WORKLOADAWARE: A Resource Allocation and Consolidation Technique for Heterogeneous Clouds

Muhammad Zakarya, Ayaz Ali Khan, Lee Gillam, Omer Rana, Rajkumar Buyya

Abstract—Datacenters are the core of cloud computing services that consume a significant amount of energy and negatively affect our environment. These problems can be solved using efficient resource management techniques, such as resource allocation and consolidation methods. This paper models the resource allocation problem as a bin-packing NP-hard problem and proposes a workload-aware resource allocation and consolidation method that accounts for the heterogeneity of resources and applications. Using plausible assumptions and real datasets from Google cluster traces, we investigated the impact of workload-aware scheduling and migration methods on infrastructure energy efficiency and workload performance. We found that our suggested method can provide around 5.25% energy savings and 8.17% workload performance gains on 12,583 heterogeneous servers and over three million tasks across three distinct applications. Moreover, a workload-aware migration technique can lower energy consumption (6.42% – 18.3%) and increase performance (7.8% – 25.03%) for specific applications, despite the fact that we found an existing trade-off between energy consumption and performance. Additionally, we observed that workload awareness may significantly lower the overall number of migrations (5.47% – 37.58%), which has no adverse effect on application performance.

Index Terms—Resource allocation, workload-aware algorithms, VM migration, energy consumption, performance.

1 INTRODUCTION

Datacenters are the core of cloud computing that provide various services related to Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) to users. According to estimates, datacenters will consume around 4.5% of the world's power consumption by 2025, which is equal to over 1,000 TWh, or almost 1 trillion tons of coal. Moreover, in the worst-case scenario, they are projected to use around 7,933 TWh by 2030 [1]. Due to their huge energy consumption, they also affect our environment in terms of carbon emissions. Furthermore, cloud services suffer from various types of workloads based on computational resource demands, i.e., compute-intensive, data-intensive, storage-intensive, batch processing, and machine learning workloads. Due to the heterogeneous nature of resources, similar workloads may run quite differently on different hardware or virtual machines (VMs). Subsequently, resources may be left unused or underutilized, consuming significant energy. Furthermore, the performance of workloads can also be affected by different resources, leading to increased runtimes and, therefore, users' costs. Such problems are usually solved with appropriate resource allocation and management techniques. Effective resource management in cloud systems requires an understanding of workloads and their requirements. This can lead to reduced costs, improved performance, and energy efficiency by customizing resource allocation, scalability, and performance methods to resource demands of workloads [2].

In the literature, various resource management techniques are used to minimize the energy consumption of datacenters in three different ways: (i) centralized; (ii) hierarchical; and (iii) decentralized. In centralized approaches, all resources in the datacenter are managed by a single scheduler. Centralized resource allocation methods have significant latencies, scalability problems, and a single point of failure. When dealing with large cluster sizes and VMs, centralized techniques might become a bottleneck in appropriate resource management. A hierarchical technique is used in situations where a global scheduler communicates with several local schedulers that are distributed across multiple levels, with each level managing a subset of resources. In this setup, the global scheduler can become a single point of failure and has scalability problems. To overcome these problems, decentralized approaches are used, where decision-making is spread across multiple independent nodes that work collaboratively without a central authority. Furthermore, the entire cluster of resources is divided into small sub-clusters so that each cluster has its own local scheduler and there is no global scheduler. By dividing resource management and decision-making into several nodes, a decentralized method may alleviate the concerns related to hierarchical and centralized methods [3].

In previous works, resource allocation is assumed to be a bin-packing problem and solved using heuristic methods. Approaches like First Fit (FF), Best Fit (BF), First Come First Serve (FCFS), Worst Fit (WF), and FillUp (and many more) have been modified to account for various objectives of the allocation problem [3]. Current resource allocation and migration approaches do not consider workloads and their types (requirements) in resource management decisions with the notable exception of approaches discussed in Sec. 6. Furthermore, various workload prediction techniques have been suggested [4], [5], [6], [7]; however, they are sensitive to prediction accuracy, necessitating more sophisticated models, and they have trouble identifying the best placement options. These approaches usually use general strategies that do not adjust to the changing needs of various workloads, which results in less-than-ideal use of resources. Allocating resources without taking into account the workload's characteristics — such as whether it is compute-intensive, memory-heavy, or latency-sensitive — may lead to wasteful energy use, decreased performance, or higher operating expenses. Therefore, it is crucial to correctly identify and categorize workloads based on their requirements and historical usage. This makes it possible for the system to adjust its

- *M. Zakarya is with the Faculty of Computing and Information Technology, Sohar University, Sultanate of Oman and with the College of Computing and Intelligent Systems, University of Khorfakkan, UAE. A.A. Khan is with the Department of Computer Science, University of Lakki Marwat, Pakistan. L. Gillam is with the Department of Computer Science, University of Surrey, UK. O. Rana is with the School of Computer Science and Informatics, Cardiff University, UK. R. Buyya is with the University of Melbourne, Australia. E-mail(s): MZakarya@su.edu.om, muhammad.zakarya@ukf.ac.ae, ayazkhan@ulm.edu.pk, l.gillam@surrey.ac.uk, ranaof@cardiff.ac.uk, rbuyya@unimelb.edu.au*

*Corresponding authors: M. Zakarya

migration and allocation plans to suit the particular requirements of every task, leading to a more equitable trade-off between cost-effectiveness, system performance, and energy efficiency [8], [9].
In this paper, we propose a resource allocation technique that accounts for workloads and the heterogeneity of resources in placement decisions. The proposed method is scalable and is implemented in a centralized manner, and it can easily be modified for decentralized systems. The following are the major contributions of our work:

- a workload-aware resource allocation policy that accounts for the heterogeneity of resources;
- a machine learning approach to predict future workloads that enables proactive allocation and migration decisions;
- a proactive workload-aware migration policy to improve performance of the workloads; and
- a framework to integrate workload clustering, classification, and prediction in resource allocation policies.

Our approach considers workload classification, workload prediction, and workload-aware allocation/migration algorithms in a heterogeneous datacenter. We present: (i) workload-aware migration policies, (ii) a composite metric N_T to capture energy-performance-migration trade-offs, and (iii) validation on a large-scale heterogeneous datacenter with different CPU architectures, workloads, and AWS-like VM types, in contrast to previous works that study prediction or migration in isolation. The rest of the paper is structured as follows. Sec. 2 is where we model the problem. Sec. 3 is where we talk about the suggested method for allocating VMs. Experimental setup and evaluation metrics are explained in Sec. 4. Using actual workload traces from Google clusters in Sec. 5, we evaluate the suggested approach. The relevant work is summarized in Sec. 6. Finally, the paper's conclusions and a list of potential future study areas are provided in Sec. 7.

2 Problem Definition

The main objective of the resource allocation process is to accommodate applications (VMs) onto a minimum number of servers to reduce energy consumption (as shown in Fig. 1). Since servers are bins and VMs are things, this optimization challenge may be thought of as a bin-packing problem [2]. VM consolidation and allocation may be thought of as NP-hard bin-packing issues (Appendix A). Since there are no ideal solutions for NP-hard problems, many heuristic techniques, including worst fit and first fit, are seen to be appropriate [10]. While ensuring that the allotted VMs do not consume more resources than the hosts can supply, the goal is to minimize the number of physical hosts required. Resources like CPU, memory, storage, and network bandwidth are often considered. Table 1 provides a brief description of various mathematical notations used in our problem formulation.
Suppose we have n real hosts and m VMs. Binary decision variables can be used to indicate if a VM is linked to a certain physical host. Let's say that x_{ij} is a binary variable that satisfies Eq. 1:

$$x_{ij} = \begin{cases} 1 & \text{if VM } i \text{ is allocated to server } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A given host's capacity cannot be exceeded by the total resource use of the allocated VMs. Three resources are taken into consideration in this paper: CPU, memory (M), and I/O. The capacity limitations for each physical host, represented by j, and VM, represented by i, are described by the following Eq. 2:

$$\begin{aligned} \sum_i x_{ij}\,\mathrm{CPU}_i &\le \mathrm{CPU}_j, \quad \forall j \in \{1,\dots,n\}, \\ \sum_i x_{ij}\,\mathrm{M}_i &\le \mathrm{M}_j, \quad \forall j \in \{1,\dots,n\}, \\ \sum_i x_{ij}\,\mathrm{IO}_i &\le \mathrm{IO}_j, \quad \forall j \in \{1,\dots,n\}. \end{aligned} \tag{2}$$

TABLE 1: List of notations with brief description

Notation	Description
W	Set of workloads
W_{CPU}, W_{MEM}	CPU, memory workloads
W_{IO}	I/O workload
V	Set of VMs
H	Set of hosts
C_i	Capacity of host i
U_i	Utilization of host i
M_v	Memory demand of VM v
C_v	CPU demand of VM v
IO_v	I/O demand of VM v
S_{ij}	Ranking score of host j for VM i
y_j	Host-active variable; $y_j = 1$ if host j is active
x_{ij}	VM–host map; $x_{ij} = 1$ if VM i mapped to host j
T_{mig}	Migration time
M_{mig}	Migrated VM memory
C_{mig}	Migration cost
C_{net}	Network cost
C_{comp}	Computational cost
E	Total energy consumption
P	Performance metric
E_{base}, E_{WA}	Energy (baseline, WA)
P_{base}, P_{WA}	Performance (baseline, WA)
$\Delta E, \Delta P$	Energy/performance gain
A	Prediction accuracy
μ	Mean value
E_f	Efficiency factor for each host
$\omega_{Resource}$	Workload specific parameter
σ^2	Variance

where the corresponding resource needs of VM_i are represented by CPU_i, $Memory_i$, and IO_i. Additionally, each VM must be allocated to exactly one physical host. This may be expressed formally in Eq. 3 as follows:

$$\sum_j x_{ij} = 1 \quad \text{for all } i \tag{3}$$

Eq. 2 can be written in a vectorized form as:

$$\sum_i x_{ij}\,\mathbf{d}_i \;\le\; \mathbf{C}_j, \quad \forall j, \tag{4}$$

where $\mathbf{d}_i = (CPU_i, M_i, IO_i)$ and $\mathbf{C}_j = (CPU_j, M_j, IO_j)$. Additional limitations can also be imposed to guarantee compatibility if some VMs have particular needs or dependencies that must be met by compatible hosts (specific allocations). For example, in our proposed method (Sec. 3), workloads are assigned to hosts based on workload type and ranking score. To explicitly denote whether a host is active, we use a binary host-activation parameter y_j, where $y_j = 1$ if host j is used to accommodate at least one VM and $y_j = 0$ otherwise. The VM–host allocation parameter x_{ij} (Eq. 1) denotes whether VM i is allocated to host j. The relationship between VM allocation and host activation parameters is enforced using Eq. 5:

$$x_{ij} \le y_j, \qquad \forall i, j. \tag{5}$$

Thus, a VM can only be allocated to an active host. The main objective of our allocation problem is to minimize the number of active hosts; thus, the optimization objective is formulated as in Eq. 6:

$$\min \sum_{j=1}^{n} y_j. \tag{6}$$

This formulation captures the bin-packing objective as described above, where hosts denote bins and VMs denote the items to be packed. The capacity constraints (Eq. 2) and the assignment constraint (Eq. 3), together with Eq. 5, ensure that each VM is mapped to exactly one active host without exceeding the available CPU, memory, or I/O capacity. The WorkloadAware approach (Sec. 3) is used as a heuristic to get an efficient solution to this NP-hard optimization problem. In particular, S_{ij} denotes the workload-aware ranking score of host j for VM i (Eq. 7). Rather than treating

S_{ij} as a host-activate parameter, WorkloadAware uses S_{ij} to rank feasible active/candidate hosts and chooses the host with the highest score while ensuring the capacity constraints. Therefore, y_j denotes host activation in our formulation, whereas x_{ij} denotes VM assignment and S_{ij} denotes the host-selection score.

Energy efficiency may be increased by minimizing the number of hosts or optimizing the usage of physical resources by characterizing VM allocation as a bin-packing problem and using appropriate algorithms. We used XGBoost to predict resource demands and a rule-based model to cluster and categorize workloads, classifying tasks into CPU-intensive, memory-intensive, I/O-intensive, and mixed categories. Because of its great accuracy and resilience to heterogeneous cloud traces, this model was chosen. We next employ our workload-aware allocation and migration methods with the categorized workload types as inputs. To estimate future workload demands, besides XGBoost, we also used other regression-based predictors, such as Artificial Neural Networks (ANNs), Support Vector Regression (SVR), and Linear Regression (LR) – as described in Sec. 3.3.

3 The Proposed Work

3.1 VM Allocation

The proposed workload-aware resource allocation method is illustrated in Alg. 1. We make the assumption that the workloads are either CPU-intensive, memory-intensive, or I/O-intensive. We use machine learning-based methods to categorize incoming workloads into one of these three types. In addition, we assume that each host is profiled according to its use history and available resources to determine the kinds of workloads it can manage most effectively. Optimizing usage to increase energy efficiency, minimizing resource fragmentation (from job termination), and ensuring workload performance are the objectives of the method [11]. Based on its needs, we assign each VM to one of the three workload categories. Based on the type of workload and current VM resource use, each host determines a ranking score. Higher ratings will go to hosts with greater resources available that are pertinent to the type of workload of the VM. The algorithm assigns the VM to the first host with sufficient resources and a high-ranking score. The ranking score is computed for each type of workload based on demand and available resources on a particular host using Eq. 7.

$$S_i = \omega_{Resource} \left(\frac{host_{AResource}}{host_{TResource}} \right) \tag{7}$$

where $host_{AResource}$ and $host_{TResource}$ refer to available and total resources in terms of CPU, memory, and I/O that denote three different types of workloads. Furthermore, $\omega_{Resource}$ denotes a workload-specific parameter for each resource type. $\omega_{Resource}$ is not a single number, but rather a set of resource-specific weights.

$$S_i = \omega^{\text{CPU}} \left(\frac{host_{A\_CPU}}{host_{T\_CPU}} \right) + \omega^{\text{M}} \left(\frac{host_{A\_M}}{host_{T\_M}} \right) + \omega^{\text{IO}} \left(\frac{host_{A\_IO}}{host_{T\_IO}} \right) \tag{8}$$

Here, $\omega^{\text{CPU}}, \omega^{\text{M}}, \omega^{\text{IO}}$ represent workload-specific weights for CPU, memory, and I/O resources, respectively. For example, a CPU-intensive workload has $\omega^{\text{CPU}} = 1$ and the others set close to zero, ensuring that the scoring function reflects the workload's dominant resource demand. Furthermore, $\omega^{\text{CPU}}, \omega^{\text{M}}, \omega^{\text{IO}} \geq 0$ and $\omega^{\text{CPU}} + \omega^{\text{M}} + \omega^{\text{IO}} = 1$. These parameters encode workload-specific emphasis: CPU-intensive workloads have $\omega^{\text{CPU}} \approx 1$, memory-intensive workloads have $\omega^{\text{M}} \approx 1$, and so on. Eq. 8 is not restricted to binary weights and can be extended to gradient or proportional values (e.g., ω^{CPU}=0.6, ω^{M}=0.4) to represent mixed workloads. We use the min-max normalization technique (Eq. 9) to normalize the scores in the range [0, 1] for easy comparison [2].

$$S_i^{\text{norm}} = \frac{S_i - S_{\min}}{S_{\max} - S_{\min}} \tag{9}$$

For the FillUp approach, we compute an efficiency factor E_f for each host and then make sure that efficient hosts are allocated first [2], [12]. The E_f simply relates to the total energy use and/or performance of a virtualized host; and is computed with Eq. 10:

$$E_f = \frac{E_{source} \times P_{source}}{E_{target} \times P_{target}} \tag{10}$$

To migrate VMs using the FillUp+WA approach, we use E_f of each host to identify whether the target host is more or less energy efficient than the source host. We prefer migrating to target hosts that are either more energy efficient than the source or, at least, have the same energy efficiency as the source host. The algorithm sorts hosts by descending order of the ranking score S_i and E_f for the given workload type. If $S_i >= T$, $E_f^{target} >= E_f^{source}$, and the host has sufficient resources to accommodate the VM, the host is selected for placement. In case there is no host that meets the placement criterion, we apply a backfilling algorithm without the workload-specific ranking score. Further details on how the energy consumption and performance of each host, VM, and/or workload are computed (statistically modeled) are given in [13], [2], [14].

From an implementation point of view, the above workload-aware algorithm was integrated into FCFS, FF, and FillUp approaches, leading to FCFS+WA, FF+WA, and FillUp+WA, respectively. While the current implementation of the WorkloadAware framework processes VMs in batches for evaluation purposes, the underlying algorithms are naturally compatible with online scheduling. In an online scenario, VMs arriving continuously can be classified and assigned to hosts in real time using the same workload-aware ranking and resource-efficiency criteria. The ranking score computation and allocation steps can be executed incrementally for each incoming VM, while the migration policy can run periodically to rebalance workloads across hosts. This ensures that WorkloadAware can maintain performance and energy efficiency in dynamic environments, supporting real-time decision-making without relying on a static workload set.

Alg. 1 takes a threshold T, a host list H, and a VM list V (Line 1). For each VM, it predicts the resource usage vector $\hat{\mathbf{d}}_v$ (CPU, memory, I/O) using regression models (ANN, XGBoost, LR, SVR) or recent usage if no history exists (Line 3), and classifies the VM into a workload type (CPU, MEM, I/O) via a rule-based classifier (Line 5–6). The algorithm computes a workload-aware score S_i for all hosts (Line 8), sorts hosts by S_i and breaks ties using energy efficiency E_f (Line 10–11). For each VM, it iterates over hosts to allocate the VM if $S_i > T$ and sufficient capacity exists; otherwise, the VM is added to a queue Q for later scheduling (Lines 16–20). Finally, all queued VMs are scheduled (Line 22), producing the allocated VM list. This ensures dynamic, workload-aware allocation while supporting deferred placement.

3.2 VM Migration

The proposed VM migration policy is illustrated in Alg. 2. This algorithm migrates VMs from overloaded and underloaded hosts so that similar workloads are not placed on the same host. The migration algorithm runs every five minutes to optimize the datacenter. While a longer interval might postpone essential load balancing, a shorter interval raises migration frequency, resulting in higher communication and performance overhead. The five-minute duration is in line with earlier VM consolidation studies (e.g., [15], [2]), which often used intervals of three to ten minutes. Furthermore, in our initial analysis, we found that cutting the

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2026.3734089

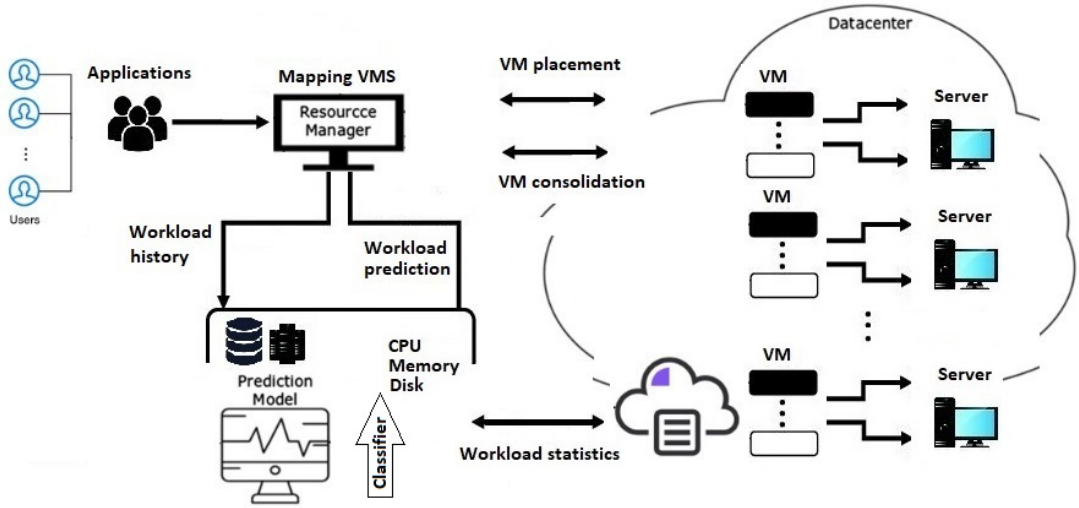

Fig. 1: Workload-aware VM allocation and consolidation framework

Algorithm 1: Workload-Aware Allocation Algorithm

Input: $T \leftarrow$ Threshold, $H \leftarrow$ Host list, $V \leftarrow$ VM list
Predict resource demand vector $\hat{\mathbf{d}}_v = (\widehat{\text{CPU}}_v, \widehat{\text{M}}_v, \widehat{\text{IO}}_v)$ using ANN/XGBoost/LR/SVR (or use recent observed usage if no history) ;
$type_v \leftarrow \text{classify}_{vm}(\hat{\mathbf{d}}_v)$;
Estimate score (S_i) of all hosts $\in H$ (Eq. 7) ;
Sort hosts H in decreasing order of their scores S_i ;
Break ties by higher E_f ;
for *each VM* $\in V$ **do**
  Classify VM type $\in$ {CPU, MEM, I/O} ;
  for *each host* $\in H$ **do**
    Compute S_i for VM ;
    if $S_i > T$ *& host has enough capacity to accommodate the VM* **then**
      allocate host ;
    else
      pick the next VM $\in V$;
      add VM to queue Q ;
    end if
  end for
  Schedule all VMs $\in Q$;
end for
Output: Allocate VMs

Algorithm 2: Workload-Aware Migration Algorithm

Input: $H \leftarrow$ Host list, $V \leftarrow$ VM list
$R(h) \leftarrow$ Remaining capacity of host $h \in H$;
$W(h) \leftarrow$ Workload types on host $h \in H$;
$U(v) \leftarrow$ Resource usage of VM $v \in V$;
$W(v) \leftarrow$ Workload type of VM $v \in V$;
$H(v) \leftarrow$ Current host of VM $v \in V$;
$T_{\text{low}}, T_{\text{high}} \leftarrow$ Thresholds for overload and underload ;
$H_{\text{overloaded}} \leftarrow \{h \in H \mid R(h) < T_{\text{low}}\}$;
$H_{\text{underloaded}} \leftarrow \{h \in H \mid R(h) > T_{\text{high}}\}$;
Sort H in decreasing order of their efficiency E_f ;
for $h_i \in H_{overloaded}$ *or* $H_{underloaded}$ **do**
  $V(h_i) \leftarrow \{v \in V \mid H(v) = h_i\}$;
  Sort $V(h_i)$ by $U(v)$ in ascending order ;
  for $v \in V(h_i)$ **do**
    migrated $\leftarrow$ false ;
    for $h_k \in H \setminus \{h_i\}$ **do**
      if $E_f^{target} >= E_f^{source}$ *and* $R(h_k) \geq U(v)$ *and* $W(v) \notin W(h_k)$ *(strict) or* $|W(h_k) \cup \{W(v)\}| \leq k$ *(relaxed)* **then**
        Migrate v from h_i to h_k ;
        Update hosts accordingly ;
        migrated $\leftarrow$ true ;
        break ;
      end if
    end for
    if *migrated* **then**
      break ;
    end if
  end for
end for

period to less than five minutes led to a considerable increase of migrations but only slight performance improvements. We prefer migrating smaller VMs first, as they are easier to migrate (less migration time) and pack well without causing further imbalances. Previous works have shown that similar workloads often compete for the same resources; therefore, the performance of the workloads is affected due to co-location [16]. To ensure workload performance, we introduce two constraints: (i) using strict constraints, a host will not run VMs with similar workloads; and (ii) in relaxed constraints, a host is allowed to run a specified number of VMs with similar workloads. This is due to the fact that the same workloads compete for similar resources and can degrade workload performance [2]. Note that hosts with remaining capacity $R(h)$ below a predefined threshold T_{low} are considered overloaded, while hosts with remaining capacity $R(h)$ higher than a predefined threshold T_{high} are considered underutilized. Moreover, to ensure energy efficiency, we do not migrate VMs to hosts that are less energy efficient than the source hosts [2].

From an implementation point of view, the above workload-aware migration algorithm was used as: (i) WAMS - workload aware migration (strict); and (ii) WAMR - workload aware migration (relaxed). In the former approach, a host is not allowed to run VMs with the same workloads; while in the latter approach, we assume that only two VMs (on a particular host) are allowed to run the same workloads. This methodology assures performance efficiency because the same workloads often compete for the same resources that have negative impacts on the workload performance. Migrations are also expensive from an energy consumption point of view due to the fact that for the duration of the migration there are two VMs running — one on the source and another on the target host. Further details on migration energy consumption and how the performance of hosts and workloads are affected with co-location or resource contention due to migrations are elaborated in [13], [17], [18].

Alg. 2 takes a host list H and VM list V (Line 1). For each host, remaining capacity $R(h)$ and current workload types $W(h)$ are initialized, while for each VM, resource usage $U(v)$, workload type $W(v)$, and current host $H(v)$ are recorded (Lines 2–6). Thresholds T_{low} and T_{high} classify hosts as overloaded or underloaded (Lines 7–9). Hosts are sorted by energy efficiency E_f (Line 10) [13]. For each host in the overloaded or underloaded sets, VMs on the host are collected and sorted by usage (Lines 11–13). Each VM is migrated to a target host if it satisfies capacity, efficiency, and workload-type constraints (Lines 14–22), updating host states accordingly. This ensures workload-aware migration, balancing load, improving energy efficiency, and avoiding co-location of similar workloads. The condition "If migrated then break" is meant to restrict the amount of migrations per host in each execution cycle. The process runs periodically; this prevents excessive migration overhead. Multiple VMs can migrate from the same host in a single cycle if a more aggressive approach is preferred by omitting the final break line. VM migrations are expensive and migrating many VMs per cycle could produce thrashing and network congestion, particularly, with 12,583 hosts and millions of VMs [2].

3.3 Workload Prediction

For workload-aware resource management, it is essential to predict current and future resource demands and understand their workload characteristics in order to assign VMs to energy- and performance-efficient hosts. Our method presents a hybrid predictive pipeline that specifically takes host architecture and VM heterogeneity into consideration, whereas previous research frequently relies on static workload classification or reactive allocation. The proposed workload prediction and classification process uses a single, sequential pipeline that comprises four stages: (i) extract resource-utilization features from workload traces; (ii) predict future CPU, memory, and I/O resource demand using regression models; (iii) convert the predicted resource demands into interpretable workload classes using a rule-based classifier; and (iv) use the workload class and predicted resource demands for workload-aware VM placement and migrations. The pipeline considers CPU, memory, and I/O utilization and is designed to account for VM and resource heterogeneities in cloud datacenters. Our innovation is their incorporation into a workload-aware allocation framework, which eliminates needless migrations and enhances the energy–performance trade-off in heterogeneous cloud datacenters, in contrast to previous work that handles prediction or classification independently [19].

In stage one, the input contains resource utilization values of each workload/VM, specifically CPU, memory, and I/O (disk) utilization obtained from the Google Cluster traces [2]. These values are used as the input features for the prediction models. In stage two, the learning models predict the resource demand values, i.e., future CPU, memory, and I/O utilization. Four regression models are evaluated for this purpose: (i) Linear Regression (LR); (ii) Support Vector Regression (SVR); (iii) Artificial Neural Network (ANN); and (iv) XGBoost. These models are resource demand predictors and are not workload classifiers in the proposed pipeline. Their output is a three-dimensional predicted resource demand vector denoted by $\hat{\mathbf{r}} = (\widehat{CPU}, \widehat{Mem}, \widehat{IO})$. In stage three, the predicted resource demand vector is passed to a deterministic rule-based classifier [20]. The classifier assigns each workload to one of four workload classes: (i) CPU-intensive; (ii) memory-intensive; (iii) I/O-intensive; or (iv) mixed. The learning models and the rule-based classifier have different roles: LR, SVR, ANN, and XGBoost predict continuous resource demand values, whereas the rule-based classifier converts these values into a categorical workload type. In stage four, the predicted workload and workload class are used for VM placement and migration. The workload classification rule is defined as:

$$\text{classify}_{vm} = \begin{cases} \text{CPU}, & \text{if } cpu > 70,\ mem < 50,\ disk < 20 \\ \text{Memory}, & \text{if } mem > 70,\ cpu < 50,\ disk < 20 \\ \text{I/O}, & \text{if } disk > 50,\ cpu < 50,\ mem < 70 \\ \text{Mixed}, & \text{otherwise.} \end{cases}$$

Prior studies on workload-aware scheduling and VM placement (e.g., [21], [2]) have used comparable cutoff values (usually 60–80%) to identify jobs that require a lot of resources. As a fair compromise between conservatism and aggression, we choose 70%. The lower thresholds for the non-dominant resources prevent a workload from being classified as CPU-, memory-, or I/O-intensive when substantial demand is simultaneously present in another resource dimension. Consequently, workloads that do not satisfy any of the dominant-resource conditions are categorized as mixed. The resulting workload class is subsequently provided to the WorkloadAware approach, where it is used together with host availability and the ranking mechanism to make placement or migration decisions. Note that in this prediction stage, XGBoost is used as a regression model; it is not used as a Random Forest classifier. Similarly, LR, SVR, and ANN are evaluated as alternative resource demand prediction models rather than as independent workload-type classifiers. ANN achieved the best prediction performance among the evaluated models, as reported in Sec. 6, and is therefore used as the preferred prediction model in the subsequent workload-aware resource management experiments [22], [23].

The models are trained using resource utilization data extracted from the Google Cluster traces. The dataset is divided into separate training and testing subsets, with the training portion (80%) used to learn the model parameters and the remaining testing portion (20%) used for evaluation. The same input features and prediction targets are used for all four regression models to ensure a fair comparison. Specifically, the input features comprise CPU, memory, and I/O utilization observations, while the prediction targets are the corresponding CPU, memory, and I/O resource demand values. The trained models are then evaluated using the same testing data, and the resulting predictions are passed through the identical rule-based classifier. In our experiments, LR was used with its default linear-regression configuration, while SVR used an RBF kernel with $C = 1.0$, $\epsilon = 0.1$, and $\gamma = \text{scale}$. The ANN employed two hidden and three dense layers (64, 32, and 16 neurons), ReLU activation, the Adam optimizer, a learning rate of 0.001, a batch size of 128, and up to 50 epochs with early stopping. XGBoost was configured with 200 estimators, a maximum depth of 6, a learning rate of 0.05, and subsampling and column-subsampling rates of 0.8. The dataset was divided into 80% training and 20% testing subsets chronologically. The workload classification thresholds were fixed according to the above rule and were applied identically to the predictions provided by each regression model. In the Google data[1], the usage of resources is recorded at 5-minute intervals. In our experiments, we replay the Google traces where the resource usage information is known. For unknown/future workloads, the trained prediction model estimates the CPU, memory, and I/O demands, after which they are classified using the same deterministic workload classification rules. This unified procedure is used to support workload-aware VM placement and migration [22], [23]. We report the outcomes in Sec. 6.

3.4 Time Complexity

For Alg. 1, predicting workload and classifying VM types requires iterating over all V VMs, resulting in a complexity of $O(V)$. Next, estimating scores for all hosts requires iterating over H hosts, and

1. https://github.com/google/cluster-data

sorting them based on scores and efficiency takes $O(H \log H)$. The allocation step involves iterating over all V VMs and, for each VM, iterating over all H hosts, leading to a complexity of $O(VH)$. If a VM cannot be allocated, it is added to a queue, and a backfilling process attempts to reallocate unassigned VMs, contributing another $O(VH)$. Summing these complexities, the overall worst-case complexity is given by $O(VH + H \log H)$. Since $O(VH)$ dominates in large-scale scenarios, the final computational complexity can be approximated as $O(VH)$.

In Alg. 2, identifying overloaded and underloaded hosts requires iterating over all H hosts, leading to a complexity of $O(H)$. Sorting the hosts based on efficiency incurs an additional complexity of $O(H \log H)$. For each host h_i in the overloaded or underloaded sets, retrieving the set of VMs requires iterating over all V VMs, contributing $O(V)$. Sorting these VMs by resource usage results in an additional $O(V \log V)$. Next, for each VM, the algorithm attempts to find a new host by iterating over all potential hosts, leading to a worst-case complexity of $O(H)$. In the worst case, all VMs are checked against all hosts, leading to a complexity of $O(VH^2)$. Summing these steps, the overall complexity is $O(H)+O(H \log H)+O(V)+O(V \log V)+O(VH^2)$. Since $O(VH^2)$ dominates in large-scale scenarios, the final worst-case complexity can be approximated as $O(VH^2)$.

The resulting time complexity of $\mathcal{O}(VH^2)$ is a worst-case bound, which should be noted. The search space is greatly reduced in practice by the suggested algorithms: (i) only candidate hosts above the threshold T are taken into consideration during the allocation phase; (ii) migration only affects overloaded and underloaded hosts instead of all hosts; and (iii) unallocated VMs are bounded in the backfill queue. Our simulations with 12,583 heterogeneous hosts demonstrate that the effective runtime is significantly closer to linear in VH. Furthermore, we observed an approximately linear growth for evaluated system sizes as discussed in Sec. 5.7. Furthermore, our method's computational overhead pales in comparison to workload execution durations. For instance, scheduling choices only took a couple of seconds every interval (5 minutes), even under heavy Google workloads, but job execution takes minutes to hours. This guarantees that the goal of enhancing workload performance is maintained: our approach delivers up to 8.17% performance improvement while retaining scalable decision-making overhead by avoiding resource contention through workload-aware placement.

4 Performance Evaluation

In our simulated datacenter, 12,583 heterogeneous computers are evenly distributed, having five distinct CPU architectures. Table 2 illustrates how the CPU design reflects heterogeneity in terms of performance and energy usage. Additionally, we built three distinct scheduling policies: FCFS, FF (first fit), and FillUp. They were then expanded with the workload-aware technique to create three additional scheduling policies: FCFS+WA, FF+WA, and FillUp+WA. Assumed to be in place in the datacenter, workload-aware migrations are activated, and the consolidation strategy moves VMs when specific servers surpass or fall short of predetermined threshold values. To migrate VMs, we use three distinct assumptions: (i) migrate all; (ii) WAMS; and (iii) WAMR. Additionally, as indicated in Table 3, we assume five distinct kinds of VMs that replicate instances of Amazon Web Services (AWS). Additionally, we make the assumption that there are three distinct kinds of workloads or applications, or W1, W2, and W3. Keep in mind that every workload consists of over a million processes, each of which has distinct requirements for the resources of the hosts, including memory, CPU cores, and runtime, among other things. Furthermore, we assume that all tasks utilize the stochastic usage model included in the traditional CloudSim simulation tools [24].

All of the experiments are conducted using PerficientCloudSim [13], an expanded version of CloudSim [24]. PerficientCloudSim facilitates the modeling of resource contention and performance degradation resulting from co-located VMs on a specific system. These tasks and their attributes, including run times, arrival times, schedule times, and priorities, are associated with the Google cluster dataset [25] and [26]. By considering every job as a separate application with a total execution time equal to the workload runtime, we handle a workload holistically. Furthermore, we assume that each task's priority in Google data indicates workload type because the priority affects billing [2]. We then use real workload benchmarks from [16] and map both datasets using statistical methods over their distributions [2]. This leads us to heterogeneity of workloads (Google data) and CPU architectures in Table 2. The performance of different scheduling and migration policies is then compared using the execution time. Each application's cost is then determined by its execution time and the kind of VMs used to perform the workload.

TABLE 2: Various characteristics of simulated servers

CPU model	Speed (MHz)	No of cores	No of ECUs	Memory (GB)	P_{idle} (Wh)	P_{max} (Wh)
E5430	2,830	8	22.4	16	166	265
E5-2620	2,000	12	24	32	70	300
E5645	2,400	12	28.8	16	63.1	200
E5-2650	2,000	16	32	24	52.9	215
E5-2670	2,600	16	41.6	24	54.1	243

As indicated in Table 2, the datacenter consists of 12,583 heterogeneous hosts that correspond to five distinct CPU platforms. The hosts' idle and maximum power use (P_{idle} and P_{max}, respectively) were derived from the SPECpower benchmarks[2]. The speeds of the hosts are converted to MIPS to ensure compatibility with the PerficientCloudSim simulator. Each core of a certain host corresponds to one vCPU of a single VM instance (Table 3), and each host can support many VMs. We also assume that each task runs in a VM instance, with the speed of the VM being precisely equivalent to each task's CPU demand, as indicated in Table 3. We also take into consideration resource contention caused by co-location, or when identical VMs are deployed on the same host and compete for the same resources [27], migration costs [2], resource heterogeneity (in terms of CPU architecture), and so on. The parameters for resource contention and CPU heterogeneity for different hosts, as displayed in Table 2, were derived from our earlier work [2]. The host over/underload thresholds were set to different values $T_{high} = \{0.8, 0.7, 0.6\}$ and $T_{low} = \{0.05, 0.1, 0.2\}$. According to [3], each migration reduces the VM performance by 10%. Additionally, each VM migration's energy expense, E_m, is simulated using the following Eq. 11:

$$E_m = 0.512 \times VM_{data} + 20.165 \tag{11}$$

where the variable VM_{data} represents both the memory pages used during the migration process and the total memory allotted to the VM (measured in MBs) [2].

4.1 Evaluation Metrics

To assess different resource allocation, consolidation, and management strategies, we take into account the overall number of migrations, energy consumption (KWh), and task execution time (in minutes) as performance measures. We employ a composite measure, represented by N_T, which is calculated using the weighted sum approach, to present the trade-off between multiple metrics, including energy consumption, performance, and the number of migrations. This measure integrates these data in a way that makes

2. https://www.spec.org/power_ssj2008/

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2026.3734089

TABLE 3: Amazon's different instance types and their characteristics

Instance type	No of vCPUs	No of ECUs	Speed (MHz) MIPS	MEMORY (GB)	Storage (GB)	Reserved price (1 year) ($/hour) US East - N. Virginia
t2.nano	1	1	1,000	0.5	1	0.006
t1.micro	1	1	1,000	0.613	1	0.02
m1.small	1	1	1,000	1.7	160	0.044
m1.medium	1	2	2,000	3.75	410	0.087
m3.medium	1	3	3,000	3.75	4	0.067

the trade-off simple to evaluate. Better performance, according to the combined criterion, is indicated by a lower value for N_T. The formula is given in Eq. 12:

$$N_T = E_w \times E_n + R_w \times R_n + M_w \times M_n \tag{12}$$

where the weights are indicated by E_w, R_w, and M_w, and the normalized values for energy consumption, performance (measured in runtimes), and the number of migrations are described by E_n, R_n, and M_n, respectively. The IaaS can change the weights to prioritize one goal above others and vice versa. Furthermore, IaaS providers usually runs under strict SLA requirements, where performance violations are not acceptable. Therefore, a constraint-satisfaction formulation would be more suitable to enforce a maximum allowable performance loss or runtime. For example, minimize E subject to $R \leq R_{\max}$, where $R_{\max}$ denotes the maximum runtime allowed in SLA. This implementation requires redesigning the allocation and migration methods to incorporate feasibility checks and rejection or penalty mechanism, which is beyond the scope of our current work. To check the model performance in predicting accurate workload types, the accuracy is calculated using Eq. 13:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

where true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) denote all four outcomes from the confusion matrix. A perfect prediction model would have zero FP and zero FN and, therefore, an accuracy of 1.0, or 100%. To compare various prediction approaches, we use other metrics such as precision (accuracy of positive predictions), recall (ability to find all positive instances), and F1-score (harmonic mean of precision and recall).

5 Experimental Results and Discussion

The experimental results for various allocation and migration policies using three different workload types are shown in Table 4. We employ the weighted sum measure N_T (Eq. 12) to jointly assess energy, performance, and migration overhead. WA-based approaches regularly achieve reduced N_T, demonstrating their balanced trade-off, as shown in Fig. 4. Using the current schedulers (FCFS, FF, FillUp), we integrate workload-aware policy, leading to three allocation strategies: FCFS+WA, FF+WA, and FillUp+WA. The NO, All, WAMS, and WAMR are the four techniques used to evaluate VM migrations. The findings show that WAMS is more successful at maintaining performance for longer workloads, while WAMR performs better for shorter workloads by minimizing needless migrations counts. In subsequent subsections, we discuss the obtained outcomes.

5.1 Energy Consumption

Based on baseline allocation policies, Fig. 2 displays the energy consumption. Because of its better workload-aware placement, we find that FillUp+WA uses up to 5.25% less energy than FillUp. The results show that for certain workloads, migrations may be expensive both in terms of energy consumption and performance. However, they can be effective if workloads are allocated and migrated appropriately. Furthermore, we also observed that relaxing the migration criterion for certain applications improves energy and performance efficiencies. For the W3 workload, the NO migration approach is 3.83%–6.99% energy efficient and 4.91%–8.22% performance efficient than the ALL migration approach for FCFS, FF, and FillUp approaches. By integrating workload awareness FCFS+WA, FF+WA, and FillUp+WA, we observed that the ALL migration approach is 0.87%–1.83% energy efficient and 1.01%–2.15% performance efficient than the NO migration approach. The outcomes are shown in Fig. 2. Our investigation suggests that workload awareness of the resource allocation and consolidation policies could significantly reduce the total number of migrations for short-running workloads. Fig. 3 shows the total number of migrations using various approaches.

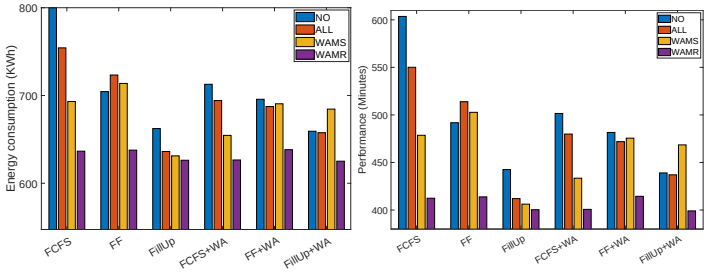

Fig. 2: Overall improvements in energy consumption (left) and performance (right) when using FCFS+WA, FF+WA, and FillUp+WA instead of FCFS, FF, and FillUp, along with different approaches to migrations

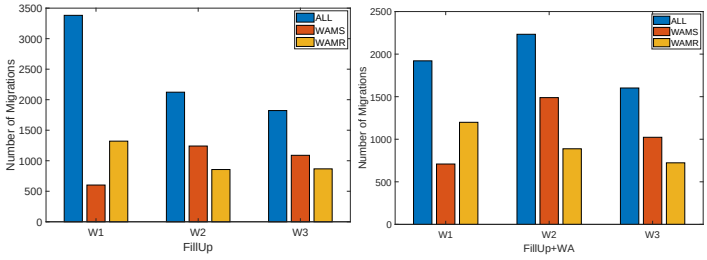

Fig. 3: Total number of migrations when using FillUp and FillUp+WA instead of FCFS and FF, along with different approaches to migrations

5.2 Performance and Execution Time

Task execution time is used to gauge performance. The WA-integrated policies (FCFS+WA, FF+WA, FillUp+WA) produce an average improvement of 8.17% over their baselines, as indicated in Table 4. This demonstrates how avoiding workload interruption improves task completion in general. For the W1 workload, workload awareness can reduce energy consumption up to 10.9%, 1.24%, and 0.46% and improve performance up to 16.91%, 2.08%, and 0.8% for FCFS, FF, and FillUp allocation policies, respectively. For the W2 workload, these numbers were observed as 0.12%, 0.61%, and 0.15% for energy and 0.15%, 0.74%, and 0.19% for performance. For the W3 workload, FCFS+WA and FillUp+WA

were noted to be 2.64%, 0.14% energy efficient and 3.05%, 0.17% performance efficient compared to FCFS and FillUp approaches. However, FF+WA produces worse results, i.e., a 4.06% increase in energy consumption and a 4.76% performance loss. This might be due to the heterogeneity of resources and workloads, in particular when workloads are competing for similar resources, as modeled in the PerficientCloudSim simulator [13]. Furthermore, we assume the priority of each task as a representation of the workload type, which may not be a realistic assumption to classify workloads. Overall, the effectiveness of the workload-aware policy is related to the allocation policy's efficiency — the more efficient is the allocation strategy, the more effective the workload awareness, and vice versa. This means that effective allocation policies can beat migration approaches if resources and workloads are kept in the placement decisions.

5.3 Energy/performance Tradeoff

We observed a tradeoff between various allocation and migration policies in achieving energy and performance improvement for different types of workloads. The tradeoff exists for various workloads and is related to the allocation and migration strategies. Fig. 4 describes how the tradeoff is balanced among energy and performance. For example, when comparing WAMS to WAMR, we noted that WAMR could offer 0.8%–10.65% and 1.44%–17.69% improvements in terms of energy and performance, respectively, for W1 and W2 (short-running workloads). This might be due to the fact that WAMR increases workload migrations to other hosts, and there are more chances that the migrated workload might run on target hosts in a more efficient way (in terms of energy and performance) than the source hosts. However, for long-running workload W3, WAMS can be 4.34%–15.05% more energy efficient and 5.13%–17.28% more performance efficient than the WAMR (except FCFS+WA). We believe that more accurate prediction approaches and assumptions about workload types are essential for better resource management.

5.4 Prediction Accuracy

Table 6 demonstrates that ANN generally achieves the highest final workload classification accuracy (up to 98.7%) and maintains a good tradeoff between energy consumption and performance across the three workloads (W1, W2, and W3). This means that the ANN not only delivers better results in terms of predictive accuracy but also does so with an optimal energy-performance balance. Furthermore, WAMR provides a slightly better performance-to-energy tradeoff (N_T), showing its effectiveness in handling heavier workloads while maintaining relatively low energy consumption. Fig. 5 shows the relative performance of several workload prediction models for various allocation and migration policies when different workloads are assumed to be running in the datacenter, with the associated error bars showing the degree of uncertainty. The "NO" approach means that classification was done before the experiments from the Google cluster data. We observed that the "NO" and "ANN" models appear to perform better than "LR" and "SVR", despite minor variations between them, particularly when it comes to accuracy for various workloads. In certain instances, "NO" and "ANN" exhibit less variance than "LR" and "SVR", according to the error bars, which also imply that the variability in outcomes is quite constant.

Table 5 reports a comparison of various workload prediction methods using multiple metrics. The ground-truth labels were obtained from resource-utilization values in Google traces, where resource utilization values were processed with a rule-based classifier (Sec. 3.3). These rule-derived labels constitute the ground truth used to evaluate the final workload classification. The outcomes demonstrate that ANN outperforms LR and SVR under both the FillUp+WA and FF+WA strategies. While SVR performs marginally better with accuracy reaching 0.93 with balanced precision, recall, and F1-score of roughly 0.90–0.91, LR consistently reaches an accuracy of 0.91 with moderate precision, recall, and F1-score values (0.88–0.89). With the highest accuracy of 0.987 and almost flawless results in every other parameter (precision 0.985, recall 0.984, and F1-score 0.984), ANN, however, performs noticeably better than both. As the most dependable model for precise and effective workload classification, the similar performance trends under FF+WA and FillUp+WA show that ANN's dominance is irrespective of the scheduling policy. Out of LR, SVR, and ANN, the accuracy of the ANN prediction was as high as 98.7%. Although, ANN has a slightly higher inference latency (≈ 0.2 ms) than LR and SVR ($\approx 0.06 - 0.09$ ms), but it remains well within the 5-minute scheduling interval and therefore does not impose any operational overhead. Moreover, LR and SVR are more suitable for latency-critical scheduling decisions. This demonstrates that using ML-based predictors for accurate workload predictions and classification is valid.

5.5 Number of Migrations

Table 4 and Fig. 3 show that the FCFS, FF, FillUp algorithms are 3.83%–6.99% less energy and 4.91%–8.22% less performance efficient than the NO migration approach (W3). This shows VM migrations are dominant source of overhead in datacenters. Mostly these migrations are useless and if a single VM is migrated several times, the performance impact would be more severe leading to high energy usage. Alg. 2 ensures single VM migration per cycle to avoid performance loss. With reduced number of migrations, the datacenter utilisation remains high as shown in Table 10. When migration thresholds are set more aggressively, it increases migration counts from 1,256 to 4,502. With the single-migration, the utilization variance remains relatively low (0.018–0.031) showing datacenter balanced state. This verifies system's ability to recover from a severely overloaded state.

5.6 Results Generalization

We conduct a number of tests using various datacenter configurations, hosts, presumptions, and workload types in order to illustrate the workloadAware approach's suitability in an actual cloud testbed. A variety of hosts were chosen based on their attributes and workload types from [12]. To determine whether or not the generalization of our findings is accurate, we verified the workloadAware approach. In our previous outcomes, workloads were categorized based on their runtimes, priority, and resource usage, which were already known. Here, we used various machine learning methods to predict the resource usage of various VMs/tasks and then classify them according to Sec. 3.3. The machine learning approaches LR, SVR, and ANN were implemented using the *Keras* library with default parameters[3], i.e., SVR with RBF kernel, $C = 1.0$, $\epsilon = 0.1$), $\gamma = \text{scale}$, and the ANN consists of two hidden and three dense layers (64-32-16 neurons) with ReLU activation, Adam optimizer ($\eta = 10^{-3}$), batch size 128, and 50 training epochs with early stopping. The experimental results are shown in Table 6 and Fig. 5. To ensure reproducibility, we ran each experiment 10 times using fixed, independent random seeds; and Table 6 reports average outcomes while the bars in Fig. 5 denotes standard deviation. We found some overlap between FillUp+WA and FF+WA, but overall, our method FillUp+WA performs better than other approaches. We found that both approaches are substantially different using the *t-test* analysis (95% confidence interval, i.e., p = 0.05), with FillUp+WA performing considerably better than FF+WA. More migrations

3. https://keras.io/

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2026.3734089

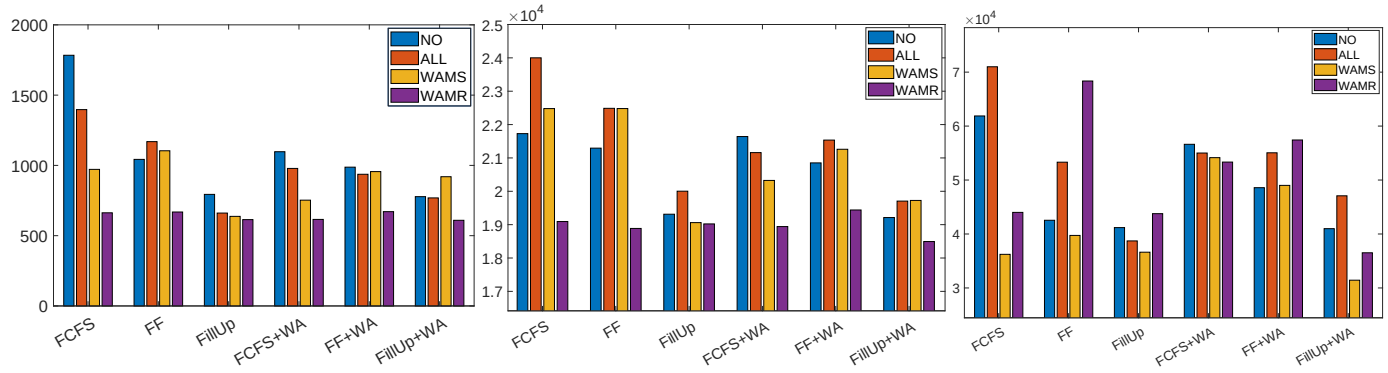

Fig. 4: The values for N_T using various scheduling and migration techniques for three different workload types [W1, W2, W3 – from left to right]

TABLE 4: Results of experiments on three distinct workload types in terms of energy consumption (E), performance (P), and total number of migrations (M) Performance is the inverse of execution time, reported in minutes, and lower numbers are preferable to high ones. Energy is measured in KWh. The best strategies for balancing the trade-off between different objectives are those with the lowest values for the N_T measure.

Scheduling technique	Migration approach	W1				W2				W3			
		M	E	P	N_T	M	E	P	N_T	M	E	P	N_T
FCFS	NO	0	799.99	603.58	7.02	0	1569.7	1503.94	15.37	0	2141.56	2172.82	21.57
	ALL	5231	754.27	550.11	6.52	2391	1615.83	1557.9	15.87	1975	2232.68	2279.4	22.56
	WAMS	1111	693.14	478.61	5.86	1643	1585.28	1522.16	15.54	1867	1824.09	1801.49	18.13
	WAMR	1921	636.56	412.43	5.24	1315	1512	1436.45	14.74	1231	1932.89	1928.74	19.31
FCFS+WA	NO	0	712.76	501.55	6.07	0	1567.9	1501.77	15.35	0	2084.96	2106.56	20.96
	ALL	2898	694.25	479.9	5.86	2133	1557.69	1489.83	15.24	1763	2066.83	2085.35	20.76
	WAMS	977	654.56	433.48	5.44	1642	1539.64	1468.72	15.04	1349	2057.08	2073.94	20.66
	WAMR	1222	626.54	400.71	5.14	1167	1508.57	1432.38	14.7	1132	2047.64	2062.9	20.55
FF	NO	0	704.44	491.82	5.98	0	1560.51	1493.19	15.27	0	1913.48	1906.04	19.1
	ALL	4123	723.32	513.91	6.19	2234	1585.44	1522.35	15.54	1856	2047.37	2062.65	20.55
	WAMS	998	713.77	502.74	6.08	1678	1585.3	1522.19	15.54	1444	1874.89	1860.91	18.68
	WAMR	1882	637.76	413.83	5.26	954	1507.25	1430.89	14.69	841	2207.15	2249.54	22.28
FF+WA	NO	0	695.69	481.59	5.89	0	1551.07	1482.09	15.17	0	1991.08	1996.75	19.94
	ALL	2199	687.41	471.91	5.8	2547	1565.67	1499.16	15.32	1709	2067.31	2085.91	20.77
	WAMS	711	690.55	475.58	5.83	1348	1559.83	1492.33	15.26	1432	1996.31	2002.87	20
	WAMR	906	638.28	414.44	5.26	845	1519.91	1445.64	14.83	1011	2093.8	2116.89	21.05
FillUp	NO	0	662.31	442.55	5.52	0	1517	1442.3	14.8	0	1895.02	1884.45	18.9
	ALL	3382	636.2	412.01	5.24	2123	1532.48	1460.41	14.96	1822	1860.46	1844.03	18.52
	WAMS	602	631.23	406.19	5.19	1241	1511.26	1435.58	14.73	1089	1830	1808.4	18.19
	WAMR	1321	626.23	400.34	5.13	856	1510.36	1434.53	14.72	867	1929.76	1925.09	19.27
FillUp+WA	NO	0	659.29	439.01	5.49	0	1514.76	1439.62	14.77	0	1892.35	1881.27	18.87
	ALL	1921	657.6	437.04	5.47	2233	1525.9	1452.65	14.89	1602	1972.38	1974.87	19.74
	WAMS	709	684.5	468.5	5.77	1489	1526.35	1453.17	14.9	1023	1749.11	1713.73	17.31
	WAMR	1199	625.16	399.09	5.12	888	1498.17	1420.21	14.59	723	1828.41	1806.48	18.17

TABLE 5: Comparison of ML models using multiple metrics – accuracy reports the final workload classification

Policy	Model	Accuracy	Precision	Recall	F1-score	Latency (ms)
FF+WA	LR	0.91	0.89	0.88	0.88	0.061
	SVR	0.93	0.91	0.90	0.91	0.089
	ANN	0.987	0.985	0.984	0.984	0.202
FillUp+WA	LR	0.91	0.89	0.88	0.88	0.058
	SVR	0.93	0.91	0.90	0.91	0.085
	ANN	0.987	0.985	0.984	0.984	0.213

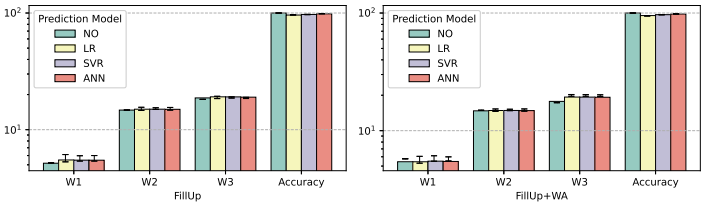

Fig. 5: Overall improvements in N_T and accuracy when using FCFS+WA instead of FillUp, along with different approaches to workload prediction – the bars denote standard deviations across ten multiple runs

take place as the number of apps increases, which raises IaaS energy consumption to 6.85% at a performance loss of about 0.01%. However, FCFS+WA's runtimes differ for 300 applications, which results in a poorer average performance. However, utilizing the *t-test* analysis, we found significant differences between FF+WA and FCFS+WA—the lowest values for p (*t-critical* = 2.374) [2].

The FillUp approach produces marginally better performance across workloads, especially when combined with prediction models like ANN and SVR. However, this comes at the cost of high energy consumption, slightly leading to higher N_T. FillUp+WA, on the other hand, exhibits greater energy efficiency and lower N_T. Even though FillUp+WA performs somewhat worse, it is still competitive, particularly when using the ANN model, which continuously produces the best prediction accuracy (up to 98.1% for FillUp and 97.9% for FillUp+WA). All things considered, FillUp is better when performance is the top concern, whereas FillUp+WA is better to balance the tradeoff between energy consumption and performance.

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2026.3734089

TABLE 6: Experimental findings for three distinct workload types and machine learning techniques in terms of energy consumption (E), performance (P), accuracy, and N_T — methods with the lowest values for the N_T measure are the most effective at balancing the trade-off between different goals

Scheduling technique	Migration approach	Prediction approach	W1			W2			W3			Accuracy
			E	P	N_T	E	P	N_T	E	P	N_T	(%)
FillUp	WAMS	No	631.23	406.19	5.19	1511.26	1435.58	14.73	1830	1808.4	18.19	-
		LR	675.65	466.98	5.71	1542.41	1476.42	15.09	1901.71	1878.22	18.9	96.4
		SVR	671.32	465.1	5.68	1551.33	1482.8	15.17	1900.2	1876.34	18.88	97.1
		ANN	673.32	464.21	5.69	1541.44	1480.91	15.11	1878.58	1862.87	18.71	98.7
	WAMR	No	626.23	400.34	5.13	1510.36	1434.53	14.72	1929.76	1925.09	19.27	-
		LR	648.53	408.89	5.29	1518.6	1454.88	14.87	1936.9	1931.03	19.34	95.8
		SVR	647.45	408.1	5.28	1509.34	1451.45	14.8	1936.1	1930.44	19.33	97.3
		ANN	644.23	408.01	5.26	1508.1	1450.4	14.79	1931.5	1928.87	19.3	98.1
FillUp+WA	WAMR	No	684.5	468.5	5.77	1526.35	1453.17	14.9	1749.11	1713.73	17.31	-
		LR	688.33	446.66	5.67	1534.76	1453.87	14.94	2001.12	1990.1	19.96	94.6
		SVR	699.88	461.9	5.81	1543.3	1456.7	15	1999.32	1989.6	19.94	96.8
		ANN	691.98	459.23	5.76	1541.66	1452.33	14.97	1989.6	1981.7	19.86	98.2
	WAMS	No	625.16	399.09	5.12	1498.17	1420.21	14.59	1828.41	1806.48	18.17	-
		LR	639.56	401.89	5.21	1511.45	1436.67	14.74	1877.76	1852.4	18.65	95.7
		SVR	638.88	401.22	5.2	1509.6	1431.7	14.71	1876.01	1851.2	18.64	96.8
		ANN	637.51	400.81	**5.19**	1501.56	1428.36	**14.65**	1870.88	1849.99	**18.6**	97.9

5.7 Scalability Analysis

The time complexity of $\mathcal{O}(VH^2)$ is a theoretical worst-case bound (Sec. 3.4). In reality, the search space is significantly limited by three mechanisms: (i) only hosts with workloads above thresholds are searched for allocation; (ii) only overloaded and underloaded hosts are searched for migration; and (iii) only VMs that were not allocated during initial placement are added to the backfilling queue. To test this behavior empirically, we have tested WorkloadAware approach for scheduling and migration decision times using different system sizes and workload type W1. The experiments were performed 10 times on a Lenovo workstation running Ubuntu 22.04 and Java 8.0 with 32GB RAM and 8-core 3.0 GHz CPU. As shown in Table 7, on average the scheduling decision time rose from ∼0.31s for 1,000 hosts and 5,000 VMs to 2.26s for 12,583 hosts and 50,000 VMs. Migration decision time was found to be 0.1s to 0.97s over the same range, in comparison. The growth we observed is significantly smaller than the worst case $O(VH^2)$, and is close to linear with the system sizes, which demonstrates that host allocation and migration methods are successful in limiting the practical search space. For large configurations, the overhead for making decisions is still significantly less than that required for the Google Cluster traces to monitor the workload every five minutes. This validates that our approach can make scheduling and migration decisions at practical timescales and achieve a performance boost of up to 8.17% by workload-aware placement and less resource contention.

TABLE 7: Scheduling and migration decisions times for the WorkloadAware approach with different system sizes – workload type W1

Scheduling approach	Migration approach	Hosts	VMs	Scheduling time (s)	Migration time (s)
FillUp+WA	WAMS	1,000	5,000	0.31±0.008	0.1±0.003
		2,500	10,000	0.48±0.009	0.21±0.01
		5,000	20,000	0.96±0.02	0.39±0.01
		10,000	40,000	1.79±0.021	0.75±0.02
		12,583	50,000	2.26±0.026	0.97±0.02

5.8 Parameters' Impacts on Results

5.8.1 Number of Co-located VMs

The number of VMs (running the same workloads) on a host is limited by parameter k used in the WAMR approach. To lessen resource competition, we utilized $k = 2$ in order to prevent significant performance loss seen in co-location and promote workload diversity. However, k is a configurable parameter rather than a universal constant. Depending on the host capacity and the anticipated per-VM demand, we can actually set k per host (or per host-class), for example:

$$k_h = \left\lfloor \frac{\beta \cdot \text{host\_TResource}_h}{\mathbb{E}[\text{VM\_demand}_{\text{type}}]} \right\rfloor \tag{14}$$

where the packing factor $\beta \in (0, 1]$ takes contention into consideration. This formulation guarantees that, rather than using a small, arbitrary constant, very large counts of same-type VMs (such as 100k CPU VMs) are packed in accordance with actual host capability. Instead of static k, we can also consider a lightweight runtime approach that adjusts k dynamically using real-time resource contention. Periodically, each host h reports its CPU, memory, and I/O utilizations, as well as performance degradation. These metrics can be combined into a contention score $CS(h) \in [0, 1]$, which reflects contention of host resources. The score can be mapped to a co-location limit $k(h)$, where low contention means a larger number of similar workloads, while high contention limits co-location. A dynamic approach would require continuous monitoring of contention metrics that have significant overhead and make results platform-dependent. To illustrate the trade-off between energy, performance, and migration overhead, we present a sensitivity analysis (Table 8) that sweeps k (and β). For FillUp+WA+WAMR, raising k considerably boosts efficiency while lowering migrations. Poor consolidation is evident when $k = 1$, as both workloads show the maximum energy (760.2 KWh), performance (490.5 min), and migrations (3421). Energy and performance gradually decline when k rises to 2 and 4, while migrations drop precipitously from 3421 to 1587, indicating more consistent VM placement. After $k = 8$, the benefits level off, with migrations dropping to roughly 1495 and energy consumption and performance leveling at 732.5 KWh and 471.8 min, respectively. At larger values of k, where the system achieves an ideal balance between migrations and consolidation efficiency, this implies diminishing returns. The optimal trade-off between energy, performance, and migration overhead is obtained with moderate values of k (about 4–8).

5.8.2 Energy/performance/migrations Weightage

Energy E, runtime/performance R, and migrations M are all component metrics that have been first min–max normalized to $[0, 1]$ (Eq. 9-style normalization). Non-negative weights E_w, R_w, M_w that add up to 1 are used in the composite score. To prevent

TABLE 8: Sensitivity analysis for number of co-located VMs (FillUp+WA+WAMR)

k	Energy (KWh)	Time (min)	Number of Migrations	Energy (KWh)	Time (min)	Number of Migrations
	W1			W2		
1	760.2	490.5	3421	760.2	490.5	3421
2	744.1	478.9	2199	744.1	478.9	2199
4	735.6	472.3	1587	735.6	472.3	1587
8	732.8	471.9	1510	732.8	471.9	1510
16	732.5	471.8	1495	732.5	471.8	1495

bias toward any one goal, we provide findings for equal weighting in the baseline experiments $(E_w, R_w, M_w) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. We performed sensitivity analysis that sweeps the weights across a grid (Table 9) and noted that with balanced weights, WAMR outperforms WAMS and FillUp by achieving the best overall tradeoff (0.365). WAMS continuously produces the lowest values when energy is given priority (0.280 and 0.260 for energy-first and extreme-energy), demonstrating its effectiveness in reducing energy use. However, WAMR's potential to reduce execution time is demonstrated by its highest performance in performance-focused settings, with 0.479 in performance-first and 0.561 in extreme-performance. Only in migration-conscious environments does FillUp exhibit benefits; when migration overhead predominates, it achieves the lowest value (0.370).

TABLE 9: Performance of algorithms under different N_T weight [lower values indicate better tradeoff]

Criteria	(E_w, R_w, M_w)	FillUp	WAMS	WAMR
Energy	(1.0, 0.0, 0.0)	0.270	**0.260**	0.265
Performance	(0.0, 1.0, 0.0)	0.621	0.588	**0.561**
Migrations	(0.0, 0.0, 1.0)	0.410	**0.405**	0.412
Energy-first	(0.70, 0.20, 0.10)	0.302	**0.280**	0.295
Performance-first	(0.10, 0.80, 0.10)	0.551	0.502	**0.479**
Migration-cons.	(0.20, 0.20, 0.60)	**0.370**	0.385	0.392
Balanced	(0.33, 0.33, 0.33)	0.421	0.398	**0.365**

According to the complexity analysis in Sec. 3, $O(VH^2)$ is the migration algorithm's primary cost. This complexity might, in fact, become unaffordable for a strictly centralized system at numbers like 100,000 hosts. The suggested framework can be expanded in two ways to address scalability: (i) *hierarchical clustering*, in which hosts are grouped into clusters (such as racks, pods, or zones) and decisions about scheduling and migration are made locally before being coordinated globally; and (ii) *decentralized execution*, in which the algorithm is run independently by each cluster or region with little coordination between clusters. These tactics make the method computationally viable at scales larger than 100,000 hosts by drastically lowering the effective problem size that each decision-maker sees. In order to prove scalability in various hyperscale datacenter scenarios, we intend to include such modifications in our subsequent work.

The wall-clock time of proposed algorithms is affected by many factors that make it lower than the theoretical worst-case complexity. In practice, the average number of VM-to-host mappings is smaller than H because First-Fit, FillUp, and the FillUp+WA terminate early when a suitable host is allocated. The host-feasibility checks (constant-time) and simple ranking score computation make per-host evaluation cost lightweight. ANN inference latency is less than millisecond and invoked once per VM per cycle that make workload-prediction overhead negligible. During scheduling, cached host states and incremental updates decrease recomputation. The system benefits from CPU-level parallelism, where VM placements are processed concurrently across cores. Finally, simulator-level overheads, e.g., logging and object creation, contribute marginally and do not affect algorithmic scalability. Overall, these factors ensure that the wall-clock time for the proposed system remains practical and below the worst-case bound reported in Sec. 3.4.

5.8.3 Thresholds for Placement & Migrations

The allocation $S_i > T$, overload T_{low}, and underload T_{high} thresholds – used in our allocation and migration policies – have impact over the obtained outcomes. The sensitivity analysis shows a straightforward trade-off i.e., raising T_{low} decreases migrations at the expense of greater utilization variance, while increasing T decreases the instantaneous deployment of VMs onto hosts with moderate usage but increases downstream migrations – as shown in Table 10. The system only executes 1256 migrations, activates 1129 hosts, and achieves the lowest utilization variance (0.023) with the lowest threshold setting ($T = 0.3$, $T_{low} = 0.05$, $T_{high} = 0.6$). This indicates stable but less energy-efficient consolidation because more hosts are still active. Raising the thresholds to moderate levels ($T = 0.6$, $T_{low} = 0.10$, $T_{high} = 0.7$) results in more migrations (2702 migrations), fewer active hosts (978), and a further decrease in variance (0.018), which is indicative of better workload consolidation and balanced host utilization. Nevertheless, the number of migrations increases significantly to 4502 and the number of active hosts to 1219 with the most aggressive threshold setting ($T = 0.8$, $T_{low} = 0.20$, $T_{high} = 0.8$), while the variance also increases (0.031). This points to instability and over-consolidation, where the energy efficiency advantages are countered by frequent migrations and unequal load distribution. These findings support the balanced operating point represented by our default thresholds (Sec. 4).

TABLE 10: Sensitivity analysis of thresholds for VM allocation and migration techniques

T	T_{low}	T_{high}	Migrations	Active hosts	Utilization variance
0.3	0.05	0.6	1256	1129	0.023
0.6	0.10	0.7	2702	978	0.018
0.8	0.20	0.8	4502	1219	0.031

5.9 Comparison with the Closest Rivals

Table 11 provides a comparison of the proposed method against closest rivals. Using PerficientCloudSim, we re-implemented all these methods and the WorkloadAware approach using the same simulation setup and experimental parameters to ensure fair evaluation with respect to resource heterogeneity and interference constraints. The outcomes show a clear trend in the performance of different scheduling and migration strategies across three workloads (W1, W2, W3). In terms of E and P, the traditional FCFS and FF approaches show the highest values, reflecting their relatively inefficient resource utilization, with FCFS consuming up to 2232.68 KWh and taking 2279.4 min for W3. FillUp improves both metrics by consolidating workloads more effectively, while SAMR, WAVMCM, WarMops, and the Game-based approach further optimize energy and performance through smarter migration policies, reducing energy consumption and execution times incrementally. With W3 requiring only 723 migrations, 1808.41 KWh, and 1798.48 min, the WorkloadAware approach consistently achieves the lowest number of migrations (M) and least energy consumption, execution times across all workloads. This suggests that, in comparison to heuristic/traditional approaches, workload prediction and awareness greatly improve scheduling efficiency, lower migration overhead, and better balance the energy-performance tradeoff. Overall, the findings show that WorkloadAware is the best method for managing cloud resources in a way that maximizes performance and saves energy.

6 Related Work

In [32], a workload management framework (sCloud) is designed to optimize system performance in distributed datacenters using renewable energy. The framework considers diverse workloads,

TABLE 11: Comparison of WorkloadAware against state-of-the-art methods [Lower N_T is better]

Method	E (KWh)	P (min)	M	N_T	E (KWh)	P (min)	M	N_T	E (KWh)	P (min)	M	N_T
	W1				W2				W3			
FCFS	754.27	550.11	5231	6.52	1615.83	1557.9	2391	15.87	2232.68	2279.4	1975	22.56
FF	723.32	513.91	4123	6.19	1585.44	1522.35	2234	15.54	2047.37	2062.65	1856	20.55
FillUp	636.20	412.01	3382	5.24	1532.48	1460.41	2123	14.96	1860.46	1844.03	1822	18.52
SAMR [28]	655.00	450.43	2812	5.53	1566.21	1471.56	2211	15.19	1901.43	1891.78	2023	18.97
WAVMCM [29]	630.56	405.21	2121	5.18	1526.89	1432.99	1994	14.8	1823.12	1828.87	1799	18.28
WarMops [30]	648.21	430.21	2556	5.39	1512.9	1441.5	2019	14.77	1826.78	1829.01	1801	18.28
Game approach [31]	640.44	420.76	2319	5.31	1509.66	1431.94	2004	14.71	1821.04	1802.55	1789	18.12
WorkloadAware	626.54	400.71	1222	5.14	1498.17	1420.21	888	14.59	1808.41	1798.48	723	18.03

including batch and transactional processes, and dynamically allocates and migrates workloads based on job execution requirements, power availability, and QoS limitations. It uses a nonlinear programming solution to optimize performance while adjusting to different renewable energy sources. The framework also features an adaptable batch job manager that uses reinforcement learning to adjust task execution. Experimental data from a real cloud testbed shows that sCloud reduces QoS violations and increases system throughput; however, its dependence on nonlinear programming that might restrict scalability for large-scale, real-time cloud systems.In [33], an Application awaRE workload Allocation (AREA) technique is proposed to optimize workload allocation in cloudlet-based edge computing systems. AREA considers network and processing latency to reduce IoT application's response time. The authors propose a heuristic technique that dynamically modifies computer resources for various applications. AREA significantly lowers average response time compared to current techniques like latency-based allocation and density-based clustering, especially when workloads are heavy and cloudlet capacities fluctuate. Even if AREA efficiently speeds up response times in cloudlet-based IoT, however, its heuristic-based methodology might not be able to handle workloads that require quick adaptation [34].

DCloud [35] is a novel cloud resource allocation framework designed to enhance job acceptance rates and resource usage while ensuring deadlines for cloud computing positions. It employs bandwidth scaling and time sliding mechanisms to optimize resource efficiency and balance demand peaks. DCloud uses a strategy-proof pricing approach to encourage tenants to make honest resource demands. Testbed experiments and simulations show that DCloud outperforms traditional allocation techniques like virtual cluster models and Amazon EC2, while doubling task acceptance rates, increasing bandwidth and VM utilization, and boosting cloud provider income by approximately 50%. Although, DCloud improves resource allocation and is aware of deadlines, but it makes the assumption that job execution estimates are correct, which could not be always true in unpredictable cloud environments.

In [29], a Workload Aware VM Consolidation Method (WAVMCM) is proposed to minimize energy consumption and maximize resource usage in edge/cloud computing. It involves hibernating unproductive hosts and dynamically assessing workload allocation to migrate VMs for energy efficiency. The method uses a migration approach to reduce performance degradation and a classification-based heuristic to distribute VMs optimally. Compared to genetic algorithm and simulated annealing techniques, WAVMCM shows significant power savings and reduced number of active hosts. However, its heuristic consolidation approach is not flexible enough for real-time applications with varying workloads. In [28], SAMR (Skewness-Avoidance Multi-Resource Allocation) is suggested for optimizing IaaS resource use in clouds. It addresses inefficiencies in current resource allocation methods, reducing resource waste when workloads differ in CPU, memory, and bandwidth demands. SAMR dynamically distributes diverse workloads to hosts while reducing resource skewness. The framework uses a prediction model based on Markov Chains to find the ideal number of active hosts to meet workload needs while consuming less energy. Experiments show that SAMR significantly increases cost-effectiveness, workload balancing, and resource efficiency in clouds. Although, SAMR optimizes the distribution of many resources in IaaS clouds, it does not completely take network constraints into consideration, which might affect performance when traffic loads are high.

WarMops is a workload-aware resource allocation technique designed to optimize runtime resource management in private clouds [30]. It integrates resource reserving and sharing techniques to analyze and categorize VMs workloads, dynamically. The strategy includes a resource configuration optimization algorithm, a dynamic resource management system, and a monitoring module. WarMops is validated using the RUBiS benchmark application, ensuring short response times and reducing resource waste. Experiments show that WarMops can save up to 75% of CPU and 50% of memory without affecting performance. However, it might not be able to manage dynamic workload variations due to static workload categorization. LA-RATS [36] is a framework for task scheduling and load-aware resource allocation for cloudlet-based Multi-Cluster Computing (MCC) systems. It optimizes system performance by dynamically allocating resources and scheduling jobs for delay-sensitive and delay-tolerant applications. The framework consists of a tree-based backfilling strategy, a genetic algorithm-based task migration mechanism, and a multi-objective DAG-based scheduling algorithm. Simulations show that LA-RATS decreases turnaround time for delay-tolerant applications, increases deadline fulfillment rate for delay-sensitive tasks, and lowers monetary costs. LA-RATS is effective to maximize resource utilization, however, its genetic algorithm based migration technique could result in an excessive amount of processing overhead for real-time applications.

CloudMap is a provisioning system that uses workload characteristics and resource heterogeneity to improve workload distribution in private clouds [37]. It uses a hybrid architecture with a global and a local placement manager to match workloads to servers based on resource demands. This approach reduces resource waste and ensures performance, compared to traditional techniques like round-robin and heterogeneity-aware approaches. Experimental evaluations using real-world traces show that CloudMap significantly lowers resource violations, enhances server utilization, and reduces the number of active servers. Though its two-layered decision-making process may raise placement latency and response time in large datacenters, CloudMap improves workload placement in diverse private clouds. [31] proposes a new coalitional game-theoretic method for VM consolidation in heterogeneous clouds. The method divides hosts based on workload levels, uses a coalitional-game-based VM consolidation algorithm to select hosts, and performs VM migrations to maximize payoff and maintain energy efficiency. The method outperforms conventional methods in terms of energy savings and load balancing. Although, it enhances load balancing and energy efficiency in heterogeneous

TABLE 12: Summary of the related works

Work	Placement	Migrations	Performance	Energy	Runtimes	Priority	Predictions	Costs
[32]	Workload-aware	Optimization	QoS	✓		QoS	✓	
[33]	Cloudlet-based	Network conditions	Low latency			Delay-sensitive		✓
[35]	Deadline-aware	Bandwidth scaling	Acceptance rate			Deadline	✓	✓
[29]	Classification-based	Reduces migrations	Workload efficiency	✓		Energy-focused		
[28]	Multi-resource	Balances workloads	Hosts				✓	✓
[30]	Resource-aware	Reactive	Resource wastage					✓
[36]	Load-aware	Cloudlets and clouds	Deadline satisfaction	✓		Delay-sensitive		✓
[37]	Heterogeneity-aware	Global/local decisions	SLA					✓
[31]	Coalitional game-based	Merge-and-split	Workload fairness	✓		✓	✓	
[38]	Fluctuation-based	Reduces migrations	Placement failure	✓			✓	
[39]	Workload-based	Offloading	Delay	✓			✓	
[40]	Workload-based	Reduces migrations	SLA	✓			Rule-based	
This	Workload-aware	Workload-aware	Exec. Time	✓	✓	✓	✓	✓

clouds, its merge-and-split strategy may result in a substantial migration overhead that could compromise application stability and real-time system performance.

In [41], authors have studied various issues related to resource allocation in heterogeneous clouds in terms of performance, CPU architecture, and energy efficiency. They contend that treating all resources equally results in cost and performance inefficiencies. To improve job runtime and overall utilization, they suggest scheduling systems that are aware of heterogeneity and take resource diversity and workload characteristics into account. A time-slotted optimization methodology for task offloading in cloud-edge systems is developed in [42]. Under time-varying network conditions, the model simultaneously optimizes communication, task scheduling, and distribution of computational resources. [43] examined issues in coexisting Wi-Fi and Bluetooth for IoT-based smart homes where both technologies compete for limited spectrum resources. To reduce interference and enhance service quality, authors suggested a service-oriented resource allocation and scheduling system that dynamically coordinates bandwidth and task execution. The system meets the latency and throughput needs of various IoT applications while ensuring equitable allocation through adaptive channel management and priority-based scheduling. In [39], authors have solved the computational offloading problem by using workload-aware modeling based on DRL approach to minimize delay and energy consumption. In [40], authors focused on addressing the VM consolidation problem using a unified framework that consists of two phases. The first phase deals with host workload detection and prediction (decided overloaded and underloaded hosts), while the second phase tackles the selection and allocation of appropriate VMs.

In [38], a workload fluctuation-based VM allocation method is proposed that predicts the future resource demand using VM workload prediction and categorizes VMs for no change, requiring more resources, or requiring fewer resources. Well-known heuristics, i.e., best fit and worst fit, are implemented to evaluate its performance. Results show that VM migrations are minimized and resources are used efficiently. Current resource allocation methods usually make the assumption that clusters are homogeneous, which oversimplifies the actual cloud environment. With servers with varying CPU architectures, performance levels, and energy footprints, datacenters are actually quite diverse. In addition, contemporary workloads with a wide range of resource requirements, like video streaming, AI inference, and IoT applications, are service-driven. Workload-aware allocation and migration techniques that adjust to task and infrastructure variation are required due to this heterogeneity. Table 12 provides a summary of the comparison between our suggested approach "WorkloadAware" and other closely comparable research. We think the comparison would make it easier for readers to spot areas that need more investigation right away. The system ensures near-optimal workload performance across several distributed datacenters by efficiently managing the intermittency of renewable energy.

7 Conclusions and Future Work

In this paper, we proposed a workload-aware resource allocation policy that accounts for resource heterogeneity, workloads, and a machine-learning approach to predict future workloads. The suggested approach was validated using Google workloads with various prediction methods. Through empirical evaluation, we found that our suggested method can provide about 5.25% energy savings and 8.17% workload performance increases compared to non workload-aware approaches. Besides an existing trade-off between workload performance and energy consumption, a workload-aware migration approach can reduce energy consumption (6.42% – 18.3%) and improve performance (7.8% – 25.03%) for certain applications.

In the future, we will use other machine-learning methods to train more sophisticated models from the host-profiled data to categorize all the servers based on the workloads they can run more efficiently in terms of energy consumption and performance. Moreover, we will consider performance degradation and energy efficiency due to hardware heterogeneities and co-located workloads (running inside VMs/containers) that compete for the same resources. Future research will expand this approach to service-based allocation, as our results show advantages of workload-aware allocation and migration in heterogeneous datacenters. Workload-aware policies can be directly linked to latency-sensitive services like video streaming, AR/VR, and real-time analytics in autonomous vehicles, guaranteeing energy efficiency and quality of service. In this work, we assumed heterogeneity primarily through CPU architectures. For I/O-intensive workloads, heterogeneity in storage (e.g., SSD vs. HDD IOPS constraints) and network bandwidth may affect the performance of data-intensive tasks that we must consider in the future.

We plan to evaluate the proposed system on a real OpenStack testbed or integrate it in a Kubernetes scheduler. We believe that our approach's modular design enables realistic deployment with only minor scheduling layer adjustments. Furthermore, using transformer-based models for dynamic workload prediction might be an interesting work. The WorkloadAware framework can be modified for decentralized architectures such that multiple local schedulers, each controlling a subset of hosts, can take the role of a global scheduler. Each host can carry out workload prediction and VM categorization locally, and ranking scores can be calculated according to the availability of local resources. This modification guarantees that large-scale datacenters can benefit from workload-aware allocation and migration without compromising fault tolerance and scalability.

Acknowledgment

This work is financially supported, in part, by Sohar University, Oman, University of Khorfakkan, UAE, and, in part, by Abdul Wali Khan University, Pakistan.

REFERENCES

[1] T. Chakraborty, C. Kopp, and A. N. Toosi, "Optimizing renewable energy utilization in cloud data centers through dynamic overbooking: An mdp-based approach," *IEEE Transactions on Cloud Computing*, 2024.

[2] M. Zakarya and L. Gillam, "Energy and performance aware resource management in heterogeneous cloud datacenters." Ph.D. dissertation, University of Surrey, 2017.

[3] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, 2013.

[4] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[5] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[6] L. J. Gadhavi and M. D. Bhavsar, "Adaptive cloud resource management through workload prediction," *Energy Systems*, vol. 13, no. 3, pp. 601–623, 2022.

[7] T. Khan, W. Tian, S. Ilager, and R. Buyya, "Workload forecasting and energy state estimation in cloud data centres: Ml-centric approach," *Future Generation Computer Systems*, vol. 128, pp. 320–332, 2022.

[8] D. Saxena, J. Kumar, A. K. Singh, and S. Schmid, "Performance analysis of machine learning centered workload prediction models for cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1313–1330, 2023.

[9] D. Saxena and A. K. Singh, "Workload pattern learning-based cloud resource management models: Concepts and meta-analysis," *IEEE Transactions on Sustainable Computing*, 2024.

[10] T. C. Ferreto, M. A. S. Netto, R. N. Calheiros, and C. A. F. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1027–1034, 2011.

[11] B. Hu, Y. Shi, G. Chen, Z. Cao, and M. Zhou, "Workload-aware scheduling of real-time jobs in cloud computing to minimize energy consumption," *IEEE Internet of Things Journal*, vol. 11, no. 1, pp. 638–652, 2023.

[12] A. A. Khan, M. Zakarya, R. Buyya, R. Khan, M. Khan, and O. Rana, "An energy and performance aware consolidation technique for containerized datacenters," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1305–1322, 2019.

[13] M. Zakarya, L. Gillam, A. A. Khan, and I. U. Rahman, "Perficient-cloudsim: a tool to simulate large-scale computation in heterogeneous clouds," *The Journal of Supercomputing*, vol. 77, pp. 3959–4013, 2021.

[14] Z. Zhou, M. Shojafar, M. Alazab, J. Abawajy, and F. Li, "Afed-ef: An energy-efficient vm allocation algorithm for iot applications in a cloud data center," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 2, pp. 658–669, 2021.

[15] A. Beloglazov and R. Buyya, "OpenStack Neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds," *Concurrency Computation*, vol. 27, no. 5, pp. 1310–1333, 2015.

[16] J. O'Loughlin and L. Gillam, "Performance evaluation for cost-efficient public infrastructure cloud use," in *International Conference on Grid Economics and Business Models*. Springer, 2014, pp. 133–145.

[17] N. Kumar, M. Singh, A. Ahmad, and J. J. Rodrigues, "Machine learning-based workload-aware vm consolidation in cloud data centers," *Future Generation Computer Systems*, vol. 128, pp. 38–52, 2022.

[18] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, I. Porres, and H. Tenhunen, "Energy-efficient vm consolidation in cloud data centers using utilization prediction model," in *2015 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 390–397.

[19] B. Feng and Z. Ding, "Application-oriented cloud workload prediction: A survey and new perspectives," *Tsinghua Science and Technology*, vol. 30, no. 1, pp. 34–54, 2024.

[20] J. Gao, H. Wang, and H. Shen, "Machine learning based workload prediction in cloud computing," in *2020 29th international conference on computer communications and networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[21] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[22] A. Javeed, A. Borg, H. Grahn, L. Lundberg, D. Patel, and S. Shirinbab, "Improving cloud efficiency: A machine learning-based stacking model for cpu utilization prediction," in *2025 8th International Conference on Data Science and Machine Learning Applications (CDMA)*. IEEE, 2025, pp. 120–125.

[23] N. Shafi, M. Abdullah, W. Iqbal, and F. Bukhari, "Dima: machine learning based dynamic infrastructure management for containerized applications," *Computing*, vol. 107, no. 3, pp. 1–30, 2025.

[24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[25] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *Proceedings of the fifteenth European conference on computer systems*, 2020, pp. 1–14.

[26] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., Mountain View, CA, USA, Technical Report*, 2011.

[27] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2470–2483, 2016.

[28] L. Wei, C. H. Foh, B. He, and J. Cai, "Towards efficient resource allocation for heterogeneous workloads in iaas clouds," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 264–275, 2015.

[29] I. Mohiuddin and A. Almogren, "Workload aware vm consolidation method in edge/cloud computing for iot applications," *Journal of Parallel and Distributed Computing*, vol. 123, pp. 204–214, 2019.

[30] J. Zhang, J. Wang, J. Wu, Z. Lu, S. Zhang, and Y. Zhong, "Warmops: a workload-aware resource management optimization strategy for iaas private clouds," in *2014 IEEE International Conference on Services Computing*. IEEE, 2014, pp. 575–582.

[31] X. Xiao, W. Zheng, Y. Xia, X. Sun, Q. Peng, and Y. Guo, "A workload-aware vm consolidation method based on coalitional game for energy-saving in cloud," *Ieee Access*, vol. 7, pp. 80 421–80 430, 2019.

[32] D. Cheng, X. Zhou, Z. Ding, Y. Wang, and M. Ji, "Heterogeneity aware workload management in distributed sustainable datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 375–387, 2018.

[33] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.

[34] H. Yuan, J. Bi, S. Li, J. Zhang, and M. Zhou, "An improved lstm-based prediction approach for resources and workload in large-scale data centers," *IEEE Internet of Things Journal*, 2024.

[35] D. Li, C. Chen, J. Guan, Y. Zhang, J. Zhu, and R. Yu, "Dcloud: deadline-aware resource allocation for cloud computing jobs," *IEEE transactions on parallel and distributed systems*, vol. 27, no. 8, pp. 2248–2260, 2015.

[36] F. Zhang, J. Ge, Z. Li, C. Li, C. Wong, L. Kong, B. Luo, and V. Chang, "A load-aware resource allocation and task scheduling for the emerging cloudlet system," *Future Generation Computer Systems*, vol. 87, pp. 438–456, 2018.

[37] B. Viswanathan, A. Verma, and S. Dutta, "Cloudmap: workload-aware placement in private heterogeneous clouds," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 9–16.

[38] F. Montazerin and A. Shameli-Sendi, "Enhancing virtual machine placement efficiency in cloud data centers through fluctuations-aware resource management," *Computers and Electrical Engineering*, vol. 122, p. 109885, 2025.

[39] "Multiobjective offloading optimization in fog computing using deep reinforcement learning, author=Mashal, Hojjat and Rezvani, Mohammad Hossein, journal=Journal of Computer Networks and Communications, volume=2024, number=1, pages=6255511, year=2024, publisher=Wiley Online Library."

[40] "Efficient cloud data center: An adaptive framework for dynamic virtual machine consolidation, author=Rozehkhani, Seyyed Meysam and Mahan, Farnaz and Pedrycz, Witold, journal=Journal of Network and Computer Applications, volume=226, pages=103885, year=2024, publisher=Elsevier."

[41] G. Lee and R. H. Katz, "Heterogeneity-Aware resource allocation and scheduling in the cloud," in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 11)*. Portland, OR: USENIX Association, Jun. 2011.

[42] W. Fan, X. Liu, H. Yuan, N. Li, and Y. Liu, "Time-slotted task offloading and resource allocation for cloud-edge-end cooperative computing networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 8, pp. 8225–8241, 2024.

[43] D. Zhao, T. Niu, B. Song, and X. Du, "Service-oriented resource allocation and task scheduling for wi-fi and bluetooth coexistence in

smart home iot systems," *ACM Trans. Internet Things*, vol. 6, no. 2, Mar. 2025.

Muhammad Zakarya received the PhD degree in Computer Science from the University of Surrey, Guildford, U.K. He is currently an assistant professor with the Faculty of Computing and Information Technology (FCIT), Sohar University, Oman, and also with the Department of Computer Science, Abdul Wali Khan University Mardan (AWKUM), Pakistan. He is a senior member of the IEEE. Dr. Zakarya is a TPC member of a few prestigious international conferences, including CCGrid, GECON, ICCCI, FIT, and UCC. He is also an associate editor of several journals in IEEE, Elsevier, and Springer.

Ayaz Ali Khan received the PhD degree in Computer Science from the Abdul Wali Khan University, Pakistan. He is currently an assistant professor with the Department of Computer Science, University of Lakki Marwat, Pakistan. His research interests include cloud computing, mobile edge clouds, the Internet of Things (IoT), performance, energy efficiency, algorithms, and resource management. He has a deep understanding of theoretical computer science and data analysis. Furthermore, he also owns a deep understanding of various statistical techniques that are largely used in applied research. His research has appeared in several international conferences, journals, and transactions of repute.

Lee Gillam is a senior lecturer in the Department of Computer Science at the University of Surrey, UK, a Chartered IT Professional Fellow of the British Computer Society (FBCS CITP), and a member of the EPSRC Peer Review College. He is the founding editor-in-chief of Springer's Open Access Journal of Cloud Computing Advances, Systems and Applications (JoCCASA), which published its first articles in April 2012, and which followed on from co-editorship of the first Springer book on cloud computing in 2010. He has been a co-author of two reports for EPSRC/JISC on cloud computing.

Omer Rana is a Professor of Performance Engineering in the School of Computer Science & Informatics at Cardiff University and Deputy Director of the Welsh e-Science Centre. He holds a PhD from Imperial College. His research interests extend to three main areas within computer science: problem-solving environments, high-performance agent systems, and novel algorithms for data analysis and management. He is an Associate Editor for Transactions on Parallel and Distributed Systems (TPDS), IEEE.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the university, commercializing its innovations in Cloud Computing. He has authored over 625 publications and seven textbooks, including "Mastering Cloud Computing," published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese, and international markets, respectively. He is one of the most highly cited authors in computer science and software engineering worldwide (h-index=135, g-index=298, 96,200+ citations). Dr. Buyya has been recognized as a "Web of Science Highly Cited Researcher" for four consecutive years since 2016, a Fellow of IEEE, and Scopus Researcher of the Year 2017 with the Excellence in Innovative Research Award and the 2020 Khwarizmi International Award for his outstanding contributions to cloud computing and distributed systems.