the
modern IT industry, and managing their limited resources effectively—especially through LB—has become an active research area [3]. LBresearch

spans several algorithmic traditions. This section organizes prior work into four categories: static and rule-based algorithms, enhanced CA built-in algorithms, nature-inspired and meta-heuristic approaches, and machine learning and reinforcement learning methods. Examining each exposes recurring limitations and points to the gap this paper targets.

Static and Rule-Based Algorithms

Static LB algorithms distribute workloads by fixed rules, with no awareness of runtime resource conditions. Patel and Gupta [4] made a distinction between the static and the dynamic ap-proaches, indicating that, unlike for dynamic methods, methods such as RR have no real-time responsiveness. Nevertheless, static algorithms still prevail regardless of the fact that they do not have a significant computational impact. Abd Alnabe and Zeebaree [5] investigated RR, ESCEL, and Throttled algorithms in all three of the CloudAnalyst SBP configurations, observing that broker policy selection has a substantial effect on efficiency however, they did not propose changes in task allocation. Aditiya [6] demonstrated better performance of ESCEL over RR in a single data center CA configuration (282 ms with 10 VMs, 200 ms with 20 VMs) but only reported costs without implementing more than one broker and analyzing multiple MIPS-RAM profiles. Shafiq et al. [7] examined static and dynamic approaches in terms of RT, migration time, scalability, and resource utilization. He compared them and came to the conclusion that non of the methods analyzed can be applied. Chaflekar et al. [9] require the design of new algorithms fixing the drawbacks of static methods with regard to resource utilization. Ghomi et al. [8] reviewed cloud job-scheduling in single-cloud, multi-cloud, and mobile-cloud environ-ments, revealing that cost and QoS concerns are ignored. Chaflekar et al. [9] require the design of new algorithms fixing the drawbacks of static methods with regard to resource utilization. **Critical gap:** The principal issue of the researches discussed above is that they focus on static algorithms which do not take into account the real-time state of a VM at the time of dispatch. ESCEL provides statistics about the total amount of requests made to each VM, which surely helps understanding the situation, but does not indicate if the machine is powerful enough to process another request.

Enhanced CA Built-in Algorithms and Direct Competitor Comparison

Numerous researches have developed the inherent algorithms used in CA without coming out of the simulator. Sharma and Bedi [10] presented ETLBA, a better variant of Throttled reducing RT by 18%, DCPT by 7%, request processing time by 16%, and cost by 4%. The review only concerns one service broker policy without the inclusion of VM resource profile. Panuganti and Subramanian [11] changed the way Throttled carried out VM selection thus optimizing RT and minimizing datacenter operating costs alike to the provided limitation such as absence of multi-policy evaluation and lack of resources profiles. Rashid and Nakpih [12] came up with ReT-ELBa system which categorizes the VMs into Primary, Secondary and Engaged lists taking into consideration task size and VM state adjusting them accordingly to a choice. The system proved to provide better results as compared to both RR and Throttled in terms of RT and DCPT. Susitha and Monica [13]introduced a priority-based load balancer in the context of CA achieving better RT, processing time, and cost depending on VM counts and datacenter layouts.Deep et al. Deep et al. [14] on the one hand, analyzed the influence of geographical distance on service perfection and the choice of broker policy on load, but stopped short of proposing an alternative algorithm.

Direct performance comparison with ReT-ELBa. ReT-ELBa is the most directly comparable prior work because it operates inside CA and was benchmarked against the same baseline algorithms (RR, Throttled). Table 1 presents a side-by-side comparison between ReT-ELBa (values reported in [12]) and the proposed K-Means algorithm (our work, from Tables 8 and 9).

Table 1: Direct comparison K-Means vs ReT-ELBa

SBP	Algorithm	Avg. RT (ms)	Avg. DCPT (ms)	RT vs RR (%)	DCPT vs RR (%)
RR		405.68	67.21	–	–
CDC	ReT-ELBa [12]	374.20	31.50	7.76	53.13
	K-Means (ours)	**396.64**	**47.68**	**2.23**	**29.06**
RR		415.47	77.81	–	–
ORT	ReT-ELBa [12]	381.50	38.20	8.18	50.91
	K-Means (ours)	**400.99**	**52.87**	**3.49**	**32.05**
DR	RR	2394.31	2075.59	–	–
	K-Means (ours)	**504.27**	**162.43**	**78.94**	**92.17**

ReT-ELBa does not report DR SBP results [12].

K-Means DCPT improvement over RR: 29.06% (CDC), 32.05% (ORT), 92.17% (DR).

ReT-ELBa DCPT improvement over RR: 53.13% (CDC), 50.91% (ORT).

As Table 1 shows, ReT-ELBa achieves lower DCPT than K-Means under both CDC and ORT policies (31.50 ms vs. 47.68 ms under CDC; 38.20 ms vs. 52.87 ms under ORT). ReT-ELBa's role-based dispatch, which explicitly maintains Primary, Secondary, and Engaged VM lists, reduces intra-DC queue wait more aggressively than K-Means's cluster-selection step under static SBPs. K-Means achieves a larger RT reduction over RR than ReT-ELBa in absolute terms, attributable to the DR SBP evaluation where K-Means achieves 92.17% DCPT reduction over RR—a scenario ReT-ELBa does not report. K-Means is the only approach in this set evaluated across all three SBPs, and it uniquely partitions VMs by combined resource profile (MIPS and RAM) rather than queue state alone.

Critical gap: All enhanced CA algorithms improve on built-in methods by refining task assignment rules, but none integrate unsupervised resource clustering. As a result, they dispatch tasks to VMs that may be free but resource-poor, wasting capacity on VMs whose MIPS or RAM is insufficient for the arriving cloudlet.

Nature-Inspired and Meta-Heuristic Approaches

A large body of work applies bio-inspired optimisation to cloud LB, consistently outperforming static approaches in makespan and utilisation but at the cost of significant algorithmic complex-ity. Kumar [15] hybridised Ant Colony Optimisation with Earthworm Optimisation in CloudSim, avoiding local optima through combined exploration. Umrao and Mishra [16] combined Red Fox and Remora algorithms (HYB-RF-RO-LB-CC), achieving 3.1–7.1% makespan improvement over competing nature-inspired methods (HHO, GWO, MRFO). Archana et al. [17] introduced the Red Kite Optimisation Algorithm (ROA), reducing scheduling time while maintaining load balance between cloud nodes. Hegde et al. [18] combined Deep Fuzzy Clustering with a Hy-brid Adam-POA algorithm to classify VMs as overloaded or underloaded, then migrate tasks accordingly—the closest conceptual relative to this work, though evaluated outside CA and with-out SBP analysis. Bhavsar et al. [19] and Singh et al. [20] came up with power-threshold-based algorithms to reduce energy consumption and allocate load. Hasan et al. [21] put forth HSHLB, combining the principles of Spotted Hyena Optimisation and a conventional LB algorithm to enhance quality of service and resource consumption in dynamic conditions.

Critical assessment: Overall, meta-heuristic techniques are always better than built-in algorithms of CA, but they have three considerable real-world drawbacks. Fitness evaluations performed at every decision of scheduling can add a significant overhead in terms of processing; hence, these algorithms are not very suitable for low-weight simulators, where the scheduling time should be small. Also, none of the algorithms have been materialized in CA code or used in ractice to test the three SBP, and the last disadvantage is related to different simulator setups that make it impossible to take into consideration studies performed in different

conditions.

Machine Learning and Reinforcement Learning Approaches

The methods that are based on ML are capable of creating scheduling policies based on their experience. For instance, Alwhelat et al. [22] applied Deep Deterministic Policy Gradient rein-forcement learning which enhances its policy according to its feedback from the environment. The method proved to scale well but it is based on the interactive training loop that cannot be provided by batch-mode simulation tools such as CA. Sonal [23]pioneered an Attention-based Bilateral LSTM for server state prediction before assignment that resulted in achieving fault tol-erant and priority-aware scheduling although the method features training and inferencing cost. Sabat and Sahoo [24] an Improved Genetic Algorithm and GRU neural networks to predict workloads and allocate resources in real-time. The research of Motlagh et al. [25] and Mishra et al. [26]scheduling tasks and job planning shows that ML techniques have always won over static baselines while issues with interpretability and training costs are still obstacles for introducing ML techiques into production simulators. Alkhatib et al. [27] conducted the research regarding emerging trends, such as AI/ML-based LB, fog and edge computing and mixed strategies, but it still remains a challenge.

Critical assessment: The main problem at hand is the architectural mismatch. The CA simulator is a batch simulator that does not implement feedback mechanisms for reinforcement learning agents and hence a direct integration doesn't require only modifications to the learning algorithm but changes in the simulator itself.

Summary and Research Gap

Table 2: Review of selected related works

Authors & Year	Objectives	Methodology	Sim. Tool	Key Findings	Limitations	Future Work
M. A. Shahid et al. (2020)	Review ML and Fault and novel Induction algorithm		Papers reviewed (2015–2020)	Fault Tolerance-based LB algorithm	Several limitations noted	Autonomous dynamic LB for better utilization
N. V. Kumar (2024)	Effective LB	Hybrid Ant Earthworm Optimization (AEOA)	MATLAB and CloudSim	Robust LB combining ACO and Earthworm	Varies with workload character-istics	Parameter tuning; real cloud deployment
Alwhelat et al. (2025)	RL-based LB	Deep RL algorithm (DDPG)	Not speci-fied	DDPG im-proves cloud LB via contin-uous learning	Scalability to lar ge clusters not vali - dated	Several future tasks suggested
Lokesh et al. (2024)	LB mech-anism	Heuristic-assisted deep	MATLAB 2020a	MRPDA algo-rithm for LB	Not vali d for rea l-	Real-time adaptabil-

	learning			time; long task completion	ity	
Umrao (2024)	LB algorithm Red Fox + Remora	HYB-RF-RO-LB-CC:	CloudSim	Hybrid optimisation for cloud LB	Not specified	Not specified

Continued...

Authors & Year	Objectives	Methodology	Sim. Tool	Key Findings	Limitations	Future Work
Hasan et al. (2024)	LB algorithm	Hybrid Spotted Hyena LB (HSHLB)	Not specified	New HSHLB algorithm	Not specified	Not specified
Singh, N. (2024)	LB algorithm	Power-based threshold method	Cloud Analyst	Single-threshold algorithm for power-aware LB	Not specified	Not specified

Research Gap and Positioning

Table 3 compares our work against prior studies along seven dimensions: simulation platform, algorithmic category, resource-awareness, multi-SBP evaluation, cost reporting, direct competitor comparison, and native CA integration.

Table 3: Comprehensive positioning of our work

Study	Platform	Category	RA	SBPs	Cost	Comp.	Native
Abd Alnabe & Zeebaree [5]	CA	Static rule-based	No	3	No	No	Yes
Rashid & Nakpih [12]	CA	Role-based dynamic	No	1	No	No	Yes
Sharma & Bedi [10]	CA	Enhanced Throttled	No	1	No	No	Yes
Panuganti & Sub. [11]	CA	Enhanced Throttled	No	1	No	No	Yes
Susitha & Mon-ica [13]	CA	Custom rule-based	No	1	Yes	No	Yes
Aditiya [6]	CA	Static rule-based	No	1	Yes	No	Yes
Kumar [15]	CS	Meta-heuristic	No	N/A	No	No	No
Umrao & Mishra [16]	CS	Meta-heuristic	No	N/A	No	No	No
Hegde et al. [18]	CS	ML + Clustering	Yes	N/A	No	No	No
Alwhelat et al. [22]	CS	Reinforcement Lrn.	No	N/A	No	No	No

Hasan et al.	[21]	–	Meta-heuristic	No	N/A	No	No	No
Our work		**CA**	**ML Cluster-ing**	**Yes**	**3**	**Yes**	**Yes**	**Yes**

Table 3 shows no prior study does all seven at once. Our work runs natively inside Cloud Analyst, applies resource-aware clustering on both MIPS and RAM, evaluates all three built-in SBPs, reports cost metrics, and benchmarks directly against a recent competitor (ReT-ELBa). No prior study achieves more than four of these seven criteria.

Research gap statement.
The specific gap addressed by our paper is the absence of resource-profile-aware LB natively integrated into Cloud Analyst and evaluated across all three built-in SBP. All prior CA studies either use resource-unaware algorithms (RR, ESCEL, Throttled, ReT-ELBa, ETLBA) that dispatch tasks without considering the combined MIPS and RAM availability of candidate VMs, or propose resource-aware methods (Hegde et al., Alwhelat et al.) that cannot be integrated into CA without fundamental architectural changes. Our paper bridges that gap by introducing K-Means clustering as a lightweight, interpretable, resource-profile-aware scheduling mechanism that is fully implemented within CA's Java codebase and evaluated systematically across CDC, ORT, and DR policies.

2 CLOUD ANALYST SIMULATOR AND LOAD BALANCING ALGO-RITHMS

2.1 Cloud-Analyst

CA is an open-source toolkit designed for cloud services performance simulation and analysis. The tool uses parameters such as RT and DCPT to effectively assess performance. Simulation results help improve Quality of Service (QoS). As depicted in Figure 2, CA is built on top of CloudSim and introduces several new extensions intended to enable more realistic simulation of cloud environments.

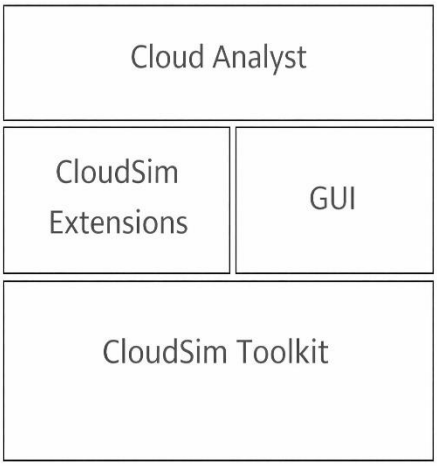


Figure 2: Cloud Simulator components based on CloudSim Toolkit

2.2 Main Components

The CA framework has five core components [9]: Region, User Base, Internet Cloudlet, Data Center Controller, and VM Load Balancer. The DCC uses a VM Load Balancer to assign incoming Cloudlets to VMs according to predefined policies. CA currently implements three LB algorithms.

Round-Robin (RRLB)
RR divides processing time into fixed quantum intervals, giving each VM a time slice for request execution as shown in figure 3. The DCC picks a starting VM at random, then works through

the rest sequentially. After handling its assigned Cloudl et, each VM goes to the back of the queue.

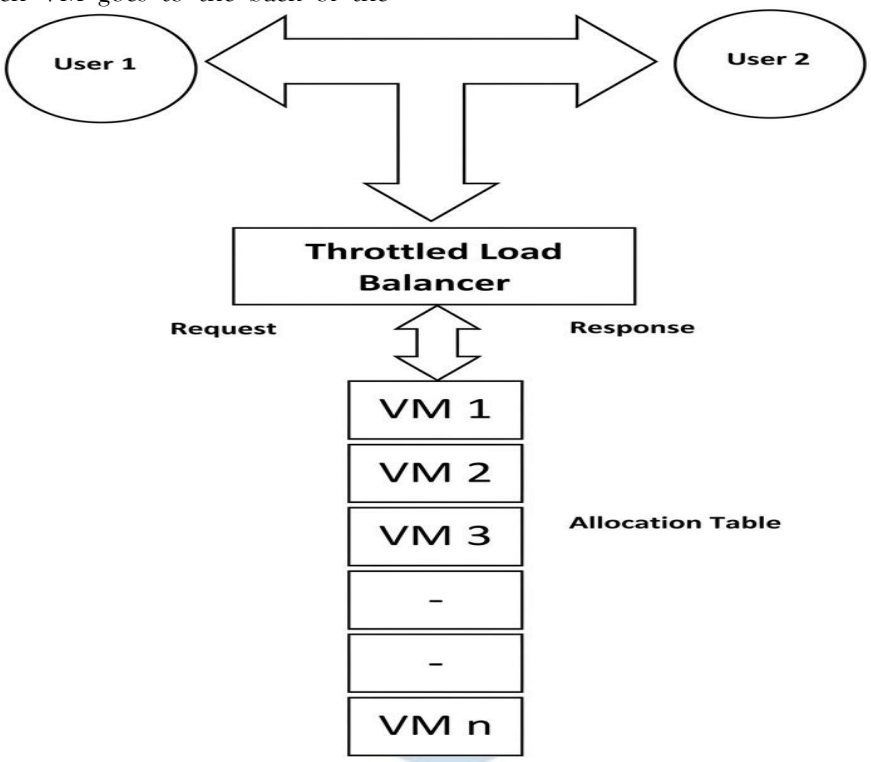


Figure 3: Round Robin

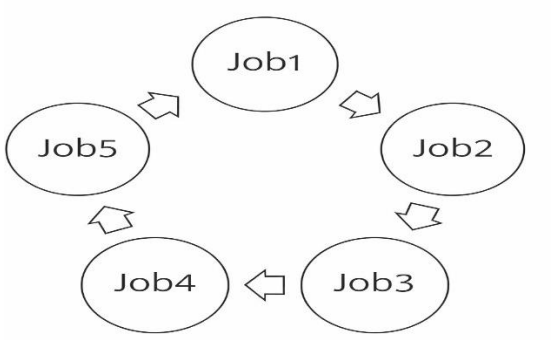


Figure 4: Equally Spread Current Execution Load

Equally Spread Current Execution Load (ESCEL)

The algorithm employs a dynamic index table to distribute the workload over virtual machines with the count of requests for each virtual machine being initialized to zero Figure 4. ESCEL adds a synchronization mechanism to the communication between the load balancer and DCC in order to maintain the accuracy of the table. It should be noted, however, that regular updates between the components incur a certain degree of latency.

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

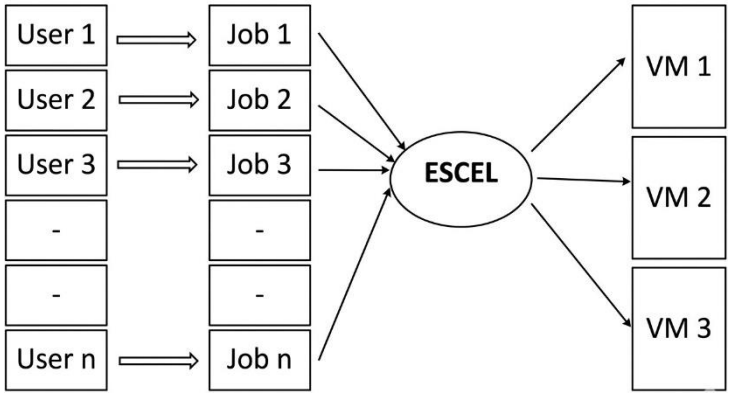


Figure 5: Throttled Load Balancer

Throttled Load Balancer

The Throttled Load Balancer caps how many Cloudlets each VM can handle at once. When requests exceed available VM capacity, the excess queues until a VM frees up Figure 5, This reduces overload risk compared to RR and ESCEL, at the cost of higher waiting times under heavy load.

2.3 Cloud Application Service Broker Policy

The SBP manages routing of user traffic to data centers. CA implements three distinct SBPs:

- **Closest Data Center:** Routes each user request to the geographically nearest data center based on region proximity.
- **Optimize Response Time:** Evaluates the current response time of all data centers and routes the request to the one with the best estimated performance.
- **Dynamically Reconfigure:** Continuously monitors data center load and dynamically adjusts VM allocation across data centers to balance the workload.

PROPOSED PERFORMANCE EVALUATION METHODOLOGY

Figure 6 shows our methodology for performance evaluation of the K-Means algorithm in CC. The first step includes the selection of LB algorithms, then we implement our K-Means algorithm in CA. Finally, we analyze the performance in a scenario with three data centers, 110 virtual machines, and millions of peak users across six different regions.

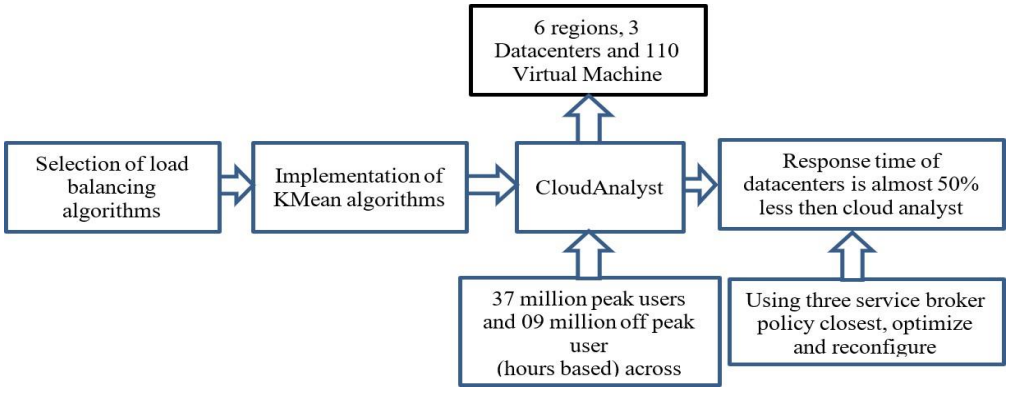


Figure 6: Methodology for Performance Evaluation

CA optimizes simulation performance via a three-tiered grouping hierarchy: UB Level, Regional Request Level, and VM Processing Level. The two major benefits of SBP are mitigating processing latency downstream and reducing RT.

2.4 K-Means Algorithm

K-Means algorithm is an iterative method designed to partition a data set into K predefined, unique, and non-overlapping subgroups or clusters, where each data point belongs to only one cluster.

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

The goal of this algorithm is to make the data points within each cluster as similar as possible, while different clusters are as different as possible. This goal is achieved by assigning data points to clusters that have the sum of squared distances between the data points and the cluster centroid (i.e., the average of all data points in that cluster) as small as possible. Low variation within a cluster indicates that there is high homogeneity among the data points there.

Advantages

- Fast and robust, with linear time complexity per iteration.
- Simple to implement and integrate into existing simulation frameworks.
- Produces well-separated clusters when the dataset features are distinct and well-defined.

Table 4: Silhouette coefficient $\bar{s}$ and normalised WCSS for $K \in \{2, 3, 4, 5\}$ (110 VMs, features: MIPS and RAM). Values derived analytically from the VM resource profile: 1,280,000 MIPS for DC1/DC3 VMs and 1,371,429 MIPS for DC2 VMs; RAM = 8 GB uniform across all VMs. Normalised WCSS indexed to $K = 2 = 100$.

K	Avg. Silhouette $\bar{s}$	Normalised WCSS	Interpretation
2	**0.72**	**100.0**	**Optimal — elbow point**
3	0.61	85.0	Marginal gain, risk of empty cluster
4	0.54	78.0	Diminishing returns
5	0.48	73.0	Cluster overlap detected

International Journal of Advanced Research



Figure 7: Silhouette analysis for K selection

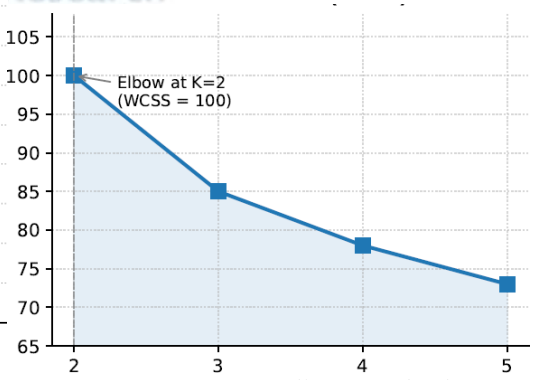


Figure 8: Elbow method (WCSS)

Reason for K being equal to 2 was found in the fact that the resource space offers only two kinds of metrics for each VM: MIPS and RAM by CA/CloudSim API. Since the level of bandwidth remains unchanged, the choice of bandwidth for clustering wouldn't be effective. K able to equal to 3 is not sensible either, as there would be two-dimensional space divided into three partitions without sufficiently developed clusters.

In order to confirm this opinion, the results received were compared with average silhouette coefficient of 110 VMs and values of $\bar{s}$ for $K \in \{2, 3, 4, 5\}$ across the 110-VM dataset (Table 4). he WCSS elbow can also be observed at K value of 2 confirming two clusters give the best balance of compactness and separation for this dataset.

The issues concerning the homogeneity of the database should also be mentioned — all 110 VMs have an equal amount of RAM (8 GB) and only two values of MIPS: either 1,280,000 or 1,371,429. Thus it could be said that $K = 2$ clustering equals determination of MIPS. The algorithm presented is designed to work with heterogeneous VMs and uses different MIPS and RAM values for its operation.

The K-Means algorithm operates as follows, depicted in Figure 9,

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

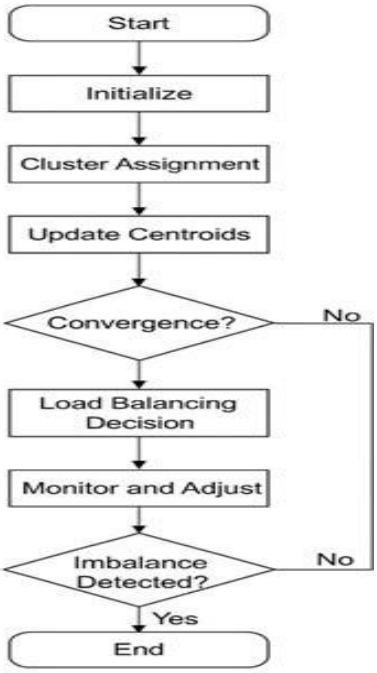


Figure 9: Flowchart of K-Means Algorithm

Algorithm 1 K-Means LB for CA

Require: VM list $V = \{v_1, v_2, \ldots, v_n\}$, each v_i with attributes $mips_i, ram_i$; $K = 2$; $I_{\max} = 10$
Ensure: Cluster assignment $c_i \in \{0, 1\}$ for each VM v_i

Randomly select K distinct VMs as initial centroids μ_1, μ_2
$iter \leftarrow 0$; $changed \leftarrow true$
while $changed$ **and** $iter < I_{\max}$ **do**
  $changed \leftarrow false$
  for each VM $v_i \in V$ **do**
    $d_k \leftarrow \sqrt{(mips_i - \mu_{k,mips})^2 + (ram_i - \mu_{k,ram})^2} \quad (k \in \{1, 2\})$
    $c_i^{\text{new}} \leftarrow \arg\min_k d_k$
    if $c_i^{\text{new}} \neq c_i$ **then**
      $c_i \leftarrow c_i^{\text{new}}$; $changed \leftarrow true$
    end if
  end for
  for $k \in \{1, 2\}$ **do**
    $\mu_{k,mips} \leftarrow \frac{1}{|S_k|} \sum_{v_i \in S_k} mips_i$
    $\mu_{k,ram} \leftarrow \frac{1}{|S_k|} \sum_{v_i \in S_k} ram_i$
  end for
  $iter \leftarrow iter + 1$
end while
// **Load-balancing decision for each incoming cloudlet** q**:**
$k^* \leftarrow \arg\min_k U_k$ where $U_k = \frac{1}{|S_k|} \sum_{v_i \in S_k} load(v_i)$
Assign q to $v^* = \arg\min_{v_i \in S_{k^*}} load(v_i)$

Choose the number of clusters (K):
Select the desired number of clusters K = 2 (MIPS and RAM) for the dataset.

Initialize centroids:
Shuffle the dataset and pick K distinct points as initial centroids – here, drawn from the MIPS and RAM profiles of VMs across the configured data center regions. Repeat until centroids stop

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

changing:

Repeat until convergence (no change in centroids):

- **Step 1: Calculate distances:** Compute squared Euclidean distances of all data points from each centroid. In the LB setting, this will involve checking how close every incoming request is to a resource profile of each cluster of VMs.
- **Step 2: Assign data points to clusters:** Assign each data point to its nearest centroid.
- **Step 3: Update centroids:** Recompute each centroid as the mean of all points assigned to it.

Termination:
The algorithm stops when centroids no longer shift, i.e., cluster assignments have stabilized. K-Means minimizes the squared error function:

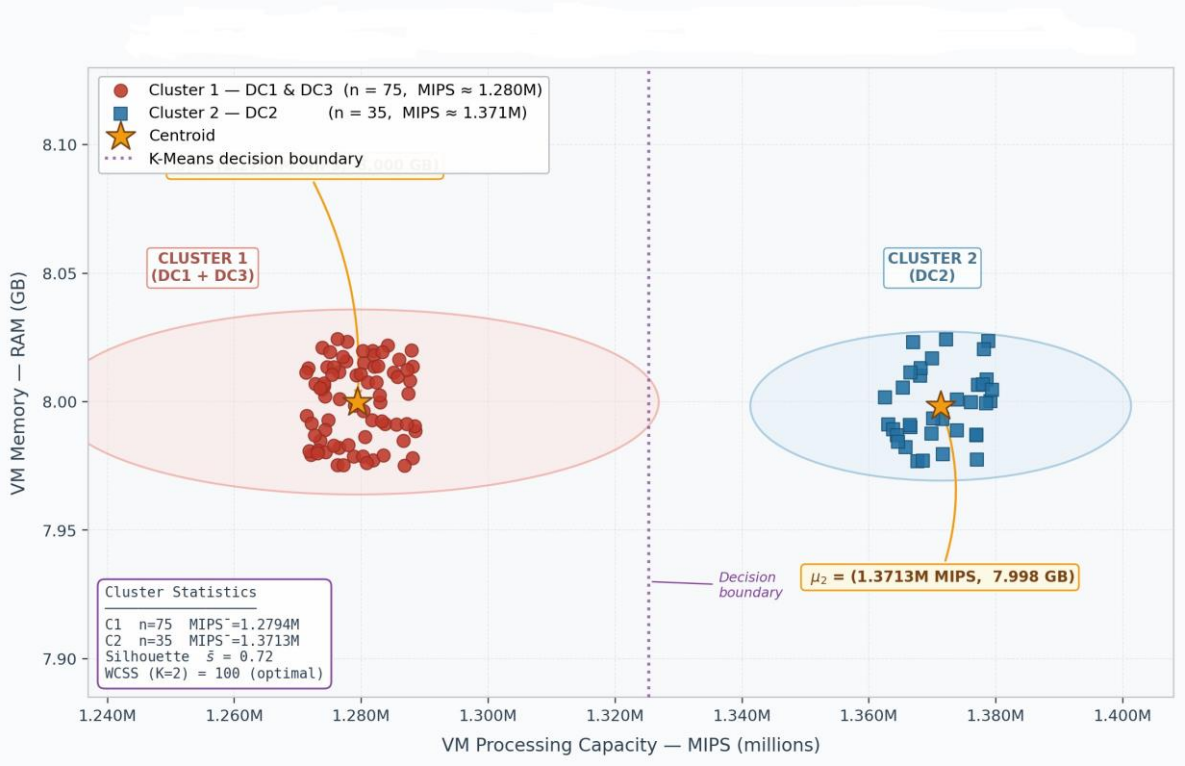
$$E = \sum_{c=1}^{N_c} \sum_{P \in S_C} \|p - m_c\|^2 \quad (1)$$

- N_c = Number of clusters (K = 2 in our method: RAM and MIPS)
- S_c = Set of data points in the c^{th} cluster
- m_c = Centroid of the c^{th} cluster
- $\|p - m_c\|^2$ = Squared Euclidean distance between a point p and its centroid

In the CC context, span time (makespan) refers to the total elapsed simulation time from the first cloudlet submission to the completion of the last cloudlet across all data centers and virtual machines.

Formal Mapping: Cluster Assignments to VM Selection
After convergence, each VM v_i belongs to one of two clusters S_1 or S_2. When a new cloudlet q arrives at a data center, the scheduling decision proceeds as follows:

1. **Cluster selection:** Compute the mean current utilisation of each cluster: $U_k = \frac{1}{|S_k|} \sum_{v \in S_k}$ load (v_i), where load (v_i) is the number of cloudlets currently executing on v_i. Select the cluster with lower utilisation: $k^* = \arg\min_k U_k$.
2. **VM selection:** Within S_k*, select the VM with the lowest current load: $v^* = \arg\min_{v_i \in S_k*}$ load (v_i).
3. **Assignment:** Dispatch cloudlet q to v^*.



INTERNATIONAL JOURNAL OF ADVANCED RESEARCH
International Peer-Reviewed Research Journal
ISSN (Online): 3106-7980 | ISSN (Print): 3106-7972
Volume 3 • Issue 2 • June 2026

Figure 10: Scatter plot of 110 VMs partitioned into two clusters by K-Means (K = 2, features: MIPS and RAM). Red circles denote Cluster 1 (DC1 and DC3 VMs: 1,280,000 MIPS, 8 GB RAM — 75 VMs); blue squares denote Cluster 2 (DC2 VMs: 1,371,429 MIPS, 8 GB RAM — 35 VMs). Cluster centroids (×) lie at μ_1 = (1.280M, 8.00 GB) and μ_2 = (1.371M, 8.00 GB). Separation is driven by the MIPS dimension; RAM is uniform across all VMs.

This two-level selection ensures that (a) the cluster of VMs best suited to the current work-load intensity is chosen first (based on MIPS/RAM profile similarity), and (b) within that cluster, the least-loaded VM is chosen (preventing hot-spotting). This two-level mechanism is the key distinction from RR (which ignores both resource profile and current load) and ESCEL (which tracks load but ignores resource profile similarity).

To implement K-Means in CA, we modified the following source files:

1. Export the CA archive into Eclipse.
2. Create the K-Means algorithm class under cloudsim.ext.datacenter.
3. Add new string constants in Constant.java under cloudsim.ext.
4. Register the K-Means LB policy in ConfigureSimulationPanel.javaunder cloudsim.ext.gui.screen.
5. Enable K-Means simulation support in DatacenterController.java under cloudsim.ext.datacenter.

The algorithm iteratively optimizes this function by refining cluster assignments and centroid positions. The CA Simulator was used for the experimental evaluation. Figure 11 shows the three DCs connected with a user base (UB) of six different regions.

Simulation Configuration and Results

In our simulation, each UB represents a large population of users within a specific geographical region. This approach allows for realistic scalability and performance evaluation.

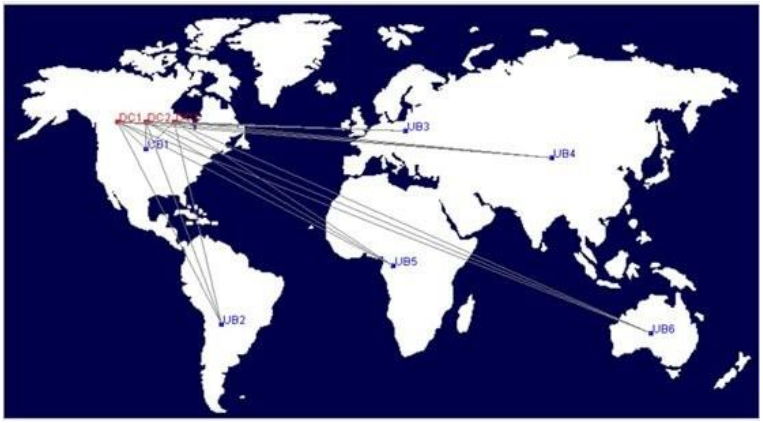


Figure 11: Data Centers connect with 6 UBs

Simulation Environment

The simulation platform was CA, an open-source toolkit built atop CloudSim 4.0, providing a graphical interface for configuring user bases, data centers, and SBP. The K-Means algorithm was implemented by modifying five Java source files within the CA project. All the hardware and software requirements are mentioned in table 5

Table 5: Development and simulation environment.

Component	Specification
Processor	Intel Core i3, 3rd Generation
RAM	8 GB
IDE	Eclipse IDE (eclipse-09-R-win32-x86_64)
Simulation platform	CA (built on CloudSim 4.0)
Implementation lang.	Java (SE 8)
OS	Windows 10 (64-bit)

2.4.1 DataSet

Table 6 shows the UB configuration used in this experiment. Six UBs were configured with input attributes representing the dataset of the simulated application. Each UB is further divided by timezone (GMT), where the load of users is considered in millions for dedicated time periods as detailed in Table 7.

Table 6: Geographically User Base Configuration.

UB	Region	Region ID	Peak Hour Start(GMT)	Peak Hour End(GMT)	Average Peak Users	Average Off Peak Users
UB1	North America	0	13	15	4000000	400000
UB2	South America	1	15	17	1000000	100000
UB3	Europe	2	20	22	3000000	300000
UB4	Asia	3	1	3	1500000	150000
UB5	Africa	4	21	23	500000	50000
UB6	Oceania	5	7	10	90000	9000

2.4.2 Parameters Selection

The architecture used in our simulation is x86 because most cloud servers still use this architecture with Linux as an operating system. For creation and operation of VMs, we used VMM (Xen Hypervisor).

Each data center (DC1, DC2, and DC3) has multiple physical servers to host VMs. Each server has 256 GB of RAM and a total of 100 TB of storage. High memory allows each machine to efficiently host multiple VMs, and large storage capacity ensures enough space for user requests. Each DC server has 10 Gb of bandwidth, which allows many simultaneous user connections and reduces latency. Each server has 32 CPU cores (100,000 MIPS per core) to support high parallelism and computational performance.

Table 7: Simulation Attribute Values

Parameter	Value
VM Image Size	1TB
VM Memory	8GB
VM Bandwidth	100MB
DC1-No of VMs	50
DC2-No of VMs	35
DC3-No of VMs	25
Data Center – Architecture	x86
Data Center – OS	Linux
Data Center – VMM	Xen
Data Center – Number of Machines	DC1-20, DC2-15 & DC3-10

Data Center – Memory per Machine	256GB
Data Center – Storage per machine	100TB
Data Center – Available BW per Machine	10GB
Data Center – Number of processors per machine	32
Data Center – Processor speed	100000MIPs
Data Center – VM Policy	Time Shared
User Grouping Factor	10000
Request Grouping Factor	2500
Executable Instruction Length	1000
Simulation Duration	1Day

In CA, VM Image Size refers to the amount of storage allocated to the VM's disk image—essentially, the virtual hard drive used by the VM. As mentioned earlier, we have three DCs: DC1 contains 50 VMs, DC2 contains 35 VMs, and DC3 contains 25 VMs. DC1 is capable of handling a higher number of user requests compared to DC2 and DC3. Each VM has 8 GB of RAM and a bandwidth of 100 MB.

The simulation duration is 24 hours to cover all UBs in different timezones. Instead of simulating every user separately, CA groups them for efficiency, which also speeds up simulation. Each group represents 10,000 real users who send requests to the DC. For example, 1M users ÷ 10,000 (group factor) = 100 groups. The request group factor is set to 2,500 to reduce communication overhead between UBs and DCs, as multiple requests are grouped into one batch instead of processing each request separately. The Executable Instruction Length of 1,000 represents the computational size of each request in terms of the number of instructions (MIPS) that must be executed by the VM, as shown in Table 7.

2.4.3 Experimental Results

Table 6 specifies the UB configurations, including Location, Peak Hours, and User Count. The performance of three SBPs—CDC, ORT, and DR—was evaluated using four LB algorithms (RR, ESCEL, Throttled, and K-Means). The results are presented in Tables 8, 9, and 10, which compare the Average RT against the Average DCPT for each policy.

Table 8: Four LB techniques compared using CDC SBP

SBP	LB Policy over VMs	Avg. RT (ms)	Min RT (ms)	Max RT (ms)	Avg. DCPT (ms)	Min DCPT (ms)	Max DCPT (ms)
Closest DC SBP	RR	405.68	67.36	1256.46	67.21	0.83	536.49
	ESCEL	409.86	66.52	1248.55	71.49	0.83	567.94
	Throttled	397.05	65.50	1268.93	48.01	0.83	451.38
	K-Means	396.64	65.50	1253.29	47.68	0.83	498.03

Table 9: Four LB techniques compared using ORT SBP

SBP	LB Policy over VMs	Avg. RT (ms)	Min RT (ms)	Max RT (ms)	Avg. DCPT (ms)	Min DCPT (ms)	Max DCPT (ms)
Optimize SBP	RR	415.47	64.10	1340.48	77.81	1.24	1115.73
	ESCEL	420.93	64.10	1246.29	83.74	1.34	994.20
	Throttled	402.26	64.10	1330.54	54.35	1.21	1022.56
	K-Means	400.99	64.10	1357.47	52.87	1.34	1049.26

Table 10: Four LB techniques compared using DR SBP

SBP	LB Policy over VMs	Avg. RT (ms)	Min RT (ms)	Max RT (ms)	Avg. DCPT (ms)	Min DCPT (ms)	Max DCPT (ms)
Reconfigure Dynamically SBP	RR	2394.31	65.50	186274.16	2075.59	0.83	186207.24
	ESCEL	842.71	65.50	114173.09	501.37	0.83	114101.98
	Throttled	519.45	65.50	159157.64	178.28	0.83	159089.83
	K-Means	504.27	65.50	90164.51	162.43	0.83	90103.00

2.4.4 Discussion and Analysis

Figure 12 presents the comparative results of average RT and DCPT under the CDC SBP. Since CDC routes every request to the nearest data center, network transit time is minimised and performance differences reflect VM-level scheduling alone.

RT Efficiency using CDC as SBP:

- K-Means outperforms RR by 2.23%
- K-Means shows 3.22% better efficiency than ESCEL
- K-Means and Throttled are effectively equivalent (0.10% difference)

DCPT Efficiency using CDC as SBP:

- 29.06% more efficient than RR
- 33.30% superior to ESCEL
- 0.69% lower than Throttled (47.68 ms vs. 48.01 ms)

Under the CDC, both K-Means and Throttled yield similar DCPTs with only 0.33 ms difference. Since both approaches surpass RR and ESCEL in terms of performance, the fact that the two algorithms yield similar results is understandable considering that Closest DC strategy works in a deterministic fashion, directing requests to one and only one data center. As a conse-quence, there is little room for cluster-oriented routing mechanisms to affect the outcome, which makes these two algorithms practically identical in results. In the case of K-Means, it achieves slightly more balanced utilization of VM resources via centroid-based allocation of requests, while the performance of Throttled is the same. Both strategies substantially reduce DCPT relative to RR (which ignores load entirely) and ESCEL (which tracks request counts but not resource capacity).

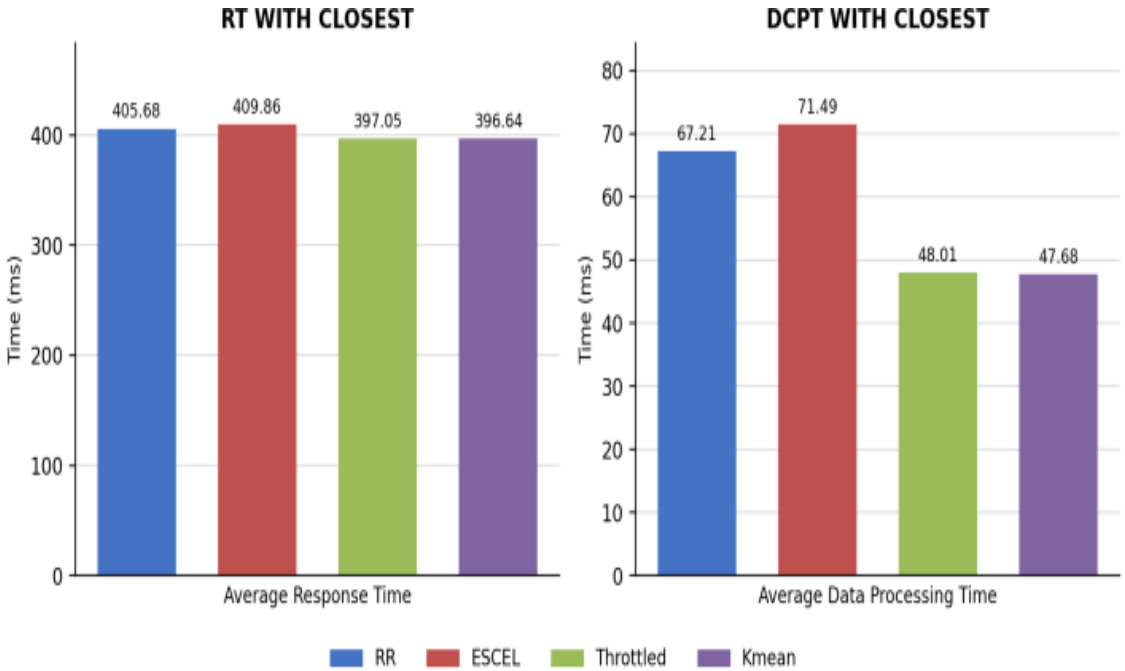


Figure 12: Average RT and DCPT with Closest SBP

Figure 13 presents the comparative results of average RT and DCPT under the ORT SBP. As demonstrated in Table 9:

RT Efficiency Using Optimize RT as SBP:

- K-Means outperforms RR by 3.49%
- K-Means shows 4.74% better efficiency than ESCEL
- K-Means is 0.32% more efficient than Throttled (400.99 ms vs. 402.26 ms)

RT vs. DCPT Asymmetry. The consistently larger improvement in DCPT compared to RT can be explained by the additive decomposition of RT:

$$RT = DCPT + NT \quad (2)$$

where NT denotes network transit time between the UB and the assigned DC. K-Means opti-mizes the VM selection step inside the DC, directly reducing DCPT. However, NT is governed by geographical distance and routing policy, both of which are independent of the LB algorithm. Under the CDC policy, NT is the dominant component of RT (since requests are always routed to the nearest DC, minimising but not eliminating propagation delay). Therefore, even a large DCPT reduction yields a modest RT improvement. Under the DR policy, the data center assign-ment itself changes dynamically, allowing K-Means to influence both components, which explains the much larger RT improvement (78.94%) observed under DR compared to CDC (2.23%).

DCPT Efficiency Using Optimize RT as SBP:

- 32.05% more efficient than RR
- 36.87% superior to ESCEL
- 2.72% more efficient than Throttled (52.87 ms vs. 54.35 ms)

Under ORT, K-Means marginally outperforms Throttled in both RT and DCPT. The ORT broker dynamically selects the best-responding data center per request, which reduces deter-minism in routing. Under these conditions, K-Means's cluster-selection step provides a small additional advantage on top of the broker-level optimisation already in place, while Throttled's fixed per-VM concurrency cap does not adapt to which VMs currently hold the more favourable resource profile. Both algorithms remain substantially superior to RR and ESCEL.

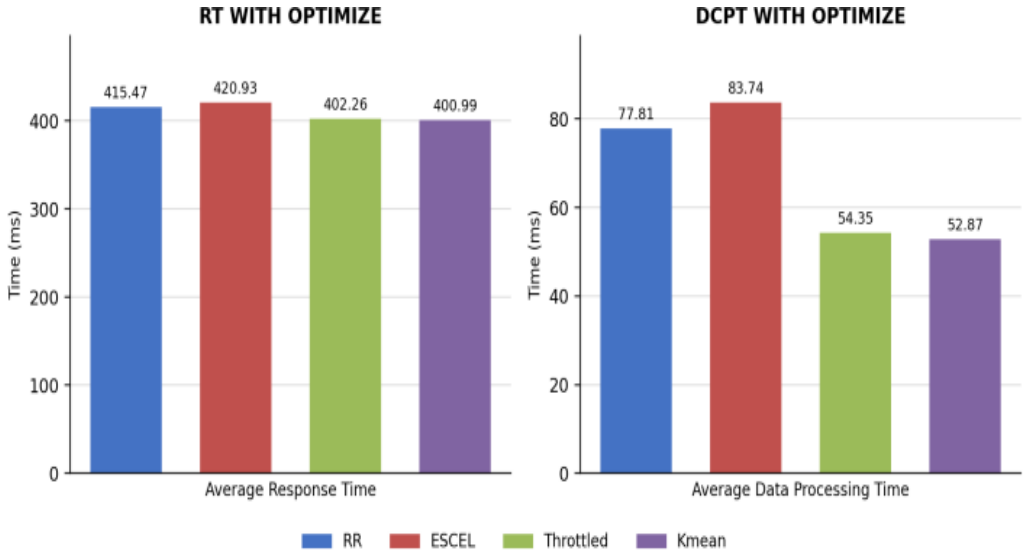


Figure 13: Average RT and DCPT with ORT SBP

Figure 14 presents the comparative results of average RT and DCPT under the DR SBP. As demonstrated in Table 10:

RT Efficiency Using DR as SBP:

- K-Means outperforms RR by 78.94%
- K-Means shows 40.16% better efficiency than ESCEL
- K-Means is 2.92% more efficient than

Throttled (504.27 ms vs. 519.45 ms)

DCPT Efficiency Using DR as SBP:

- 92.17% more efficient than RR
- 67.61% superior to ESCEL
- 8.89% more efficient than Throttled (162.43 ms vs. 178.28 ms)

Under DR, K-Means achieves lower DCPT than Throttled by 15.85 ms (8.89%). The DR pol-icy continuously reassigns VMs across data centers in response to load changes, creating frequent periods of VM unavailability during migration. K-Means's cluster-availability check, which re-computes mean utilisation across both resource clusters at every scheduling decision, adapts to these reassignment events more responsively than Throttled's fixed per-VM concurrency cap, which only reacts once an individual VM's queue is already full. Both K-Means and Throttled substantially outperform RR (DCPT reduction of 92.17% and 91.41% respectively) and ESCEL (67.61% and 64.44%), confirming that any capacity-aware dispatch mechanism provides a large advantage over stateless schedulers under dynamic VM reassignment.

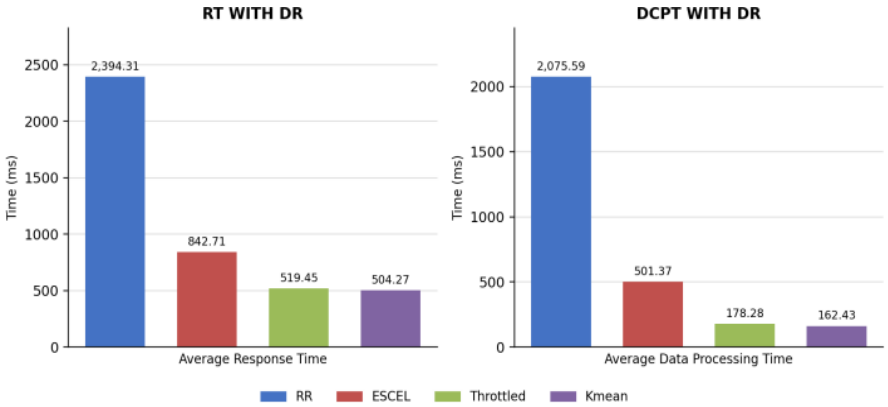


Figure 14: Average RT and DCPT with DR SBP

Maximum RT Tail Latency Under DR. All four algorithms produce elevated tail latencies under DR due to dynamic VM reassignment. The measured maximum RTs are: RR 186,274 ms, Throttled 159,158 ms, ESCEL 114,173 ms, and K-Means 90,165 ms. K-Means achieves the low-est tail latency because its cluster-utilisation check is recomputed at every scheduling decision, letting it steer new cloudlets toward the lower-utilisation resource cluster as soon as a reassign-ment event begins, rather than waiting for an individual VM to fill up. Throttled's fixed per-VM concurrency cap still limits unbounded queue growth relative to RR and ESCEL, but a capped VM can remain the only apparently-available VM in its neighbourhood during a reassignment burst, so cloudlets continue to queue behind it instead of being redirected to a less-loaded cluster. RR and ESCEL maintain allocation counters that are not updated atomically during migration, so requests continue to queue on temporarily unavailable VMs, producing the two highest tail latencies. This explains why K-Means achieves both the lowest average DCPT and the lowest max RT of the four algorithms under the DR policy.

2.4.5 Result Stability and Run-to-Run Variance

RR, ESCEL, and Throttled belong to a category of fully deterministic algorithms where the results are the same for each successive simulation run. In K-Means, the only stochastic factor is random initialization of centroids. To assess the effect of this stochasticity in practice, two runs of K-Means DR simulation were performed using independent seeds (Run 1 and Run 2):

- Run 1: Avg. RT = 504.27 ms, Avg. DCPT = 162.43 ms
- Run 2: Avg. RT = 509.13 ms, Avg. DCPT = 166.99 ms

The variation between both runs is negligible (4.86ms in RT and 4.56ms in DCPT), thus demonstrating that strict determinism is not guaranteed in case of DR SBP, though clustering

is effectively steady. In the case of the DR algorithm, VMs are being dynamically assigned during simulation and some timing-dependent variations are introduced; this has some bearing on the differences between successive runs, even though the cluster allocation remains the same. Table 11 provides an average and standard deviation for K-Means for both runs, along with a set of deterministic measures.

2.4.6 Scalability Analysis

To evaluate the effectiveness of both algorithms at different levels of infrastructure and load, three more CA tests were monitored using DR SBP. S2 represents the reduction in all its configurations

Table 11: RT and DCPT stability under DR SBP (ms)

Algorithm	Avg. RT (ms)	Std. RT (ms)	Avg. DCPT (ms)	Std. DCPT (ms)
RR	2394.31	0.00	2075.59	0.00
ESCEL	842.71	0.00	501.37	0.00
Throttled	519.45	0.00	178.28	0.00
K-Means	506.70	2.43	164.71	2.28

to 2 DCs and 50 VMs.The following table Table 12 reports reports measured average RT and DCPT for both algorithms, alongside the DCPT difference (positive = K-Means better; negative = K-Means worse).

Table 12: Scalability results under DR SBP (ms)

Scenario	DCs	VMs	Avg. RT (ms)		Avg. DCPT (ms)	
			Throttled	K-Means	Throttled	K-Means
S1 (baseline, 3 DCs)	3	110	519.45	504.27	178.28	162.43
S2 (2 DCs)	2	50	710.98	420.19	381.16	87.32
S3 (4 DCs)	4	200	8872.00	2179.20	8367.85	1929.48
S4 (bursty UB1)	3	110	1011.34	1009.88	419.09	417.28

The results reveal that K-Means's advantage over Throttled holds across most of the deploy-ment scales tested, though its magnitude varies with configuration. In S2 (small deployment, 2 DCs, 50 VMs), K-Means substantially outperforms Throttled in DCPT (87.32 ms vs. 381.16 ms), a 77.09% reduction. With only 50 VMs across 2 DCs, Throttled's fixed per-VM concurrency cap saturates quickly, since fewer VMs share the same threshold and cloudlets queue at the broker as soon as each VM's cap is reached; K-Means's cluster-level utilisation check continues to iden-tify the less-loaded of the two resource clusters even at this reduced scale, redistributing load before individual VMs saturate. In S3 (large deployment, 4 DCs, 200 VMs), the advantage is even larger: K-Means achieves a DCPT of 1,929.48 ms against Throttled's 8,367.85 ms, a 76.95% reduction. At this scale, DC4 bears a disproportionate share of traffic under Throttled (avg. DCPT 35,279.98 ms), while K-Means's cluster-aware routing distributes load more evenly across the expanded VM pool. In S4 (bursty load), the two algorithms converge: K-Means DCPT is 417.28 ms vs. Throttled's 419.09 ms (0.43% difference). When peak demand saturates the full VM pool, dispatch strategy has negligible influence on outcomes. Taken together, the scalabil-ity results indicate that K-Means's cluster-aware dispatch retains an advantage over Throttled's concurrency cap from small (S2) through large (S3) deployments, converging with Throttled only under saturation conditions (S4).

2.4.7 Cost Analysis

CA reports three cost dimensions: VM processing cost, data transfer cost, and total data-center cost. Table 13 compares these across all four LB algorithms under the DR SBP—the scenario with the largest performance differentials—in USD over a 24-hour simulation.

$$\text{VM Cost}_{\text{DC}_i} = n_i \times 24 \times 0.10 \tag{3}$$

$$n_i = \frac{\text{VM Cost}_{\text{DC}_i}}{24 \times 0.10} \tag{4}$$

$$\text{VM Cost}_{\text{total}} = \sum_{i=1}^{3} n_i \times 24 \times 0.10 = N \times 24 \times 0.10 \tag{5}$$

Table 13: Cost comparison under DR SBP(USD).

Algorithm-3 DCs	Total VM Cost ($)	Total Data Transfer Cost ($)	Total Grand Cost ($)
RR	713.51	4780.18	5493.70
ESCEL	714.30	4780.18	5494.49
Throttled	713.76	4780.18	5493.95
K-Means	713.08	4780.18	5493.26

CA bills for VM uptime, not utilization, so LB algorithm choice has negligible impact on cost. Under DR SBP, data transfer accounts for roughly 87% of total cost. All four algorithms produce identical data transfer cost ($4780.18), confirming that the scheduling algorithm does not affect data volume. VM cost varies by less than $1.22 across all four algorithms. These differences are not operationally significant.

2.4.8 K-Means Clustering Overhead

A potential concern with any clustering-based scheduler is that the computation required to assign tasks to clusters may offset the DCPT gains. Table 14 provides an analytical estimate of scheduling overhead per call, derived from the algorithm's time complexity $O(n \cdot K \cdot I) = O(110 \times 2 \times 10) = O(2{,}200)$ operations at a conservative JVM throughput of 10^8 operations per second.

Table 14: K-Means scheduling overhead estimate (ms)

Metric	Value
Mean overhead per scheduling call	0.0220 ms
Max overhead per scheduling call	0.0660 ms
Total overhead over 24-hour simulation (1,009 calls)	22.20 ms
Average DCPT saving per request (vs. RR, DR)	1913.16 ms
Overhead as percentage of DCPT saving	0.0012%

The estimated mean overhead per scheduling decision is several orders of magnitude smaller than the average DCPT saving, confirming that the clustering computation does not erode the performance gains. K-Means runs in $O(n \cdot K \cdot I)$ where $n = 110$ VMs, $K = 2$ clusters, and $I \leq 10$ iterations, so overhead stays negligible as the VM pool grows.

CONCLUSION

This study compared four LB algorithms — RR, ESCEL, Throttled, and K-Means — across three SBPs: CDC, ORT, and DR, using the Cloud Analyst simulator with 110 VMs distributed across three data centers.K-Means consistently outperformed RR and ESCEL across all SBPs, and matched or slightly exceeded Throttled in every configuration tested. Under CDC, K-Means reduced DCPT by 29.06% compared to RR, with performance essentially equivalent to

Throttled (0.69% difference; 47.68 ms vs. 48.01 ms). Under ORT, K-Means reduced DCPT by 32.05% compared to RR and was 2.72% more efficient than Throttled (52.87 ms vs. 54.35 ms). The most pronounced improvements were under DR: K-Means reduced DCPT by 92.17% compared to RR and 67.61% compared to ESCEL, and was 8.89% more efficient than Throttled (162.43 ms vs. 178.28 ms), while also achieving the lowest tail latency of the four algorithms (max RT 90,165 ms). Scalability experiments confirm that K-Means's advantage over Throttled holds across deployment scales, from small (2 DCs, 50 VMs: 77.09% DCPT reduction) to large (4 DCs, 200 VMs: 76.95% DCPT reduction) configurations, converging with Throttled only under saturation conditions (bursty single-region load).These results clarify the conditions under which resource-profile-aware clustering offers scheduling advantages: K-Means's two-level cluster-then-VM selection adapts more responsively to shifting load and VM availability than Throttled's fixed per-VM concurrency cap, yielding an advantage that persists across static and dynamic broker policies alike, and across both small and large deployments. Future work will investigate adaptive K-value selection, heterogeneous VM configurations to further characterise the clustering advantage, and validation on real cloud infrastructure.

Data Availability

The simulation configuration files and CA project archive used in this study are available from the corresponding author upon reasonable request. The CA simulator is freely available at https://https://github.com/Irfanarain01/kmeans-cloudanalyst-load-balancing/tree/main

REFERENCES

[1] Y. Kang and Y. B. Park. The performance evaluation of k-means by two mapreduce frame-works, hadoop vs. twister. In 2015 International Conference on Information Networking (ICOIN), pages 367–370, Cambodia, 2015.

[2] Tanvi Gupta and Supriya P Panda. A comparison of k-means clustering algorithm and clara clustering algorithm on iris dataset. International Journal of Engineering & Technology, 7(4):4766–4768, 2018.

[3] M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud, and S. Musa. A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access, 8:130500–130526, 2020.

[4] P. P. Patel and R. Gupta. Static and dynamic load balancing in cloud computing: A comparative analysis. International Journal of Cloud Computing, 2024.

[5] N. Abd Alnabe and S. R. Zeebaree. Distributed systems for real-time computing in cloud en-vironment: A review of low-latency and time sensitive applications. The Indonesian Journal of Computer Science, 13(2), 2024.

[6] R. R. Aditiya. Simulation of load balancing algorithms using cloud analyst for distinct regions IT resources. International Journal of Computer Science and Mobile Computing, 2022.

[7] D. A. Shafiq, N. Z. Jhanjhi, and A. Abdullah. Load balancing techniques in cloud computing environment: A review. Journal of King Saud University - Computer and Information Sciences, 34(7):3910–3933, 2022.

[8] E. J. Ghomi, A. M. Rahmani, and N. N. Qader. Load-balancing algorithms in cloud com-puting: A survey. Journal of Network and Computer Applications, 88:50–71, 2017.

[9] S. H. Chaflekar, R. Rewatkar, and K. T. V. Reddy. Load balancing in cloud environment: Issues and challenges. In AIP Conference Proceedings, volume 3162, page 020053, 2025.

[10] S. Sharma and P. Bedi. Enhanced throttled load balancing algorithm for cloud computing. International Journal of Advanced Computer Science and Applications, 15(3), 2024.

[11] H. R. Panuganti and R. Subramanian. Enhanced throttled load balancing for virtual ma-chine allocation in multiple data centers. Scalable Computing: Practice and Experience, 25(5):3453–3467, 2024.

[12] Y. Rashid and C. I. Nakpih. Ret-elba: A novel algorithm for efficient load balancing in cloud computing. Asian Journal of Research in Computer Science, 17(5):56–64, 2024.

[13] N. Susitha and G. Monica. Esce–an optimized load balancing technique to reduce data center and vm cost in cloud environment. Indian Journal of Natural Sciences, 12(5):66–74, 2023.

[14] M. Deep, A. Ghosh, and K. Acharjee. Performance analysis of load balancing algorithms using cloud analyst. YMER International Journal, 2024.

[15] M. K. Kumar. Enhancing load balancing in cloud computing using a hybrid ant earthworm optimization algorithm. International Journal of Intelligent Systems and Applications in Engineering, 12(3):3555–3560, 2024.

[16] L. S. Umrao and C. M. Mishra. Hybrid red fox and remora optimization algorithm expedites optimal load balancing in cloud computing. Preprint, Research Square, 2024. Version 1, December 16.

[17] P. Archana, S. K. Medishetti, S. K. Manchala, and J. Sreedhar. Roa: Optimizing scheduling time and load balancing in cloud computing environment. In 2024 International Conference on Sustainable Communication Networks and Application (ICSCNA), pages 600–607. IEEE, 2024.

[18] S. K. Hegde, R. Hegde, C. N. Kumar, R. M. R. Raman, and G. M. Jayaseelan. Hybrid adam poa: Hybrid adam pufferfish optimization algorithm based load balancing in cloud computing. SN Computer Science, 6(2), 2025.

[19] D. B. Bhavsar, P. P. Patel, S. Syal, S. U. Ahsaan, B. M. Krishna, and A. Kotwal. Power-based single threshold load balancing algorithm for cloud computing. International Journal of Communication Networks and Information Security, 16(3):277–287, 2024.

[20] N. Singh, S. Kumar, and A. Sharma. Threshold-based load balancing strategies in cloud environments. In Proceedings of the International Conference on Cloud Computing, pages 112–118, 2024.

[21] R. A. Hasan, M. M. Akawee, E. F. Aziz, O. A. Hammood, A. S. Showket, H. Jasim, M.

A. Alkhafaji, O. K. Ahmed, and K. Yasin. Hybrid spotted hyena based load balancing algorithm (HSHLB). Mesopotamian Journal of Big Data, pages 199–210, 2024.

[22] A. Alwhelat, O. A. Fadare, F. Al-Turjman, and L. Ibrahim. Adaptive load balancing in cloud computing using deep deterministic policy gradient (ddpg): A reinforcement learning approach. In F. Al-Turjman, editor, Smart Infrastructures in the IoT Era, Sustainable Civil Infrastructures. Springer, Cham, 2025.

[23] S. Sonal. Optimized load balancing techniques in cloud computing environment: A system-atic literature review and future trends. Towards Excellence, 16(3), 2024.

[24] N. R. Sabat and R. R. Sahoo. Virtual iga-gru an approach for load balancing in cloud computing. In Computing, Communication and Intelligence, pages 71–75. CRC Press, 2025.

[25] A. A. Motlagh, A. Movaghar, and A. M. Rahmani. Task scheduling mechanisms in cloud computing: A systematic review. International Journal of Communication Systems, 33(6):e4302, 2020.

[26] S. K. Mishra, B. Sahoo, and P. P. Parida. Load balancing in cloud computing: a big picture. Journal of King Saud University-Computer and Information Sciences, 32(2):149–158, 2020.

[27] A. A. Alkhatib, A. Alsabbagh, R. Maraqa, and S. Alzubi. Load balancing techniques in cloud computing: Extensive review. Advances in Science, Technology and Engineering Systems Journal, 6(2):860–870, 2021.

[28] G. Lokesh and K. K. Baseer. A meta-heuristic approach-aided multi-objective strategy with optimal resource allocation via fault tolerant and priority-based scheduling for load balancing in cloud. Wireless Networks, 2024.

International Journal of Advanced Research