Ingénierie des Systèmes d'Information
Vol. 31, No. 8, August, 2026, pp. 2531-2547
Journal homepage: http://iieta.org/journals/isi

An Adaptive Dual-Layer K-Means++ Scheduling Framework for Priority-Aware Cloudlet–VM Allocation in Heterogeneous Cloud Environments

Mohanad Yahya Al-Hamami*, Mohsen Nickray

Department of Computer and Information Technology, Faculty of Engineering, University of Qom, Qom 3716146611, Iran

Corresponding Author Email: mohanad.alhammami@uokufa.edu.iq

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/isi.310802

Received: 5 April 2026
Revised: 25 June 2026
Accepted: 10 July 2026
Available online: 31 August 2026

Keywords:
cloud computing, cloudlet scheduling, virtual machine allocation, load balancing, K-means++ clustering, resource management, heterogeneous cloud environments

ABSTRACT

Cloud computing (CC) environments increasingly face challenges in workload imbalance and resource contention due to heterogeneous virtual machines (VMs) and dynamically changing task demands. Existing scheduling approaches often perform task clustering or resource clustering separately, which limits adaptive task–resource matching and may reduce resource utilization efficiency. This study proposes an adaptive dual-layer K-means++ scheduling framework for priority-aware cloudlet–VM allocation in heterogeneous cloud environments. The proposed framework integrates cloudlet prioritization, dynamic VM resource clustering, level-based matching, and runtime fallback allocation within a unified scheduling architecture. Cloudlets are first categorized into high-, medium-, and low-priority groups according to task length, while VMs are dynamically clustered according to normalized Central Processing Unit (CPU) and Random Access Memory (RAM) resource weights. A Utilization-Level Comparator (ULC) is further introduced to select suitable VMs within each resource group, enabling adaptive scheduling under changing workload conditions. The proposed framework is evaluated using CloudSim simulations with different VM configurations (8, 16, and 32 VMs) and workload sizes ranging from 50 to 1000 cloudlets. Experimental results demonstrate that the proposed approach achieves substantial reductions in average makespan compared with K-means-based HEFT and Efficient K-means scheduling methods. In particular, Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets (PAKS-VMC) reduces makespan by 85.46%–91.90% compared with Kmean HEFT (KmeanH) and by 91.76%–94.78% compared with Efficient Kmean (Ekmean) under the tested 16-VM configuration. These findings indicate that the proposed framework provides an efficient and adaptive solution for workload management and resource allocation (RA) in heterogeneous cloud computing environments.

1. INTRODUCTION

Cloud computing (CC) is a computing model that enables customers to run various computing activities using virtualized resources from cloud data centers (CDCs). As internet technologies advanced and reliance on CC systems increased, there was a need for smarter, more dynamic resource and task management mechanisms to achieve effective load balancing (LB) and quality of service (QoS) [1-3]. The computing model CC provides storage resources, as well as computing devices and applications, to extend the ability to scale distributed computing jobs and task execution. Provides three types of cloud services to the CC concept: infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). The most commonly used of these is the IaaS model, which provides processing and memory capacity under a Service Level Agreement (SLA) that guarantees specific quality and configuration parameters [4, 5]. One of the most apparent trends in the global CC market is the strategy of building a robust, scalable computing environment to run complex tasks using the IaaS model. Under the IaaS model, an SLA is signed between a cloud provider and customers when a clear system goal and the required configuration are established [6-8]. The various cloud service models are shown in Figure 1.

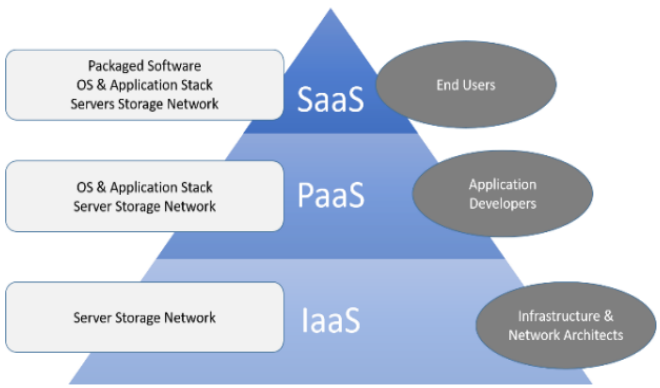

Figure 1. The various models have been adapted into cloud service models [5]

The unbalanced distribution of workload, caused by ineffective work schedules and workload heterogeneity, is a significant issue in this sphere. As the number of users increases and the variance in task sizes (cloudlets) increases, cloud environments often suffer from resource contention, long execution times, and SLA violations. To address these issues, intelligent and dynamic resource allocation (RA) and task scheduling frameworks are needed to achieve efficient LB [9].

The load-balancing strategy enhances overall system performance making user satisfaction, ensuring equal resource use, and preventing overload on any single node. Using effective LB, resources are utilized to the maximum, thereby reducing resource waste [10]. A cloud-based LB process is used to provide equal distribution of dynamically changing workloads to cloud nodes. LB is also useful for ensuring an equitable distribution of dynamic workloads across cloud nodes. Load management is critical to the stability and performance of the system [11]. This efficiency is primarily achieved through intelligent RA and task scheduling mechanisms that rely on policy-based algorithms implemented within the existing infrastructure [12, 13]. There are traditionally three main classes of LB algorithms: static, dynamic, and hybrid. The foundation of static algorithms is a priori information about available system resources and task demands, which can make them perform poorly in dynamic, unpredictable clouds. This type of algorithm has serious issues when system resources are abruptly lost, and system activities are disrupted [14]. It is also ineffective, and users' needs cannot be fulfilled instantly. Examples include (CLBVM, Min-Min, Max-Min, Shortest Job Scheduling, (OLB + LBMM), and Round Robin) [15]. However, dynamic algorithms operate in real time, adapting their decisions based on the system's condition, resource availability, and the nature of the tasks. This method can be used to eradicate the disadvantages of static approaches. One of them is evenly spread current implementation (ESCE), honey bee search, ant colony, and biased random sampling [16]. The hybrid approaches aim to coax the best from both.

Hybrid algorithms are built by combining two algorithms to exploit their advantages. Several authors applied machine learning (ML) techniques by combining load-balancing mechanisms with ML to improve resource management and cloud performance. Examples include:

The authors proposed a hybrid load-balancing method that combines features of two main strategies, namely Equally Spread Current Execution (ESSC) and Double Priority (DP), to create a joint algorithm. This approach reduces response time and improves reliability between users and cloud resources [17]. A scalable hybrid multi-objective load-balancing algorithm (IMH_LB) achieves significant improvements across key QoS metrics in large-scale cloud environments (resource utilization, makespan). It combines the Grey Wolf Optimization (GWO) algorithm with the velocity-driven approach of the Particle Swarm Optimization (PSO) [18]. Researchers have developed a hybrid technique that combines Genetic Algorithms (GAs) with ML algorithms and other traditional techniques, including Round-Robin and Least-Connections. They tested it using a Python simulation, which demonstrated better resource usage and faster task completion [19]. The research team has developed a hybrid scheme that integrates WOA and LOA, along with credit-based and resource-sensitive scheduling. FILL, SPIL, and PEFT dynamically allocate tasks to maximize LB, minimize makespan, and improve efficiency in cloud environments [20]. A hybrid task-scheduling algorithm using a combination of Hill-Climbing and GAs improves the optimal RA in the cloud environment. The algorithm combines local and global search to accelerate convergence, reduce makespan, and improve LB. The Simulation has demonstrated higher execution efficiency and shorter completion time than standalone heuristics [21]. Efficient clustering and hybrid optimization techniques are recent trends that are key to system performance and resource utilization. Comparing the structure of the WOA with that of K-means clustering could help create a better clustering structure and reduce energy consumption in complex deployments, thereby demonstrating the efficiency of adaptive, intelligent clustering processes [22]. A model that combines task offloading based on GKOA with priority scheduling based on fuzzy algorithms (FPTS) could successfully reduce waiting time and energy consumption in MEC environments. The study showed that overall system performance can be enhanced through the use of priority-based scheduling and flexible RA [23]. With the growing number of algorithms for RA and task scheduling to achieve LB in CC networks, modern hybrid algorithms, including Kmean HEFT (KmeanH) and Efficient Kmean (Ekmean) for the RA and LB, still have limitations when applied to heterogeneous virtual environments. Such methods may be challenging to implement in a way that is efficient for clustering virtual machines (VMs), particularly in terms of Central Processing Unit (CPU) and memory resources, or dynamically assigning work to meet changing workload requirements. The resulting impact of such systems is long makespan and execution times, resource underutilization, and load imbalance among the data center nodes, all of which have a negative impact on the overall QoS. From the perspective of information systems engineering, task scheduling and LB in cloud infrastructures are not just algorithmic problems, but also have implications for the performance and reliability of large-scale enterprise and industrial applications that are deployed in or on cloud services with strict SLAs [24]. Modern information systems are experiencing traction in various sectors of the economy, including e-Commerce, online banking, healthcare, and Industrial IoT, where cloud-based systems are being adopted to manage highly variable workloads while maintaining the required QoS [25]. In such a setting, poor scheduling can result in SLA violations, slow responses, and degraded user experience at the application layer [26]. By aligning task priorities with heterogeneous VM capabilities, the proposed Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets (PAKS-VMC) framework aims to provide more predictable and robust QoS for these information systems and to support the reliable execution of business-critical processes across cloud infrastructures. This paper addresses the performance gap in existing dynamic task scheduling and RA approaches, and the need for a more sophisticated, dynamic approach to grouping VMs, assigning tasks, and managing real-time VM resource states within a single runtime system. The paper proposes a framework, PAKS-VMC, to address this gap. The proposed PAKS-VMC framework builds upon our previously published KMPPVM-DRM approach [27], which introduced dynamic K-means++-based VM clustering for RA. While KMPPVM-DRM focused on VM clustering and RA, it did not address coordinated cloudlet clustering, priority-aware task–VM matching, fallback allocation, or an integrated runtime scheduling framework. These research limitations motivated

the development of PAKS-VMC, which extends the previous work through dual-layer clustering, priority-aware level matching, the Utilization-Level Comparator (ULC), runtime adaptive scheduling, and a fallback reallocation mechanism, thereby providing a more comprehensive solution for LB in heterogeneous cloud environments. The proposed framework uses an integrated dual-layer clustering algorithm based on unsupervised K-means++, thereby improving the scheduling process's efficiency. Cloudlets are categorized into low-, medium-, and high-priority lists based on task length, enabling a lightweight, efficient, priority-driven process. Dynamically, the VMs are adaptively partitioned into three resource levels—low, medium, and high —according to their CPU and memory weights. Under this dual-clustering protocol, the VMs of each cluster are then allocated to tasks at the ULC level to balance execution and reduce the makespan.

Accordingly, the study addresses key questions:

•How well can an integrated dual-layer hybrid (using unsupervised K-means++) strategy be used to perform priority-based task classification and dynamically cluster VMs based on runtime CPU and Random Access Memory (RAM) weights?

•Does the dual-layer clustering approach improve scheduling efficiency (reduce average makespan) and resource alignment compared to (KmeanH and Ekmean) methods?

•To what extent does the level-matching allocation strategy between clustered tasks and VMs enhance LB and reduce resource contention?

•How does the fallback assignment mechanism contribute to execution continuity and utilization when ideal VMs are not available in a corresponding cluster?

•Can the proposed PAKS-VMC framework contribute to improving LB in heterogeneous CC infrastructures?

The key contributions of the proposed framework are pointed out below:

•Propose an integrated dual-layer runtime unsupervised clustering mechanism using K-means++ to classify cloudlets based on task length into high, medium, and low priority levels, and to cluster VMs dynamically in parallel into three resource tiers based on runtime CPU and memory characteristics (dynamic weighting).

•Introduce a dynamic weighting mechanism for VMs, calculated from normalized CPU and memory metrics using tunable weight factors (α for CPU and β for RAM) to reflect the current state of VM resources.

•Propose a priority-conscious level-based task allocation strategy to ensure the highest possible efficiency of task execution and efficient resource utilization. Allocate each cloudlet in the cluster to its own VM at the same level (e.g., a high-priority task to a high-resource VM).

•Include a dynamic fallback mechanism that keeps tasks running by reallocating them to other VM clusters when resources are unavailable at the desired level.

•Develop an integrated dual-layered scheduling framework known as PAKS-VMC, and evaluate its effectiveness compared to existing clustering-based scheduling methods (Ekmean and KmeanH) in a CloudSim simulation setup, where results of the experiments indicate a reduced average makespan and better load-balancing performance under the tested scenarios.

The proposed framework, PAKS-VMC, is a hybrid dynamic load-balancing framework designed for heterogeneous cloud environments. It implements unsupervised K-means++ clustering within an integrated dual-layer execution architecture, with cloudlets assigned to priority classes based on length, and VMs dynamically assigned to resource tiers based on their capacity, using CPU and RAM weight factors. Besides the dual-layer clustering mechanism, the framework also implements a priority-aware level-matching allocation policy, a ULC to select the most appropriate VM based on its dynamic resource weight, and a fallback reallocation mechanism to handle allocation failures, which support adaptive scheduling, improved resource utilization, and enhanced execution performance.

The rest of this paper is further subdivided into four sections: Section 2 presents the related work, Section 3presents the proposed approach, Section 4 presents the experimental environment and results, Section 5 presents the conclusion and future work.

2. RELATED WORK

Task scheduling and LB in CC have become a topic of widespread research over the past few years, as the need to utilize resources better and enhance QoS is growing [28]. Numerous studies have explored grouping and heuristic optimization-based mechanisms to improve QoS, minimize makespan, and reduce execution time. In particular, clustering-based methods have been widely used for classifying tasks or VMs to support more organized allocation techniques [29]. However, most of these lack the flexibility to adapt to heterogeneous, real-time VM environments, where CPU and memory availability fluctuate dynamically. Also, most existing literature focuses on task clustering or VM clustering individually without considering a coordinated task–VM clustering mechanism that matches the characteristics of tasks with those of VMs. This constraint often leads to suboptimal workload-resource mapping and poor workload performance in dynamic environments. This section provides a description of the main existing approaches, outlines their advantages and disadvantages, and emphasizes the research gap targeted by the proposed PAKS-VMC model. Relevant studies in this domain are summarized as follows: Muthusamy and Chandran [30] proposed a model of a framework called Cluster-Based Task Scheduling (CBTS) to cluster tasks according to their length and VMs according to processing capacity using K-means clustering. The purpose of the model is to reduce execution time and makespan, and to achieve dynamic LB in CDCs. Sohani and Jain [31] proposed a RA model that was dependent on the forecasting of future resource usage in order to optimize VM allocation in real time. The approach optimizes LB, shortens the makespan, and minimizes energy consumption. Meyer et al. [32] introduced the ML-based dynamic classification paradigm for interference-aware and CC resource scheduling. The method focuses on classifying applications by interference level to reduce performance degradation and SLA violations caused by resource contention. The evaluation metrics include resource utilization, SLA violations, and workload efficiency. Meyer et al. [33] proposed Interference-Aware Dynamic Architecture (IADA), a scheduling model that aims to alleviate cross-application interference in cloud environments, especially for latency-sensitive services. The scheme combines Software-Defined Networking (SDN) and dynamic scheduling to account for the workload's sensitivity to delay. The core goal is to reduce SLA breaches and increase resource utilization when numerous applications share the same resources.

Mustapha and Gupta [34] presented a way of optimizing RA and task scheduling with the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) mechanism that is premised on the idea of categorization of VMs based on their past behavior. This approach minimized execution time and the average start and finish times. Elsakaan and Amroun [35] proposed a multi-level hybrid approach that uses K-means clustering, Round-Robin, and a GA to achieve effective LB. The process improves makespan, response time, and SLA compliance. Ghasemi and Keshavarzi [36] proposed an energy-efficient VM placement model using a K-means clustering algorithm with reinforcement learning to reduce energy consumption in heterogeneous cloud resource settings. The strategy delivered better VM utilization and reduced SLA violations. Lwin [37] introduced a multi-step scheduling model that incorporates K-means-based clustering and a multi-objective optimization algorithm for scientific activities in heterogeneous cloud environments. The methodology considers communication, computation, first-finish time, makespan, resource usage, and task length to improve scheduling decisions. Alsubaei et al. [38] proposed two K-means-based scheduling algorithms: Ekmean, in which tasks are grouped based on their structural similarity, and KmeanH, in which a combination of K-means and the HEFT mapping algorithm is employed to schedule and assign tasks to VMs. The solutions reduce the makespan and optimize scheduling in heterogeneous clouds. Zhu [39] proposed an unsupervised learning-based cloud resource management algorithm based on DBSCAN clustering. The approach classifies cloud resources based on utilization patterns, enabling proactive allocation and dynamic scaling. Dynamic clouds were tested for scalability, resource usage, and responsiveness. Lwin and Thaw [40] proposed a multi-objective scheduling management system for cloud computational scientific workflows using K-means clustering. The method uses computing and communication costs to sort jobs by shortest finish time and period, and allocate them to the right VMs. Experiments on Montage and Sipht workflows in WorkFlowSim indicate considerable improvements in makespan, execution time, and LB compared to FCFS and Min-Min. Alharbe [41] presented a new scheduling algorithm, Budget and Time Constrained Heterogeneous Early Completion (BDHEFT), an upgraded version of the HEFT algorithm that uses fuzzy waterfall methods for cloud project scheduling. By grouping resources according to their characteristics, the method reduces search costs and speeds up resource selection under budget and deadline constraints. Experimental validation shows that execution time and scheduling costs are reduced, but reliance on fuzzy modeling increases system complexity.

From the comparative analysis of the related works, key insights are observed:

•The majority of clustering-based methods [30, 34, 37, 40, 41] use K-means, DBSCAN, or fuzzy clustering to cluster tasks or resources, thereby improving execution time and makespan. Nevertheless, these approaches generally lack fallback mechanisms for unallocated cloudlets, making them less adaptable to real-time heterogeneous clouds. Interference-aware and predictive models [32, 33] can significantly improve SLA compliance; however, they often incur high computational costs and require workload profiling.

•Hybrid systems that combine clustering with metaheuristics or reinforcement learning [31, 36, 38, 39] are very effective at optimization but are computationally complex, less scalable, and highly dependent on parameter tuning or training processes. Energy-efficient schemes [36] enhance VM utilization but address power consumption only to a limited extent and do not account for task-level QoS needs.

Table 1. Comparative analysis of related work and the proposed PAKS-VMC framework

Reference	Method	Clustering Target	Dynamic VM Weighting	Priority-Aware Tasks	Fallback Mechanism	Limitation
Muthusamy and Chandran [30]	CBTS (K-means)	Tasks + VMs	×	×	×	No dynamic VM adaptation at runtime
Sohani and Jain [31]	Predictive RA	VMs only	Partial	×	×	High computational overhead
Meyer et al. [32]	ML Classification	Tasks only	×	×	×	Requires workload profiling
Mustapha and Gupta [34]	DBSCAN	VMs only	×	×	×	Sensitive to clustering parameters
Elsakaan and Amroun [35]	K-means + GA + RR	Tasks + VMs	×	Partial	×	Lacks adaptive VM weighting, poor scalability
Ghasemi and Keshavarzi [36]	K-means + RL	VMs only	Partial	×	×	Energy-focused, ignores task QoS
Lwin [37]	K-means + Multi-objective	Tasks only	×	Partial	×	No dynamic VM resource tracking
Alsubaei et al. [38]	Ekmean / KmeanH	Tasks only	×	√	×	Static clustering, random assignment
Zhu [39]	DBSCAN	Resource utilisation patterns	Partial	×	×	Sensitive to clustering parameters
Lwin and Thaw [40]	K-means workflow	Tasks only	×	Partial	×	Limited to workflow tasks
Alharbe [41]	BDHEFT + Fuzzy	Resources	Partial	√	×	Fuzzy complexity, budget-constrained
Proposed PAKS-VMC	K-means++ Dual-Layer	Tasks + VMs	√	√	√	Evaluated only within CloudSim-scale simulations

Note: VM = virtual machine, CBTS = Cluster-Based Task Scheduling, RA = resource allocation, ML = machine learning, DBSCAN = Density-Based Spatial Clustering of Applications with Noise, GA = Genetic Algorithm, QoS = quality of service, PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, BDHEFT = Budget and Time Constrained Heterogeneous Early Completion, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

Table 2. Methodological comparison of KMPPVM-DRM and PAKS-VMC

Methodological Aspect	Previous Work (KMPPVM-DRM)	Present Work (PAKS-VMC)
Research objective	Dynamic VM clustering and resource allocation (RA)	Integrated runtime scheduling of cloudlets and VMs
Research scope	VM clustering and resource allocation (RA)	Integrated cloudlet–VM scheduling and runtime resource management
Main contribution	Dynamic K-means++ -based VM clustering framework	Integrated dual-layer K-means++ scheduling framework
Clustering mechanism	Dynamic K-means++ clustering for VMs	Coordinated dual-layer K-means++ clustering for cloudlets and VMs
Scheduling strategy	VM allocation based on dynamic VM weights	Priority-aware level matching with runtime adaptive scheduling
Fallback mechanism	Not included	Runtime fallback allocation
Experimental evaluation	ExT, AST, AFT	Makespan, AExT, AST, AFT
Research progression	Methodological foundation of the present study	Methodological extension with integrated runtime scheduling

Note: KMPPVM-DRM = K-means++ Based Dynamic Clustering Approach for Virtual Machine Resource Allocation in Cloud Data Centers, PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, VM = virtual machine, AST = Average Start Time, AFT = Average Finish Time, AExT = Average Execution Time.

Although previous works have shown the benefits of clustering-based and hybrid scheduling methods to reduce makespan, to balance the load, and to optimize resource utilization, several limitations remain in heterogeneous cloud environments. Most existing approaches perform either task clustering or VM clustering independently, which limits adaptive matching between workload characteristics and runtime resource states. Further, many approaches rely on static scheduling assumptions and heuristic allocation policies, or employ computationally intensive optimization mechanisms, limiting their flexibility under dynamically varying workloads. Furthermore, several clustering-based scheduling frameworks lack runtime fallback mechanisms and dynamic VM reclassification, which may lead to resource imbalance and allocation interruptions under heterogeneous conditions. Thus, there is still a need for an integrated, lightweight scheduling framework that efficiently supports adaptive task prioritization, dynamic VM clustering, and runtime resource-aware allocation, all within a unified scheduling architecture. Table 1 provides a comparative analysis of related studies and the PAKS-VMC framework.

Our previously published study [27] introduced a dynamic K-means++-based framework for VM clustering and RA in heterogeneous cloud environments. The study demonstrated the effectiveness of adaptive VM clustering for improving resource management. Building upon this methodological foundation, the present work extends the previous framework by integrating coordinated cloudlet scheduling with dynamic VM clustering into a unified runtime scheduling framework. Specifically, the proposed PAKS-VMC framework introduces coordinated dual-layer clustering, priority-aware level matching, runtime adaptive scheduling, and a fallback reallocation mechanism. A methodological comparison between KMPPVM-DRM and the proposed PAKS-VMC framework is presented in Table 2.

In light of these limitations, this paper proposes the PAKS-VMC framework, which integrates dual-layer K-means++ clustering for both VMs and cloudlets within a unified runtime scheduling architecture. The framework's goal is to enhance adaptive task–VM matching, runtime resource utilization, and scheduling performance in heterogeneous cloud environments. Accordingly, the proposed PAKS-VMC framework differs from existing clustering-based scheduling approaches by integrating a K-means++-based clustering mechanism, dynamic VM weight updating, priority-aware level matching, and fallback-based runtime allocation. The details of the framework and its scheduling mechanisms are presented in the next section.

3. PROPOSED FRAMEWORK

This section introduces the proposed framework, PAKS-VMC, for the CC environment. It implements unsupervised K-means++ clustering within an integrated dual-layer runtime framework. This dual-layer clustering mechanism integrates the advantages of priority-aware cloudlet classification and adaptive dynamic VM clustering simultaneously, forming a hybrid scheduling mechanism that leverages the benefits of dynamic scheduling and balanced RA. The main objective of this approach is to support adaptive task scheduling and balanced resource utilization in cloud data centres, thereby reducing average makespan and improving load-balancing performance under dynamic workload conditions. In contrast to traditional approaches that group tasks or VMs separately, the proposed framework integrates cloudlet classification, VM clustering, and runtime allocation mechanisms within a unified scheduling architecture, as illustrated in Figure 2.

The proposed approach (PAKS-VMC) includes the following.

3.1 Integrated dual-layer clustering mechanism

Task clustering: Upon arrival at the broker, tasks are clustered by K-means++ into three priority classes (high, medium, low) based on their length, enabling prioritization. Each priority level is placed in a separate queue.

VM clustering: VMs are dynamically grouped into three resource levels–low, medium, and high– using K-means++ based on normalized weights calculated from CPU and RAM capacities. These weights are calculated according to Eq. (1), which allows clusters to adapt to the current state of system resources.

$$W_{vmi} = \frac{(\alpha * Cpu_{vm}) + (\beta * Ram_{vm})}{100} \quad (1)$$

W_{vmi} is the computed VM weight, while α and β in Eq. (1)

are weighting coefficients assigned to CPU and RAM resources, respectively, such that ($\alpha + \beta = 1$). These parameters, $\alpha = 0.6$ and $\beta = 0.4$, were selected to reflect the greater importance of CPU relative to memory resources in a cloud environment. Also, Cpu_{vm} in Eq. (1) means available CPU resources, while Ram_{vm} means available RAM resources in the VMs. During runtime, a weight map is maintained to record VM identifiers, available resource capacities, and the VM weight values. These weights are updated continuously and reflect the system's state. VMs are grouped into multiple levels of availability using the K-means++ algorithm based on the updated weights. Figure 3 illustrates the complete VM weight computation and clustering process.

In PAKS-VMC, tasks and VMs are grouped into k clusters through the K-means++ clustering algorithm. The key features of this algorithm are as follows:

•Number of clusters (k): In this study, $k = 3$, representing three distinct levels of VM capacity and corresponding task compatibility (high, medium, and low).

•Centroid initialization strategy: K-means++ selects initial centroids to enhance stability and convergence. The initial centroids for VM clustering are selected based on dynamic weights computed from CPU and RAM capacities. In task clustering, the centroids are calculated based on task length. This method makes the clusters more balanced, thus accelerating convergence.

•Distance metric: The clustering process uses the Euclidean distance metric. For VMs, the distance is calculated using normalized weights for CPU and RAM that reflect the availability of processing and memory resources. For tasks, the distance is computed using length values, allowing similar-sized tasks to be effectively grouped.

3.2 Priority-aware level-matching strategy

After the integrated dual-layer clustering of tasks and resources is complete, a priority-aware level-matching strategy is implemented, pairing each task queue with the VM group of its respective level (e.g., the high-priority task queue is paired with the high-capacity VM group). This step organizes a systematic correspondence between task queues and VM groups based on their respective levels. In contrast, the actual task assignment is deferred to the next step via the ULC. This alignment ensures that allocation decisions remain consistent with task requirements and the available resource capacities throughout the scheduling process.

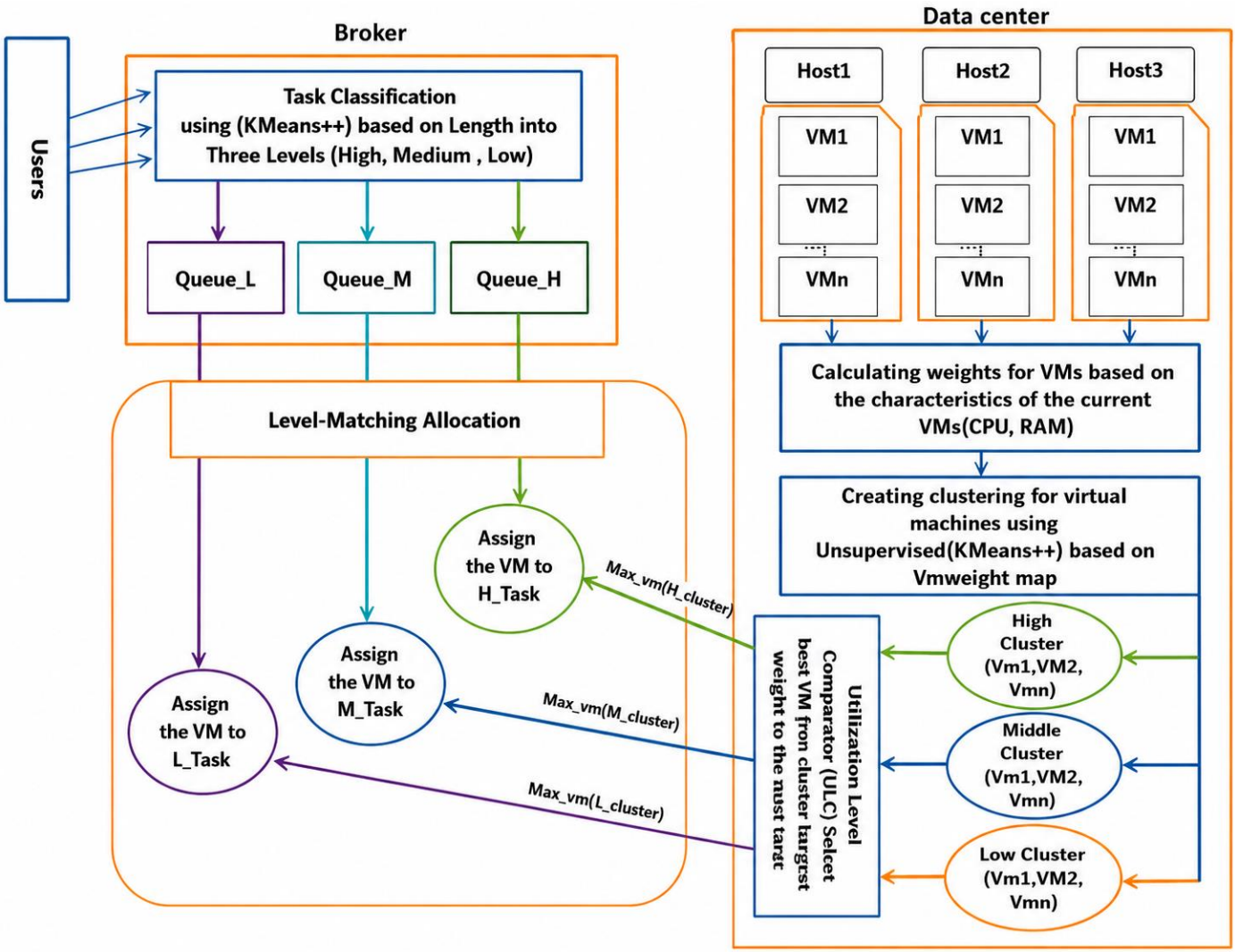

Figure 2. Diagram of the Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets (PAKS-VMC) framework

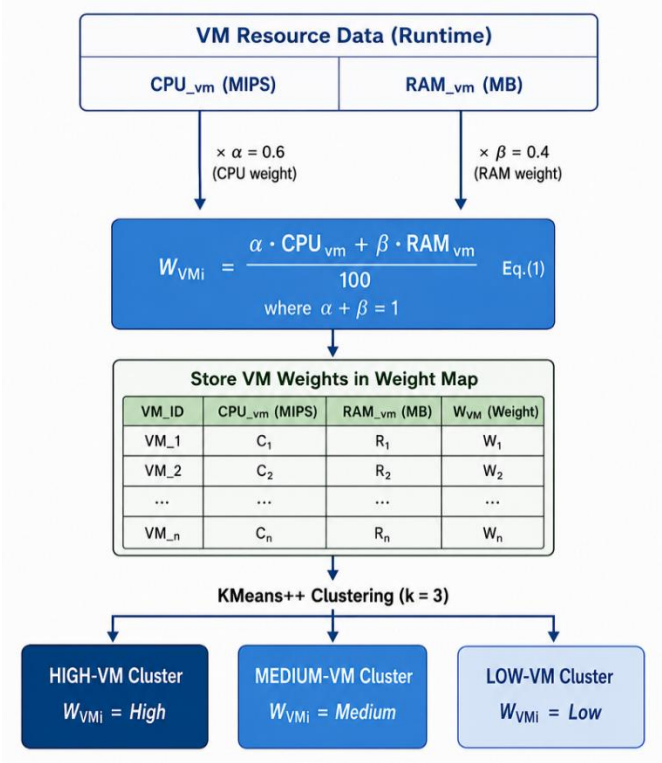

Figure 3. Virtual machine (VM) dynamic weight calculation, weight map storage, and K-means++ Clustering process

3.3 Utilization-Level Comparator

The ULC is applied to select the optimal VM within each VM cluster based on the earlier-calculated weight. ULC takes such weights and picks the VM with the highest resource score. Then the task in the queue related to that VM is allocated to the selected VM, as specified in Eq. (2). The process is dynamically and concurrently applied across the three priority levels to ensure a proper match between task needs and resource capacities. The VMs are utilized optimally during runtime.

$$VM_{assigned} = argmax_{vmi \in cj} W_{vmi} \quad (2)$$

where, $VM_{assigned}$: the VM chosen to work on the following task. VM_i: a VM of C_j. C_j: the cluster of the VMs. W_{vmi}: the weight of a VM_i is the optimal capacity, or how suitable the VM is for the cloudlet. $argmax$: the function selects the VM with the highest weight.

The dynamically changing weight map for the VMs is updated after task assignment, based on variations in resource utilization caused by task execution.

These updated weights are then used in subsequent scheduling cycles to support adaptive allocation decisions, and the process continues iteratively until all necessary tasks are accomplished.

3.4 Fallback mechanism

The proposed PAKS-VMC framework includes a lightweight fallback mechanism to ensure allocation continuity during runtime scheduling. The number of available VM tiers can temporarily fluctuate as VM resource weights are dynamically updated and re-clustering is initiated, potentially resulting in fewer than 3 VM tiers at a given scheduling cycle. When the VM tier matching a given cloudlet priority level is unavailable, the affected cloudlets in that priority queue are rescheduled to the remaining VM tiers based on the updated resource suitability. The mechanism averts execution interruption, minimizes queuing delay, and promotes even distribution of available resources in dynamic and heterogeneous cloud environments, as explained in Eq. (3):

$$VM_{\{alt\}} = argmax_{\{VM_i \in C_{\{\{avail\}\}}\}}(W_{\{vmi\}}) \quad (3)$$

where, VM_{alt}: the fallback mechanism chooses an alternative VM when the VM tier for the priority match is unavailable. C_{avail} is the set of the available VMs of the remaining VM tiers after re-clustering. Figure 4 illustrates the priority-aware level-matching and fallback allocation mechanisms in the proposed PAKS-VMC framework.

Following the computation of the weights of VMs as per Eq. (1), the selection mechanism as explained in Eq. (2), and the fallback mechanism as defined in Eq. (3), the symbols and parameters of the proposed PAKS-VMC model are summarized in the Nomenclature in order to facilitate easy understanding.

Overall, the proposed framework (PAKS-VMC) is a sophisticated, adaptive framework for dynamic RA and task scheduling in heterogeneous cloud systems. It exploits the power of an integrated dual-layer K-means++ clustering mechanism, combining priority-aware cloudlet grouping with resource-aware dynamic VM clustering, to deliver improved performance and scalability compared to existing clustering- and scheduling-based algorithms. The goal of the proposed PAKS-VMC is to schedule the workload and maximize RA in heterogeneous cloud data centres. The pseudocode for the proposed PAKS-VMC approach is provided to depict the underlying formulas, parameters, and decision-making process. The proposed PAKS-VMC approach was tested with various task sizes (e.g., 50–1000 tasks) and different numbers of VMs (e.g., 8, 16, and 32). The flowchart shown in Figure 5 represents the steps involved in the proposed PAKS-VMC scheduling and RA approach. The integrated process for dual-layer clustering is described in the following flowchart, with cloudlets clustered using the priority K-means++ protocol, and VMs clustered dynamically based on their weights. It is also the level-matching strategy that is sensitive to priority and the ULC-based VM selection mechanism during runtime scheduling. Finally, the flowchart illustrates the adaptive (dynamic) RA within the dynamic weight update, fallback handling, and the QoS assessment phases. The experiments were carried out over a limited range of workloads; however, the computational properties of K-means++ suggest it may be applicable in larger cloud environments. Since K-means++ is an unsupervised algorithm, it adapts to changes in the number of VMs. At the same time, the weight-based classification (CPU and RAM with adjustable parameters α and β) allows the system to extend naturally to larger VM pools and heterogeneous workloads. Future work includes evaluating the proposed framework in larger-scale cloud environments with up to 100 VMs, workloads of over 10,000 tasks, and simulated concurrent user requests.

Moreover, unlike either clustering-only scheduling systems such as the study [30] or complex hybrid systems such as the study [35], the PAKS-VMC framework, as shown in Algorithm 1, is an integrated dual-layer runtime architecture. Although other approaches, such as interference-aware or

predictive scheduling [32], address SLA compliance through workload profiling, they are computationally expensive. Although these methods are not part of the existing framework, they may be explored in future research to be combined with the proposed PAKS-VMC and predictive or regression-based models to select VMs more appropriately, thereby predicting resource usage and allocating resources more efficiently in the highly dynamic cloud environment.

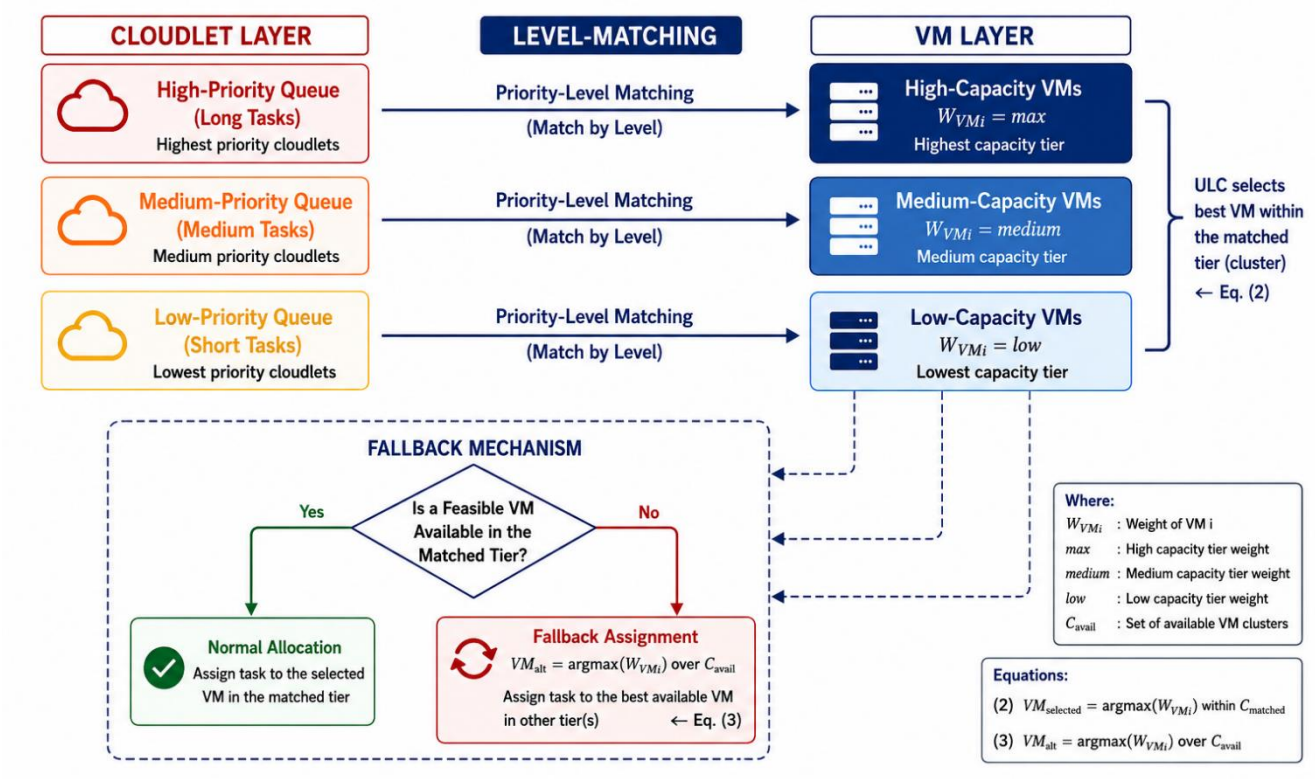

Figure 4. Priority-aware level-matching and fallback allocation mechanism in the proposed Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets (PAKS-VMC)

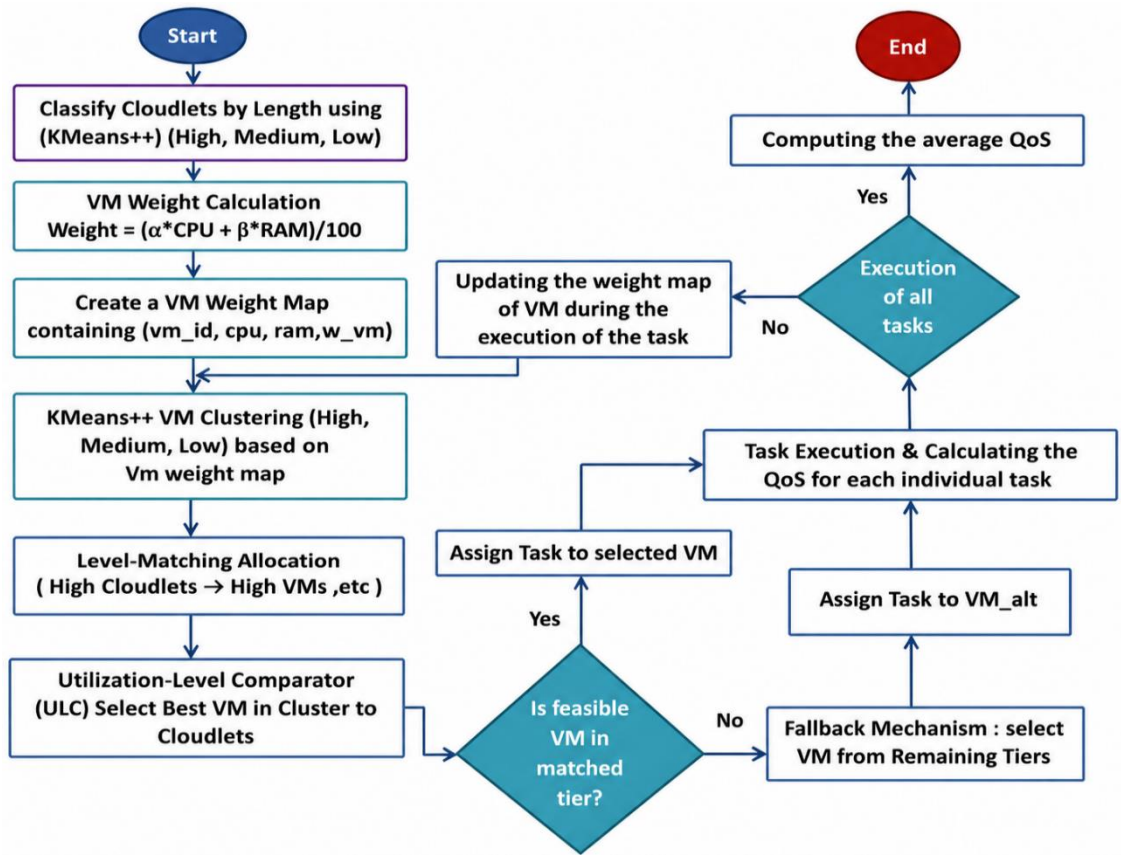

Figure 5. Flowchart illustrating the proposed framework

Algorithm 1. The proposed (PAKS-VMC) framework

Input:
• List of Cloudlets (Tasks) with length $L(c)$
• List of VMs with characteristics
• Clustering parameter $K = 3$
Output: Cloudlets assigned to VMs, QoS averages
Phase 1: Integrated Dual-Layer Clustering (Single Framework)
1. Task clustering (K-means++): three priority queues by task length
2. *TaskClusters* ← *K-means++*(*feature* = $L(c)$, $K = 3$)
3. Map task clusters into three priority queues:
(Q_H, Q_M, Q_L) ← MapTaskClustersToQueues(TaskClusters)
high, medium, low
4. VM weight computation + VM clustering (K-means++): three VM groups by weight
For each $\boldsymbol{VM_i} \in \boldsymbol{V}$ do
5. $\boldsymbol{VM_{w(VM_i)}} = ((\boldsymbol{\alpha} * \boldsymbol{Cpu_{vm(VM_i)}} + (\boldsymbol{\beta} * \boldsymbol{Ram_{vm(VM_i)}}))/\mathbf{100}$
End for
6. *VMClusters* ← *K-means++*(*feature* = $\boldsymbol{VM_{w(VM_i)}}$, $K = 3$)
7. (C_H, C_M, C_L) ← *MapVMClustersToGroups(VMClusters)*
high, medium, low
Phase 2: Priority-Aware Level-Matching
8. Align $Q_H \leftarrow C_H$, $Q_M \leftarrow C_M$, $Q_L \leftarrow C_L$
Phase 3: ULC Selection + Simultaneous Assignment
9. While (Q_H not empty) OR (Q_M not empty) OR (Q_L not empty) do
10. For each pair (Q_c, Q_{cj}) ∈ {(Q_H, C_H), (Q_M, C_M), (Q_L, C_L)} do
11. If Q is empty then continue
12. $c \leftarrow$ front(Q_c)
13. ULC inside the matched group: choose the VM with highest weight
$$\boldsymbol{VM_{assigned}} = \boldsymbol{argmax_{vmi \in cj}} \boldsymbol{W_{vmi}}$$
14. If $\boldsymbol{VM_{assigned}}$ is feasible for c then
15. Assign $c \rightarrow \boldsymbol{VM_{assigned}}$; pop(Q_c)
16. Update VM weight after assignment/execution
17. Update $\boldsymbol{Cpu_{vm}}(\boldsymbol{VM_{assigned}})$, $\boldsymbol{Ram_{vm}}(\boldsymbol{VM_{assigned}})$
$\mathbf{VM_{w(VM_{assigned})}} = ((\boldsymbol{\alpha} * \boldsymbol{Cpu_{vm}}(\mathbf{VM_{assigned}}) + (\boldsymbol{\beta} * \boldsymbol{Ram_{vm}}(\mathbf{VM_{assigned}})))/\mathbf{100}$
Else
Phase 4: Fallback Mechanism
$$\boldsymbol{VM_{\{alt\}}} = \boldsymbol{argmax}_{\{\boldsymbol{VM_i} \in \boldsymbol{C_{\{\{avail\}\}}}\}} (\boldsymbol{W_{\{vmi\}}})$$
18. If $\boldsymbol{VM_{\{alt\}}}$ is feasible then
19. Assign $c \rightarrow \boldsymbol{VM_{\{alt\}}}$; pop($Q_c$)
20. Update $\boldsymbol{Cpu_{vm}}(\boldsymbol{VM_{\{alt\}}})$, $\boldsymbol{Ram_{vm}}(\boldsymbol{VM_{\{alt\}}})$
$\boldsymbol{VM_w}(\boldsymbol{VM_{\{alt\}}})) = ((\boldsymbol{\alpha} * \boldsymbol{Cpu_{vm}}(\boldsymbol{VM_{\{alt\}}})) + (\boldsymbol{\beta} * \boldsymbol{Ram_{vm}}(\boldsymbol{VM_{\{alt\}}})))/\mathbf{100}$
End if
End if
End for
End while
For each executed_task_ti do
Record execution time, start and finish time
If all tasks completed then
Compute and output average QoS (Makespan, execution, start, and finish time) metrics; break
Else
Update VMweights dynamically to reflect current loads; continue

4. EXPERIMENTAL ENVIRONMENT AND RESULTS

This section describes the experimental setup of the proposed framework and carries out a detailed interpretation of the evaluation results.

4.1 Experimental environment

4.1.1 Hardware specification

All experiments were performed on a local machine with the CPU, RAM, and Windows 10 (64-bit) operating system specifications outlined in Table 3. The purpose of such a setup is to show the lightness and affordability of a solution that does not require powerful computing tools to deliver efficient results. More sophisticated load-balancing algorithms, such as deep learning, tend to require powerful GPUs and large amounts of memory.

Table 3. Hardware specification

Component	Specification
Operating system	Windows 10, 64-bit architecture
Processor	Intel Core i7 @ 2.6 GHz
RAM	16.0 GB

4.1.2 Software and tools

CloudSim serves as the primary simulation tool for researchers who measure cloud performance and model cloud solutions to eliminate dependence on physical infrastructure and minimise costs. It is an externally importable software that can be used in programming environments to simulate CC infrastructures. Entities and computing resources used to test the proposed method are modelled based on the allocation and scheduling scenario within the cloud environment. The simulation was conducted using CloudSim 3.0.3. It was chosen for this work because it performs well, supports Java-based development, and is compatible with external Java libraries for clustering (such as Smile). It is applicable to the proposed model, making it compatible with ML and reproducible in similar research. Simulation-based results may not fully capture all runtime characteristics of real-world cloud infrastructures, such as network variability, virtualization overhead, and multi-tenant interference. The CloudSim simulation environment was configured using the default event-driven architecture provided by CloudSim 3.0.3.

VMs were created using the CloudletSchedulerSpaceShared scheduling policy, in which each VM executes cloudlets in a space-shared manner across the allocated processing elements (PEs). CloudSim internally manages the event queue through its discrete-event simulation engine. In addition, waiting queues were created for tasks after applying the clustering process to classify cloudlets into high-, medium-, and low-priority groups based on their computational lengths, resulting in dynamic queue sizes that vary with workload conditions and runtime clustering results. The simulation was undertaken using a single data centre with 2 hosts, with the number of VMs set to 8, 16, and 32 to represent different levels of resource availability. The sizes of the experimental tasks ranged from 50 to 1000 cloudlets. These different task loads and VM settings were used to test the performance of the proposed PAKS-VMC approach with typical and overloaded workloads in the simulation environment. The summary of the configuration of all submitted tasks (cloudlets) is in Table 4. The number of tasks ranges from 50 to 1000, and the simulation can handle both

heavy and light workloads. There are also three categories of tasks by length, with lengths of 1,000 (lightweight user request), 2,000 (moderate processing), and 4,000 (compute-intensive application), similar to a data analytics application. These simulation values were chosen to represent a diversity comparable to that found in previous task scheduling studies [38]. Its design will enable extensive testing of the load-balancing algorithms on a range of heterogeneous workloads of different magnitudes and complexity.

Table 5 shows the Low, Medium, and High VM types, along with RAM, MIPS, and PEs. These configuration values are adopted from [38] to maintain consistency with existing benchmarks. These configurations are used in the proposed PAKS-VMC framework, where VMs are classified into low-, medium-, and high-resource clusters using K-means++ based on their CPU and memory weights. As a result of this categorization, the cloud environment can distribute work based on size and resource requirements, resulting in real-time, balanced utilization of all resources.

Table 4. Task workload configuration parameters

Parameters	Values
Task length	1000/2000, 4000
Input file size	300 Byte
Output file size	300 Byte
Processing element (PE)	1-2

Table 5. Type of virtual machines (VMs)

VM Category	Mips, RAM (MB), Pe
Low	(1000,512,1)
Medium	(3000,1536,2)
High	(4000,2048,3)

4.2 Performance metrics

To fully assess the performance of the proposed PAKS-VMC framework, multiple performance indicators were considered. To be consistent with the study under comparison [38], the average makespan measure was adopted for direct comparison. However, the proposed PAKS-VMC framework incorporates a broader range of performance indicators within the evaluation framework to make it more comprehensive, including Average Start Time (AST), Average Finish Time (AFT), and Average Execution Time (AExT).

•**Makespan:** The maximum time taken by any Cloudlet to execute on available VMs. It is among the most significant factors in the assessment of scheduling algorithms, as it directly affects the system's overall performance and the time required to complete workloads. The mathematical formulation of makespan is adopted from [38], as expressed in Eq. (4):

$$Makespan = Max\{Finishing\ Time(Ti, Vj)\} \quad (4)$$

•**Execution Time (ExT):** ExT is the actual CPU execution duration taken to perform a task on a VM [42]. It is one of the primary metrics of VM performance because minimizing execution time also improves overall system performance.

•**Finish Time (FT) and Start Time (ST):** The start time and finish time of a task are the times the task begins and finishes, respectively. These are some important points to consider when scheduling. ST occurs when a resource (e.g., a VM) is assigned to the task and the task starts running. FT is the time at which the task finishes its execution. The completion time (CT) or (FT) is calculated with the help of Eq. (5) below:

$$FT = ST + ExT \quad (5)$$

where, ST is the start time, and ExT is the execution time.

In that regard, the improvement is reflected in reductions in the following metrics: average makespan, ExT, ST, and FT. The fact that all these metrics have lower values implies that it will run faster, experience fewer delays, and better utilize its resources, resulting in improved QoS and LB. All time-based metrics are reported in seconds (s), consistent with the CloudSim simulation environment.

4.3 Evaluation of PAKS-VMC under different task loads and Virtual Machine configurations

Here, the proposed PAKS-VMC framework is evaluated under different test cases with numbers of VMs of 8, 16, and 32 to represent varying levels of resource availability. The sizes of the experimental tasks ranged from 50 to 1000 cloudlets, allowing testing of the framework under varying load conditions. This variant allows for simulating a variety of cloud conditions and checking scheduling and resource-allocation actions when workload increases. The proposed PAKS-VMC framework is compared with representative baseline clustering-based scheduling models, i.e., Ekmean and KmeanH [38]. In this paper, the following performance measures are used for evaluation:

Table 6. Average makespan (s) with 8 virtual machines (VMs)

No. of Tasks	KmeanH	Ekmean	Proposed PAKS-VMC
50	333	327.6	31.161
100	717.6	703	71.804
150	1035	1029.6	121.982
200	1360	1580	145.212

Note: PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

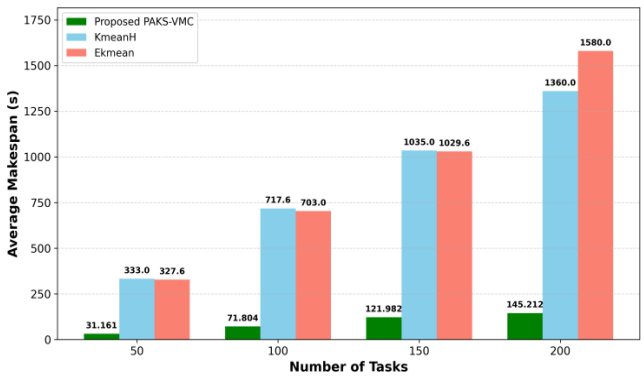

Figure 6. Comparison of average makespan (s) with 8 virtual machines (VMs)

Makespan: Table 6 compares the models (Ekmean, KmeanH, and PAKS-VMC) across various task counts and VM configurations. The average makespan of the Ekmean model also increases with the number of tasks, from 327.6 s for 50 tasks to 1580 s for 200 tasks, indicating that the average makespan grows proportionally with the task load. The KmeanH tends to follow the same trend, with the average

makespan values increasing as the number of tasks increases. Notably, the suggested PAKS-VMC framework has a much lower average makespan when all tasks are considered (e.g., 31.161 s at 50 tasks) and better average makespan efficiency than the other models. In Figure 6, the average makespan of PAKS-VMC has been compared with those of other models. The lower average makespan of the PAKS-VMC model indicates that the model has the potential to make task scheduling, RA, and the system as a whole more efficient.

In Tables 7 and 8, the average makespan values represent the maximum finish times averaged across all cloudlets executed on the available VMs by various models (Ekmean, KmeanH, and PAKS-VMC) for different task counts and VM configurations. For Ekmean and KmeanH, the average makespan shows a clear upward trend as the number of tasks increases; that is, the higher the number of tasks, the greater the average makespan in larger workloads, as demonstrated in Figures 7 and 8. In particular, the average makespan of Ekmean increases from 2017.6 s at 200 tasks to 17625 s at 1000 tasks with 16 VMs, and from 2027 s to 13664 s under the same task range with 32 VMs. Conversely, the proposed PAKS-VMC framework has a lower average makespan across all task counts with 16 and 32 VMs, demonstrating its scheduling and resource utilization efficiency under constrained resource conditions. For example, PAKS-VMC achieves a significantly lower average makespan of 1070.058 s for 1000 tasks with 16 VMs and 1059.386 s with 32 VMs. The reduced average makespan in the PAKS-VMC model is due to its ability to optimize task scheduling and dynamically allocate resources within the system, leading to rapid execution and improved system performance.

Execution Time (ExT), Start Time (ST), and Finish Time (FT): Table 9 shows the average values of Execution Time (AExT), Start Time (AST), and Finish Time (AFT) obtained using the proposed PAKS-VMC model with 8 VMs. The three time-related metrics, reported in seconds(s), all grew in a similar, proportional manner as the number of submitted tasks increased from 50 to 200, indicating the scalability of the proposed model under heavier loads. Specifically, the AExT rose from 0.962 at 50 tasks to 1.103 s at 200 tasks, while the AST increased from 9.488 to 47.026 s, and the AFT from 10.45 to 48.099 s.

The performance across all measures clearly shows that PAKS-VMC is a balanced scheduler whose delays are reduced, and for the heterogeneous cloud, it is highly scalable, with good and predictable performance. The fact that AST and AFT values are very close to each other also indicates that there are no significant delays in queues and that tasks begin and end within a small time interval, providing evidence of reasonable task allocation and smooth execution under the proposed PAKS-VMC framework.

Table 7. Average makespan (s) with 16 virtual machines (VMs)

No. of Tasks	KmeanH	Ekmean	Proposed PAKS-VMC
200	1563.6	2017.6	140.231
400	2952	4585.6	238.99
600	4850	7931.3	624.848
800	5985	10562	869.651
1000	7662	17625	1070.058

Note: PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

Table 8. Average makespan (s) with 32 virtual machines (VMs)

No. of Tasks	KmeanH	Ekmean	Proposed PAKS-VMC
200	1482	2027	131.335
400	2796	4411	232.18
600	4213	7487.3	590.319
800	5573	12082	835.813
1000	6914	13664	1059.386

Note: PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

Table 9. Comparison of AExT, AST, and AFT (s) with 8 virtual machines (VMs)

No. of Tasks	AExT(s)	AST(s)	AFT(s)
50	0.962	9.488	10.45
100	1.009	24.288	25.312
150	1.060	44.084	45.145
200	1.103	47.026	48.099

Note: AST = Average Start Time, AFT = Average Finish Time, AExT = Average Execution Time.

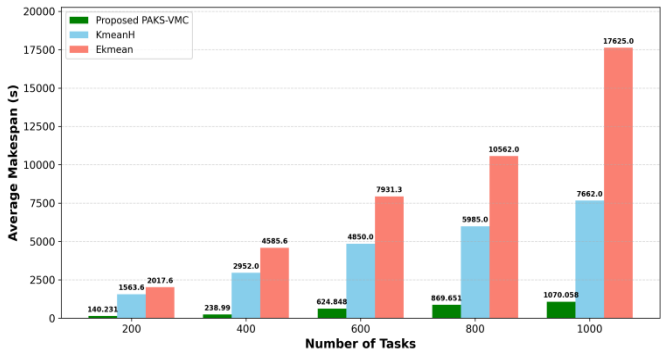

Figure 7. Comparison of average makespan (s) with 16 virtual machines (VMs)

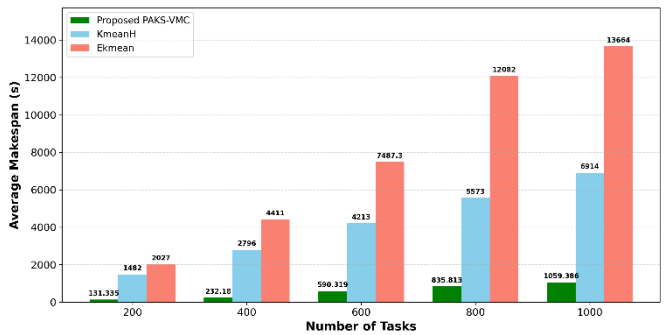

Figure 8. Comparison of average makespan (s) with 32 virtual machines (VMs)

The simulation results for 16 and 32 VMs, shown in Tables 10 and 11, reveal the performance evolution of the proposed PAKS-VMC framework under higher computational capacities and task loads ranging from 200 to 1000. With 16 VMs, the AExT rises from 1.095 s at 200 tasks to 1.592 s at 1000 tasks, while AST and AFT increase from approximately 43.872 to 329.674 s and 44.968 to 331.267 s, respectively.

Similarly, under 32 VMs, the corresponding values of AExT, AST, and AFT are slightly lower across all workloads; e.g., at 200 tasks, the corresponding values are 1.0743 s for AExT, 36.5576 s for AST, and 37.6322 s for AFT, while at 1000 tasks, they are 1.5907, 320.5589, and 322.1492 s,

respectively.

The result shows that as the number of VMs increases, all the time-dependent metrics tend to decrease, showing the efficiency in balancing workloads and delivering stable and predictable performance in different scenarios.

Table 10. Comparison of AExT, AST, and AFT (s) with 16 virtual machines (VMs)

No. of Tasks	AExT (s)	AST (s)	AFT (s)
200	1.095	43.872	44.968
400	1.216	75.975	77.19
600	1.422	215.55	216.972
800	1.538	284.447	285.986
1000	1.592	329.674	331.267

Note: AST = Average Start Time, AFT = Average Finish Time, AExT = Average Execution Time.

Table 11. Comparison of AExT, AST, and AFT (s) with 32 virtual machines (VMs)

No. of Tasks	AExT (s)	AST (s)	AFT (s)
200	1.0743	36.5576	37.6322
400	1.2210	67.2492	68.4704
600	1.3908	196.0602	197.4511
800	1.5153	264.090	265.6053
1000	1.5907	320.5589	322.1492

Note: AST = Average Start Time, AFT = Average Finish Time, AExT = Average Execution Time.

4.4 Performance analysis

A series of experiments was performed to assess the effect of computing resources on the performance of the proposed framework with respect to 16 and 32 VMs and 200-1000 tasks. This experimental context aimed to compare the flexibility of the framework in low- and high-resource contexts and to assess the stability of the rate of improvement when using the framework with the tasks offered in the set environment compared with that of the Ekmean and KmeanH models. The proposed PAKS-VMC framework has been evaluated using two analytical performance metrics: Relative Reduction, which measures the percentage improvement relative to the comparative models, and the harmonic mean, which offers a balanced measure of overall performance.

4.4.1 Performance evaluation for the 16-VM scenario

Table 12 shows the relative reduction in the average makespan achieved by the proposed framework compared to baseline models. The proposed model showed significant performance improvements, achieving 85.46% to 91.90% over KmeanH and 91.76% to 94.78% over Ekmean.

Table 12. Relative reduction (%) in average makespan using 16 virtual machines (VMs)

No. of Tasks	KmeanH	Ekmean
200	91.03	93.04
400	91.90	94.78
600	87.11	92.12
800	85.46	91.76
1000	86.03	93.92

Note: KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

To provide an aggregate evaluation of the proposed method's performance, particularly given differing task counts, the harmonic mean of each method's average makespan was calculated. Table 13 shows the harmonic means of the results.

Table 13. Harmonic mean of average makespan (s) with 16 virtual machines (VMs)

Methods	Harmonic Mean(s)
Ekmean	5044.33
KmeanH	3373.62
Proposed (PAKS-VMC)	333.33

Note: PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

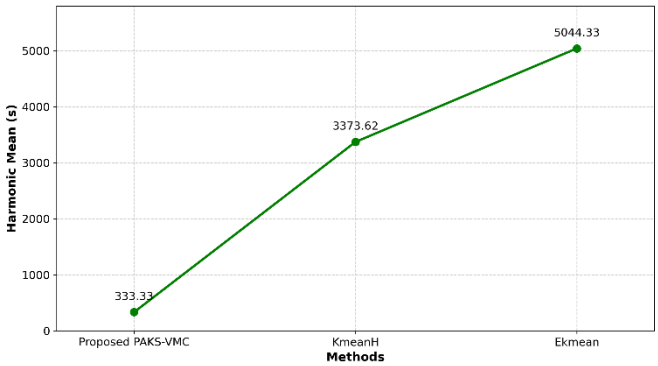

Figure 9. Comparison of harmonic means of average makespan (s) with 16 virtual machines (VMs)

The PAKS-VMC model had the lowest harmonic mean, showing better, more stable performance as the number of tasks increased compared with the baseline models. This helps to explain the relative performance improvement associated with a decrease in average makespan and provides justification for the use of the PAKS-VMC approach in managing resources. Figure 9 presents the harmonic mean of PAKS-VMC against the baseline models. The findings indicated that the PAKS-VMC model was significantly better than the baseline models at minimizing the average makespan across 16 VMs, demonstrating its efficient utilization of available resources. Figure 10 shows the relative reduction (%) in average makespan achieved by PAKS-VMC compared with Ekmean and KmeanH using 16 VMs.

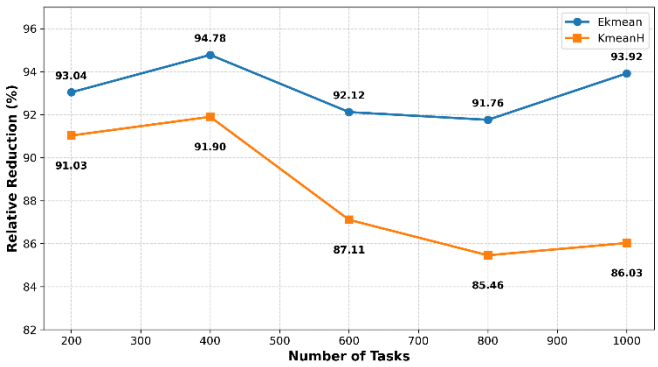

Figure 10. Relative reduction (%) in average makespan with 16 virtual machines (VMs)

4.4.2 Performance evaluation for the 32-VM scenario

Table 14 presents the relative reduction in average makespan achieved by the proposed model compared to baseline clustering-based scheduling models. The proposed

model achieved significant performance improvements over all baseline models, with gains of 84.67% to 91.69% over the KmeanH model and 92.11% to 94.73% over the Ekmean model. Figure 11 illustrates the relative reduction (%) in average makespan achieved by PAKS-VMC compared with Ekmean and KmeanH using 32 VMs.

Table 14. Relative reduction (%) in average makespan using 32 virtual machines (VMs)

No. of Tasks	KmeanH	Ekmean
200	91.13	93.52
400	91.69	94.73
600	85.98	92.11
800	85	93.08
1000	84.67	92.24

Note: KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

Table 15. Harmonic mean of average makespan (s) with 32 virtual machines (VMs)

Methods	Harmonic Mean(s)
Ekmean	4952.66
KmeanH	3137.06
Proposed (PAKS-VMC)	317.35

Note: PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets.

The percentages of relative improvement for both VM configurations are presented in Tables 12 and 14. The results reveal that the rate of improvement is always high, and it validates the effectiveness of the proposed scheduling mechanism for different workloads. Table 15 reports the harmonic mean of the average makespan for each model. The proposed PAKS-VMC model achieves the best performance, exhibiting the lowest harmonic mean when 32 VMs are used. It shows its increased capacity to speed up execution and properly distribute computational load, leading to better overall performance despite potential resource constraints. The harmonic means of PAKS-VMC relative to the baseline models are shown in Figure 12.

This clearly indicates that the proposed PAKS-VMC model is more scalable and adaptable to changes in the number of VMs and their workloads. These relative-reduction percentages are generally higher, and makespan lower, across the different experimental setups, which strongly suggests that the model is capable of effectively balancing the computational loads. As a result, the proposed framework achieves improved system performance.

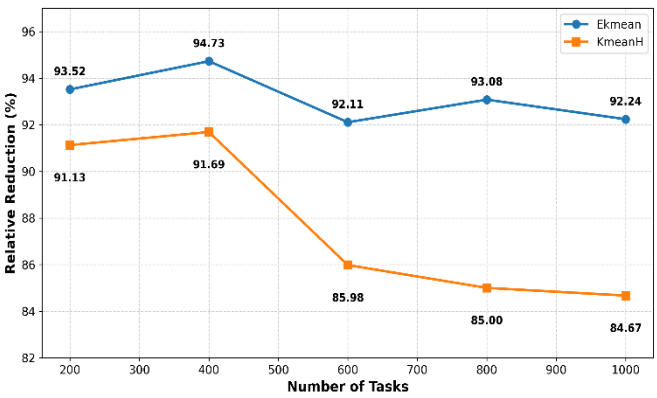

Figure 11. Relative reduction (%) in average makespan with 32 virtual machines (VMs)

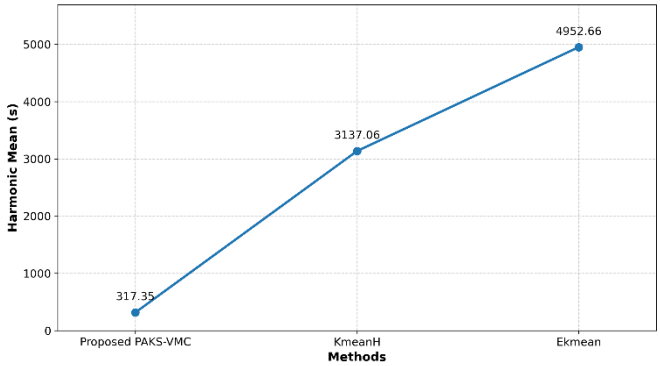

Figure 12. Comparison of harmonic means of average makespan (s) with 32 virtual machines (VMs)

4.3 Paired *t*-test statistical analysis across experimental scenarios

All the observed performance improvements were statistically validated through a paired *t*-test between PAKS-VMC and the two clustering models used in the baseline (KmeanH and Ekmean) in both the evaluated scenarios, 16-VM and 32-VM. The former includes 16 VMs, and the latter includes 32 VMs. The analysis was conducted to determine whether the differences observed across task loads were a true sign of improvement or merely chance effects. The paired *t*-test analysis showed that the performance gap was statistically significant ($p < 0.05$) in both evaluation scenarios, indicating the effectiveness of PAKS-VMC in a constrained-resource scenario. The findings of the paired *t*-test comparing the proposed PAKS-VMC model with baseline clustering-based scheduling models on the key performance indicator, average makespan, in computing environments with 16 and 32 VMs are presented in Table 16.

Table 16. Results of paired statistical analysis for average makespan in 16- and 32-VM scenarios

No. of VMs	Metric	Comparison	*t*-Statistic	*p*-Value
16	Average makespan	PAKS-VMC vs KmeanH	–4.4450	0.0113
		PAKS-VMC vs Ekmean	–3.1526	0.0344
32	Average makespan	PAKS-VMC vs KmeanH	–4.5851	0.0101
		PAKS-VMC vs Ekmean	–3.6194	0.0224

Note: VM = virtual machines, PAKS-VMC = Priority-Aware K-means++ Scheduling of Virtual Machines and Cloudlets, KmeanH = Kmean HEFT, Ekmean = Efficient Kmean.

Table 16 shows that all *p*-values are lower than the significance threshold ($p < 0.05$), indicating that the performance differences between the proposed PAKS-VMC framework and the compared scheduling approaches are statistically significant. Furthermore, all the *t*-statistics calculated for the obtained results are negative, which means that the average makespan values obtained by PAKS-VMC are always less than the average makespan values obtained by KmeanH and Ekmean in every paired comparison. In other words, the differences are not only statistically significant but also directional in favor of the proposed framework. This observation is aligned with the relative reduction analysis in Tables 12 and 14, where PAKS-VMC achieves improvements

exceeding 84%–94% across different task loads and VM counts, and with the harmonic mean results in Tables 13 and 15, which show that PAKS-VMC maintains the lowest overall makespan across both 16 and 32 VMs. These findings validate the observed performance improvements and confirm that they are not due to random variation but are associated with the adaptive scheduling and dynamic clustering mechanisms integrated within the proposed framework. Moreover, the consistency of the statistical significance across various workload sizes and VM configurations demonstrates the robustness, scalability, and adaptability of the PAKS-VMC framework in heterogeneous cloud environments. The *p*-values obtained from the paired *t*-tests are also illustrated in Figures 13 and 14, where all reported values remain below 0.05, further supporting the effectiveness and stability of the proposed scheduling framework. Importantly, statistical significance is obtained in both resource-constrained 16 VMs and higher-capacity 32 VMs scenarios, suggesting that the performance gains offered by PAKS-VMC are robust to changes in VM availability and workload intensity. This robustness supports the claim that the proposed dual-layer scheduling framework is not tuned to a single configuration, but rather generalizes well under varying cloud resource conditions.

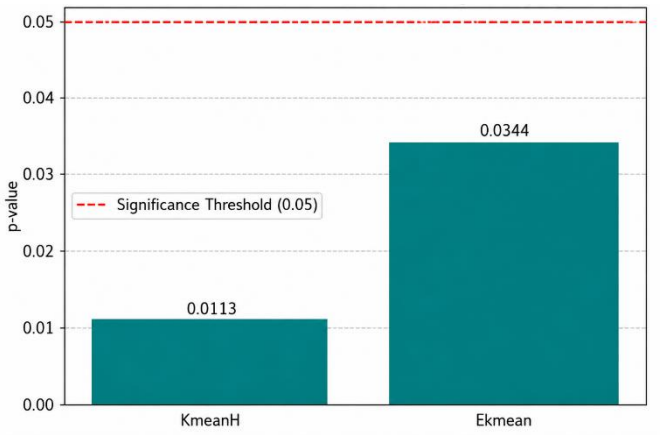

Figure 13. Statistical significance for average makespan with 16 virtual machines (VMs)

Figure 14. Statistical significance for average makespan with 32 virtual machines (VMs)

4.4 Discussion and interpretation

Compared with the baseline clustering-based scheduling methods (Ekmean and KmeanH), the proposed PAKS-VMC framework demonstrates improved scheduling performance under the tested scenarios. Ekmean introduces randomness into the task–VM assignment through stochastic substitutions, which may lead to unstable behaviour across repeated executions and require multiple iterations before convergence. KmeanH combines HEFT-based assignment, which is tailored to static workflow assumptions and has limited flexibility under dynamic resource changes. In contrast, PAKS-VMC employs dual K-means++ clustering of tasks and VMs, dynamic weight updating, a priority-aware level-matching strategy, and a fallback mechanism, yielding more adaptive and resource-aware scheduling.

Experimental results show that PAKS-VMC achieves lower average makespan and better load distribution than Ekmean and KmeanH across 8, 16, and 32 VMs and workloads from 50 to 1,000 cloudlets. The observed improvements are consistent: Relative reduction analysis indicates average makespan reductions ranging from 84.67% to 94.78%, while harmonic mean values confirm that PAKS-VMC maintains the lowest overall makespan across VM configurations. Furthermore, paired *t*-test results indicate that these gains are statistically significant at the 95% confidence level ($p < 0.05$) for all pairwise comparisons, confirming that the performance improvements are both practically meaningful and statistically robust.

From an information system engineering perspective, the improvements in makespan and LB are relevant for enterprise and industrial cloud applications such as healthcare, financial services, and workflow-based business systems. By combining priority-aware task clustering with dynamic VM grouping, PAKS-VMC supports more adaptive RA and reduced execution times under heterogeneous and dynamic workloads. The framework is technically feasible for integration into real cloud management environments, as it relies on commonly available VM monitoring metrics (CPU and memory utilization). However, real platforms may introduce additional variability and infrastructure-level overheads not fully captured in CloudSim, making large-scale real-world validation an important direction for future work.

In addition to scheduling effectiveness, scalability and runtime overhead are also important considerations for practical cloud deployment. The computational complexity of PAKS-VMC is dominated by dual K-means++ clustering and runtime task allocation. Let N_c and N_v denote the numbers of cloudlets and VMs, respectively, and I K-means++ iterations (fixed $k = 3$); the dominant cost is approximately (O (N_c. I + N_v. I)) per scheduling cycle, indicating near-linear scalability. This scalability behavior supports the applicability of the framework under increasing workload demands. The priority-aware level-matching and ULC perform assignment within each VM tier, while the fallback mechanism adds only limited overhead. Although the current experiments cover 50–1,000 cloudlets and 8–32 VMs, repeated runtime re-clustering and centralized VM-weight management may become bottlenecks under highly dynamic workloads. Future work will investigate distributed and mini-batch clustering strategies, parallel scheduling mechanisms, and the integration of energy-aware and SLA-aware optimization objectives.

5. CONCLUSION AND FUTURE WORK

This paper presented PAKS-VMC, an integrated runtime

scheduling framework for heterogeneous cloud environments. The framework uses unsupervised K-means++ clustering to group cloudlets into priority categories based on execution length, and dynamically clusters VMs using resource-aware CPU and RAM weight factors (α and β). In addition, the proposed framework integrates a priority-aware level-matching allocation strategy, a ULC to select the most suitable VM within each tier, and a fallback reallocation mechanism to maintain allocation continuity under resource constraints.

Experimental evaluations across workloads of 50–1,000 cloudlets and 8–32 VMs showed that PAKS-VMC consistently outperformed the baseline clustering-based algorithms (Ekmean and KmeanH). The framework achieves lower average makespan, AExT, AST, and AFT, with relative reductions up to 94.78%, lower harmonic mean values, and improved load distribution. The gains are stable across VM configurations and statistically significant ($p < 0.05$) according to paired *t*-tests.

Despite these results, the framework has limitations. It relies on centralized maintenance of VM weights and cluster memberships, which may become a bottleneck under highly dynamic, large-scale workloads. Runtime clustering and fallback-based reallocation may also introduce additional scheduling overhead and temporary resource contention in environments with thousands of tasks and VMs. Future work will examine this overhead, evaluate scalability and long-term performance on larger distributed cloud infrastructures, and integrate energy-conscious and SLA-aware metrics. Sensitivity analysis of the CPU and RAM weighting parameters (α, β) under diverse workload conditions will also be conducted to further enhance the robustness and applicability of PAKS-VMC.

REFERENCE

[1] Shafiq, D.A., Jhanjhi, N.Z., Abdullah, A. (2022). Load balancing techniques in cloud computing environment: A review. Journal of King Saud University Computer and Information Sciences, 34(7): 3910-3933. https://doi.org/10.1016/j.jksuci.2021.02.007

[2] Oyediran, M.O., Ojo, O.S., Ajagbe, S.A., Aiyeniko, O., Obuzor, P.C., Adigun, M.O. (2024). Comprehensive review of load balancing in cloud computing system. International Journal of Electrical & Computer Engineering, 14(3): 3244-3255. https://doi.org/10.11591/ijece.v14i3.pp3244-3255

[3] Rajammal, K., Chinnadurai, M. (2025). Dynamic load balancing in cloud computing using predictive graph networks and adaptive neural scheduling. Scientific Reports, 15(1): 22181. https://doi.org/10.1038/s41598-025-97494-2

[4] Aron, R., Abraham, A. (2022). Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence. Engineering Applications of Artificial Intelligence, 116: 105345. https://doi.org/10.1016/j.engappai.2022.105345

[5] Malallah, H.S., Qashi, R., Abdulrahman, L.M., Omer, M.A., Yazdeen, A.A. (2023). Performance analysis of enterprise cloud computing: A review. Journal of Applied Science and Technology Trends, 4(1): 1-12. https://doi.org/10.38094/jastt401139

[6] Nadeem, S., Amin, N.U., Zaman, S.K.U., et al. (2023). Runtime management of service level agreements through proactive resource provisioning for a cloud environment. Electronics, 12(2): 296. https://doi.org/10.3390/electronics12020296

[7] Madni, S.H.H., Faheem, M., Younas, M., Masum, M.H., Shah, S. (2024). Critical review on resource scheduling in IaaS clouds: Taxonomy, issues, challenges, and future directions. The Journal of Engineering, 2024(8): e12420. https://doi.org/10.1049/tje2.12420

[8] Shukur, H., Zeebaree, S., Zebari, R., Zeebaree, D., Ahmed, O., Salih, A. (2020). Cloud computing virtualization of resources allocation for distributed systems. Journal of Applied Science and Technology Trends, 1(2): 98-105. https://doi.org/10.38094/jastt1331

[9] Devi, N., Dalal, S., Solanki, K., et al. (2024). A systematic literature review for load balancing and task scheduling techniques in cloud computing. Artificial Intelligence Review, 57(10): 276. https://doi.org/10.1007/s10462-024-10925-w

[10] Mukundha, C., Venkatesh, N., Akshay, K. (2017). A comprehensive study report on load balancing techniques in cloud computing. International Journal of Engineering Research and Development, 13(9): 35-42.

[11] Syed, D., Muhammad, G., Rizvi, S. (2024). Systematic review: Load balancing in cloud computing by using metaheuristic based dynamic algorithms. Intelligent Automation and Soft Computing, 39(3): 437-476. https://doi.org/10.32604/iasc.2024.050681

[12] Awad, W.K., Ariffin, K.A.Z., Nazri, M.Z.A., Yassen, E.T. (2025). Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review. Journal of Intelligent Systems, 34(1): 20240441. https://doi.org/10.1515/jisys-2024-0441

[13] Rout, S., Patra, S.S., Patel, P., Sahoo, K.S. (2020). Intelligent load balancing techniques in software defined networks: A systematic review. In 2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC), Gunupur Odisha, India, pp. 1-6. https://doi.org/10.1109/iSSSC50941.2020.9358873

[14] Alrammahi, M.A.M., Bhaya, W.S. (2023). A state-of-the-art survey and taxonomy of classification, algorithms, and techniques for load balancing in SDN. Journal of Al-Qadisiyah for Computer Science and Mathematics, 15(3): 164-182. https://doi.org/10.29304/jqcm.2023.15.3.1273

[15] Omar, H.K., Jihad, K.H., Hussein, S.F. (2021). Comparative analysis of the essential CPU scheduling algorithms. Bulletin of Electrical Engineering and Informatics, 10(5): 2742-2750. https://doi.org/10.11591/eei.v10i5.2812

[16] Omranian, S.A., Goudarzi, M. (2025). Greedy algorithm for dynamic allocation of intelligent services in vehicular edge computing. Cluster Computing, 28(1): 48. https://doi.org/10.1007/s10586-024-04817-5

[17] Sharma, A., Sharma, K.K. (2023). Cloud computing: Hybrid load balancing algorithm proposal. International Journal of Intelligent Systems and Applications in Engineering, 11(10s): 859-864. https://www.ijisae.org/index.php/IJISAE/article/view/3356.

[18] Syed, D., Muhammad, G., Ali, S. (2026). A highly scalable improved multi-objective hybrid load balancing (IMH_LB) algorithm. ICT Express, 12(2): 275-282.

https://doi.org/10.1016/j.icte.2025.06.018
[19] Niyasudeen, F., Mohan, M. (2025). Designing a hybrid load balancing algorithm for optimized resource allocation in cloud environments using python. Journal of Information Systems Engineering and Management, 10(20s): 264-276.
[20] Narwal, A. (2024). Resource utilization based on hybrid WOA-LOA optimization with credit based resource aware load balancing and scheduling algorithm for cloud computing. Journal of Grid Computing, 22(3): 61. https://doi.org/10.1007/s10723-024-09776-0
[21] Alla, V.R.S.P., Medikondu, N.R., Parige, L.S., et al. (2024). Optimizing task scheduling in cloud computing: A hybrid artificial intelligence approach. Cogent Engineering, 11(1): 2328355. https://doi.org/10.1080/23311916.2024.2328355
[22] Flayyih, K.H., Nickray, M. (2024). Energy-efficient clustering in wireless sensor networks using metaheuristic algorithms. Edelweiss Applied Science and Technology, 8(6): 8582-8610. https://doi.org/10.55214/25768484.v8i6.3848
[23] Hosseini, E., Nickray, M., GH, S. (2023). Priority-based task scheduling using fuzzy system in mobile edge computing. Nashriyyah-i Muhandisi-i Barq va Muhandisi-i Kampyutar-i Iran, 100(4): 257.
[24] Singh, P., Prakash, V., Bathla, G., Singh, R.K. (2022). QoS aware task consolidation approach for maintaining SLA violations in cloud computing. Computers and Electrical Engineering, 99: 107789. https://doi.org/10.1016/j.compeleceng.2022.107789
[25] Ghandour, O., El Kafhali, S., Hanini, M. (2024). Adaptive workload management in cloud computing for service level agreements compliance and resource optimization. Computers and Electrical Engineering, 120: 109712. https://doi.org/10.1016/j.compeleceng.2024.109712
[26] Kapetanidou, I.A., Sarros, C.A., Ledakis, G., Tsaoussidis, V. (2025). Feed4Cloud: Towards trustworthy QoE-aware cloud service monitoring using blockchain. Future Generation Computer Systems, 163: 107532. https://doi.org/10.1016/j.future.2024.107532
[27] Al-hamami, M.Y., Nickray, M. (2025). KMPPVM-DRM: A K-means++ based dynamic clustering approach for virtual machine resource allocation in cloud data centers. Informatica, 49(31). https://doi.org/10.31449/inf.v49i31.8761
[28] Shankar, J., Hussain, I., Zafar, S., Khan, I.R., Khalique, A. (2024). Effective resource allocation and load balancing in green cloud computing. In International Conference on ICT for Digital, Smart, and Sustainable Development, pp. 423-439. https://doi.org/10.1007/978-981-97-7831-7_26
[29] Chowdhary, S.K., Rao, A.L.N. (2023). A task clustering based QoS aware scheduling algorithm for task execution in cloud-IoT model for education services. Multimedia Tools and Applications, 82(29): 44783-44800. https://doi.org/10.1007/s11042-023-15392-z
[30] Muthusamy, G., Chandran, S.R. (2021). Cluster-based task scheduling using K-means clustering for load balancing in cloud datacenters. Journal of Internet Technology, 22(1): 121-130.
[31] Sohani, M., Jain, S.C. (2021). A predictive priority-based dynamic resource provisioning scheme with load balancing in heterogeneous cloud computing. IEEE Access, 9: 62653-62664. https://doi.org/10.1109/ACCESS.2021.3074833
[32] Meyer, V., Kirchoff, D.F., Da Silva, M.L., De Rose, C.A. (2021). ML-driven classification scheme for dynamic interference-aware resource scheduling in cloud infrastructures. Journal of Systems Architecture, 116: 102064. https://doi.org/10.1016/j.sysarc.2021.102064
[33] Meyer, V., da Silva, M.L., Kirchoff, D.F., De Rose, C.A. (2022). IADA: A dynamic interference-aware cloud scheduling architecture for latency-sensitive workloads. Journal of Systems and Software, 194: 111491. https://doi.org/10.1016/j.jss.2022.111491
[34] Mustapha, S.D.S., Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. Internet of Things and Cyber-Physical Systems, 4: 32-39. https://doi.org/10.1016/j.iotcps.2023.07.001
[35] Elsakaan, N., Amroun, K. (2024). A novel multi-level hybrid load balancing and tasks scheduling algorithm for cloud computing environment. The Journal of Supercomputing, 80(9): 13434-13474. https://doi.org/10.1007/s11227-024-05990-5
[36] Ghasemi, A., Keshavarzi, A. (2024). Energy-efficient virtual machine placement in heterogeneous cloud data centers: A clustering-enhanced multi-objective, multi-reward reinforcement learning approach. Cluster Computing, 27(10): 14149-14166. https://doi.org/10.1007/s10586-024-04657-3
[37] Lwin, J. (2024). Enhancing cloud task scheduling with multi-objective optimization using K-means clustering and dynamic resource allocation. Journal of Computing Theories and Applications, 2(2): 202-211. https://doi.org/10.62411/jcta.11337
[38] Alsubaei, F.S., Hamed, A.Y., Hassan, M.R., Mohery, M., Elnahary, M.K. (2024). Machine learning approach to optimal task scheduling in cloud communication. Alexandria Engineering Journal, 89: 1-30. https://doi.org/10.1016/j.aej.2024.01.040
[39] Zhu, Q. (2025). Autonomous cloud resource management through DBSCAN-based unsupervised learning. Optimizations in Applied Machine Learning, 5(1): 5. https://doi.org/10.71070/oaml.v5i1.112
[40] Lwin, J., Thaw, T.Z. (2025). Leveraging k-means clustering for multi-objective task scheduling in cloud-based scientific workflows. In 2025 IEEE Conference on Computer Applications (ICCA), Yangon, Myanmar, pp. 1-6. https://doi.org/10.1109/ICCA65395.2025.11011149
[41] Alharbe, N.R. (2025). Fuzzy clustering based scheduling algorithm for minimizing the tasks completion time in cloud computing environment. Scientific Reports, 15(1): 19505. https://doi.org/10.1038/s41598-025-02654-z
[42] Alworafi, M.A., Dhari, A., Al-Hashmi, A.A., Darem, A.B. (2016). An improved SJF scheduling algorithm in cloud computing environment. In 2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT), Mysuru, India, pp. 208-212. https://doi.org/10.1109/ICEECCOT.2016.7955216

NOMENCLATURE

α	The weight of the relative contribution of the CPU resource $\alpha = 0.6$.

β	Tunable weight factor for RAM contribution, where β = 0.4.
Cpu_{vm}	The virtual machine (VM) CPU resource capacity (in MIPS).
Ram_{vm}	The virtual machine (VM) RAM resource capacity.
$VM_{assigned}$	The incoming task is assigned to the virtual machine (VM).
VM_i	A particular virtual machine (VM) of the cluster C_j.
C_j	The group of virtual machines (VMs).
W_{vmi}	The weight value of the VM_i.
$argmax$	The function selects the virtual machine (VM) with the highest weight.
VM_{alt}	An alternative virtual machine (VM) is chosen by the fallback mechanism.
C_{avail}	The set of available virtual machines (VMs) of the remaining virtual machine (VM) tiers after re-clustering.