Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2024 SCPE. Volume 25, Issues 3, pp. 1963–1979, DOI 10.12694/scpe.v25i3.2689

ENHANCED FEATURE-DRIVEN MULTI-OBJECTIVE LEARNING FOR OPTIMAL CLOUD RESOURCE ALLOCATION

UMA MAHESWARA RAO I* AND JKR SASTRY†

Abstract. In cloud networks, especially those with distributed computing setups and data centers, one of the biggest obstacles is allocating resources. This is the key area, and this must be balanced between optimizing system performance on one side and affordability, stability (reliance) of operation, and energy efficiency. The importance of improving resource allocation methodologies in these complex cloud computing systems is recognized, and therefore this paper comes with an appropriate title–"Enhanced Feature-Driven Multi-Objective Learning for Optimal Cloud Resource Allocation" (OCRA), which integrates together both the latest machine learning techniques as well as traditional concepts from research into cloud computing. OCRA capably analyzes historical files on CPU, memory, disk and network usage. In addition to neatly assimilating large data sets such as that was the compliance rate with past SLAs or workload frequencies over certain time periods and resource allocations; even their patterns of service requests are an important piece of information for many busy people's lives today the adaptive mechanism is one of the defining traits of the model. It can accurately anticipate changes in resource demand and immediately adjust supply, fully able to respond rapidly when fluctuations arise suddenly or unexpectedly. Multi-Objective Random Forests are at the very core of OCRA. Each tree for decision making is specially designed to meet a particular performance objective in mind. Combining these trees into a Random Forest ensemble increases not only the model's predictive accuracy but also its stability. Pareto optimization is wisely used to maintain a balance among performance indicators, without an excessive focus on one effect alone. OCRA is proven empirically through experimental studies where key performance indicators such as Resource Utilization Rate and Quality of Service (QoS) Adherence Rate are taken into account. OCRA is both energy-efficient, an important attribute in today's environmentally conscious world, and does not sacrifice performance. As far as speed, flexibility and overall efficiency are concerned, OCRA has always been superior to the other cloud resources allocation programs of its own day. While it's still not quite ready for users who don't have a firm background in computer science or programming skills (ocra is plotted on 0-x), with sufficient memory and dominant minutes turn into mechanical equipment without configuration services

Key words: QoS, Optimal Cloud Resource Allocation, Driven Multi-Objective Learning, historical SLA, cloud computing, virtual machine, Ant Colony Optimization .

AMS subject classifications.

1. Introduction.

Introduction. The use of cloud computing in modern technology has grown rapidly [1]. For a wide range of services and applications, this paradigm offers scalable computational and storage resources. The efficient allocation of cloud resources becomes increasingly crucial and challenging as the domain expands [2]. Previously, cloud platforms were believed to be enormous reservoirs of computing and storage resources. These platforms must, however, allocate resources wisely to operate at the best possible rate given the exponential growth in demand. In the highly competitive cloud service market, inefficient allocation can result in higher operational costs and a worse user experience, which is a crucial metric [3]. The need for more agile solutions arises from the inability of traditional static and rule-based resource allocation strategies to address the dynamic nature of modern workloads.

The Optimal Cloud Resource Allocation (OCRA) model is a response to this pressing need. Advanced resource rate is combined with traditional resource allocation techniques. The model uses Multi-Objective Random Forests to balance multiple objectives. OCRA employs this method to forecast future resource requirements and anomalies based on historical data. The capacity of the model to recognize intricate correlations between resources ensures a more sophisticated and successful resource allocation strategy. Such a strategy is

*Research Scholar, Dept., of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Andhra Pradesh, India (inkolluchanti@gmail.com, Corresponding author)

†Professor, Dept., of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Andhra Pradesh, India (drsastry@kluniversity.in).

essential. By ensuring efficient resource utilization, cloud platforms can enhance the user experience, reduce operational costs, and maintain service quality. By applying a forward-thinking viewpoint, machine learning enables cloud platforms to plan ahead and anticipate future MS demands instead of merely responding to them.

In this article on cloud resource allocation, the challenges of contemporary cloud platforms are explained. A comparative analysis comparing OCRA to other widely used methodologies will be provided by an empirical study. The purpose of this article is to lay the groundwork for future research and development in the field by providing a structured understanding of current challenges in cloud resource allocation and potential solutions.

Following this introduction, Section 2 delves into a detailed literature review, highlighting the evolution of cloud resource management strategies. Section 3 meticulously details the architecture and inner workings of OCRA. Section 4 presents an exhaustive experimental study, offering empirical evidence of OCRA's superiority. Concluding remarks and potential avenues for future research are discussed in Section 5.

2. Related Work.. Zuo, Li-Yun et al. [4] proposed an integrated ant colony optimization algorithm to address the challenges of cloud computing complexity and resource uncertainty. This advanced algorithm incorporates entropy for measuring resource uncertainty and enhances collaboration among ants through global pheromone updates. It also employs a Min-min algorithm-derived heuristic for minimizing activation time and load balance adjustments. The algorithm's superior performance in time scheduling and load balancing is validated through cloud simulation experiments.

TAI, Li et al., [5] proposed a dynamic scheduling approach to tackle manufacturing resource scheduling in cloud manufacturing environments. The authors present a manufacturing resource scheduling method that combines genetic and ant colony algorithms to quickly and accurately converge to optimal solutions. Simulation results validate the effectiveness of the proposed algorithm. Hui Jiang et al. [6] introduced a cloud-based disassembly system for waste electronic equipment. They employ a multi-objective genetic algorithm to minimize makespan and cost while considering the uncertainty of the disassembly process. The proposed algorithm generates Pareto optimal solutions, providing users with choices for preferred disassembly services and proves its effectiveness in solving task scheduling and resource allocation in cloud-based disassembly.

Yuan, S. U. N., et al., [7] addressed the challenges of optimizing spectrum efficiency, energy efficiency, and front haul efficiency in Cloud Radio Access Networks (C-RANs). They propose joint optimization algorithms, including a Lagrange dual decomposition method and a Modified Particle Swarm Optimization (M-PSO) algorithm, to achieve multi-objective optimization. Simulation results demonstrate the effectiveness of these algorithms in balancing conflicting network optimization goals.

Prasad Devarasetty et al., [8] focused on efficient resource allocation in cloud computing to reduce energy consumption and minimize costs. The authors propose a multi-objective Ant Colony Optimization (ACO) algorithm, which outperforms existing approaches in terms of resource utilization, makespan, and energy consumption. Statistical tests confirm the algorithm's superiority, providing a robust solution to the resource allocation problem in cloud computing.

Mahendra Bhatu Gawali et al., [9] focused is on task scheduling and resource allocation in cloud computing. Cloud computing offers shared resources accessible over the internet, but effective resource allocation is crucial for optimal performance. Existing methods often overlook preemption and varying task sizes, leading to delays and underutilized resources. To address these issues, the article introduces a heuristic approach that incorporates task preemption and employs a combination of techniques, including modified analytic hierarchy process and divide-and-conquer. The aim is to enhance scheduling and allocation in cloud computing, with the goal of improving performance metrics like turnaround time and response time. The article presents the proposed approach's effectiveness through comparisons with existing frameworks, showcasing its potential to offer a more efficient solution for cloud computing resource management.

Mahbuba Afrin et al. [10] developed into the realm of resource allocation for robotic workflows in smart factories. This article tackles the challenge of optimal resource allocation in scenarios involving multiple robots and Cloud instances collaborating under constraints related to energy consumption and cost. The primary objectives are to optimize makespan, energy consumption, and cost while efficiently allocating resources for robotic workflow tasks. To address these complex optimization goals, the article proposes an Edge Cloud-based system designed to allocate computing resources for robotic tasks in smart factory environments. The study introduces a constrained multi-objective optimization problem, utilizing the NSGA-II algorithm with enhancements. Synthetic workload experiments confirm the proposed approach's effectiveness, outperforming state-of-the-art methods by a substantial margin in optimizing makespan, energy, and cost attributes in various scenarios.

Zhao-Hui Liu et al. [11] tackle the intricate task of resource scheduling in cloud manufacturing, a model characterized by networked manufacturing resources and services. This environment presents challenges due to incomplete, asymmetric, and non-transparent information exchange, making optimized resource scheduling a formidable task. Geographic distribution differences, logistics costs, and user preferences further complicate the issue. To address these challenges, the article presents an iterative double auction mechanism rooted in game theory. This mechanism aims to optimize resource allocation, balance the interests of resource demanders and providers, and prevent harmful market behaviors. The article's main contribution is this game-theoretical approach, designed to enhance the efficiency of resource allocation in cloud manufacturing systems while ensuring economic benefits for participants. Simulation experiments demonstrate its effectiveness, showing improved resource allocation, reduced costs, and enhanced service quality.

Prassanna J et al., [12] the challenge of load balancing in cloud server environments due to unpredictable bursty workloads is addressed. Traditional load balancing algorithms often struggle with sudden spikes in user requests, impacting scheduling efficiency, energy consumption, and response time. Inadequate load balancing can also result in uneven resource distribution, leading to user dissatisfaction and increased service costs. The article proposes a novel task scheduling technique called Threshold Based Multi-Objective Memetic Optimized Round Robin Scheduling (T-MMORRS). This technique leverages a burst detector to assess workload conditions and select the most suitable load balancing algorithm. T-MMORRS combines the Threshold Multi-Objective Memetic Optimization (TMMO) and Weighted Multi-Objective Memetic Optimized Round Robin Scheduling (WMMORRS) algorithms to optimize task scheduling for improved efficiency, reduced energy consumption, and enhanced performance compared to existing load balancing methods.

AM Senthil Kumar et al. [13] explored resource allocation in cloud computing environments, focusing on the demand for resources and computation. They propose a Hybrid Genetic Ant Colony Optimization algorithm, which combines Genetic Algorithm (GA) and Ant Colony Optimization (ACO) to improve multi-objective resource allocation. This hybrid algorithm enhances GA solutions with ACO before the selection operation, effectively addressing resource allocation issues in cloud computing environments. The algorithm considers and optimizes Quality of Service (QoS) parameters like response time, completion time, makespan, and throughput. Experimental results demonstrate the superior performance of this Hybrid Genetic Ant Colony Optimization algorithm compared to conventional optimization techniques, making it a promising solution for efficient resource allocation.

M. Alamelu et al. [14] tackled the challenge of efficiently allocating available resources to execution tasks in cloud computing. Cloud computing's dynamic nature requires optimal resource allocation to achieve optimal machine utilization, reduce energy consumption, and provide reliable resources. The article introduces a dynamic approach that leverages all Quality of Service (QoS) outputs to achieve these objectives. It employs a load balancing algorithm inspired by bee behavior to address limitations in existing research, which often focuses on optimizing single aspects of cloud computing without considering interconnections. Real-time Eucalyptus cloud-based performance evaluations demonstrate the effectiveness of this approach, showcasing improvements in computational time, reaction time, makespan, load variability, and imbalance levels compared to existing algorithms.

Murali Mohan Vutukuru et al. [15] focused on optimizing resource scheduling strategies in cloud computing environments, with a specific emphasis on Quality of Service (QoS). Cloud computing has gained popularity due to its scalability and cost-effectiveness, but efficient resource allocation is crucial. The authors aim to design scheduling solutions capable of detecting suitable resource matches and client-specific workload prerequisites. They propose multi-objective resource scheduling strategies that take into account QoS, idle intervals, and batch scheduling, aiming to maximize resource utilization and scheduling efficiency while improving response times and minimizing resource wastage.

Ramasubbareddy Somula et al. [16] introduced the concept of Multi-Objective Genetic Algorithm-Based Resource Scheduling (MOGALMCC). MOGALMCC utilizes genetic algorithms to balance virtual machine (VM) load among cloudlets, enhancing application performance in terms of response time. By considering factors like distance, bandwidth, memory, and cloudlet server load, MOGALMCC seeks optimal cloudlet allocation before scheduling VMs. This framework aims to minimize VM failure rates, reduce execution time, and decrease task waiting times on the server.

Bela Shrimali et al. [17] addressed resource allocation in cloud environments with a focus on energy efficiency. As data centers worldwide consume increasing amounts of energy, the authors propose a multi-objective optimization (MOO)-based technique for resource allocation. This technique simultaneously optimizes resource allocation in terms of performance and energy efficiency. By achieving this balance, it reduces energy consumption while meeting Service Level Agreements (SLAs) set by customers. The article's contribution lies in introducing a comprehensive framework that considers both performance and energy efficiency, thereby providing an effective means of resource management in cloud environments.

J. Arravinth et al. [18] tackled the challenge of meeting increased user demands in cloud computing by introducing the concept of inter-cloud resource sharing. This approach utilizes multiple cloud service providers to address resource limitations in individual clouds. The authors propose a multi-agent approach called "multi-agent with multi-objective optimized resource allocation on inter-cloud" (MOGARIC). MOGARIC combines adaptive tree seed optimization (ATSO) and multi-objective optimization to efficiently allocate cloud resources in inter-cloud environments. By minimizing makespan, cost, and maximizing resource utilization, MOGARIC improves resource allocation and service performance. Experimental results demonstrate the superiority of MOGARIC over existing approaches in terms of makespan, cost efficiency, and resource utilization.

George et al., [19] the focused is on addressing resource allocation challenges in cloud computing. The heterogeneous nature of cloud resources adds complexity to the allocation problem. Efficient allocation of resources is crucial to process a large number of task requests while maintaining high-quality service standards (QoS). This article introduces a Multi-objective Auto-encoder Deep Neural Network-based (MA-DNN) method that combines Sen's Multi-objective functions and Auto-encoder Deep Neural Network models to enhance resource allocation efficiency in cloud computing. The primary goal is to efficiently allocate resources while improving QoS by reducing task scheduling time and increasing task scheduling efficiency. The proposed method significantly outperforms existing algorithms in experimental tests, demonstrating its potential to enhance resource allocation in cloud computing.

S. Ramamoorthy et al. [20] discussed resource scheduling in cloud computing infrastructure-based services. They emphasize that resource scheduling is often treated as a single-objective problem, although it inherently involves multiple objectives. They propose the MCAMO technique, a novel approach that handles multi-objectives and constraints during resource scheduling in infrastructure-based cloud services. The MCAMO technique aims to reduce user billing costs and increase cloud service provider revenue. It considers job constraints and client objectives, determining resource allocation using a fitness value approach. The method's performance is evaluated against existing multi-objective VM machine scheduling techniques, and it demonstrates superior resource scheduling optimization.

Gola, Kamal Kumar et al. [21] addressed the challenge of resource allocation in cloud computing with a focus on Quality of Service (QoS). They introduce a novel Multi-objective Hybrid Capuchin Search with Genetic Algorithm (MHCSGA) based hierarchical resource allocation scheme. This approach optimizes resource utilization, response time, makespan, execution time, and throughput. The allocation process begins with a clustering method, partitioning tasks into clusters and optimizing resource allocation. The proposed algorithm is evaluated using the GWA-T-12 Bitbrains dataset, demonstrating superior makespan performance compared to state-of-the-art methods for varying task volumes (50, 100, 150, and 200 tasks). This research aims to improve resource allocation efficiency in cloud computing while maintaining QoS standards.

Resource allocation is a persistent problem for the cloud computing industry. The extant literature highlights the necessity of the suggested OCRA model by demonstrating that, despite the fact that many solutions address specific aspects of this problem, there is still a sizable gap that calls for an integrative approach. Ant colony optimization was used by Zuo, Li-Yun, et al. [4] to deal with resource uncertainty. OCRA, on the other hand, provides real-time adaptation with both conventional and sophisticated features. An approach to cloud manufacturing scheduling was proposed by TAI, Li et al. [5]. OCRA can be tailored to various cloud environments due to its wide range of applications.

Prasad Devarasetty et al. [8] emphasized the reduction of costs and energy consumption. OCRA surpasses

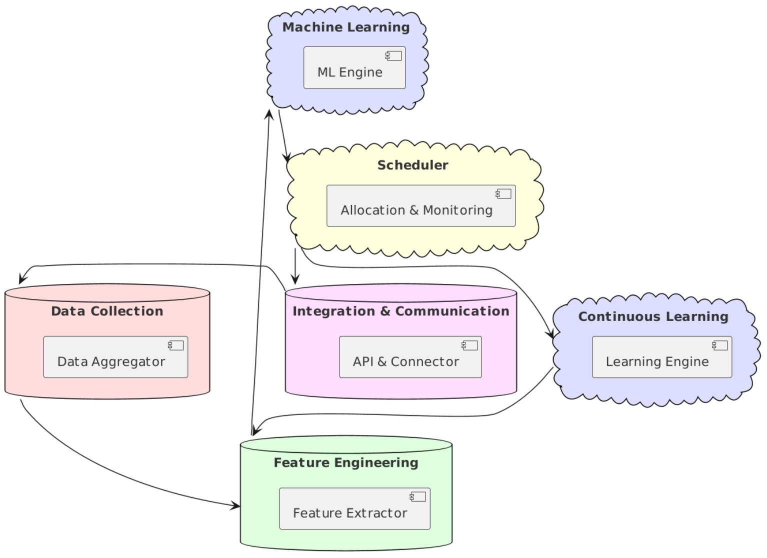

Fig. 3.1: OCRA architecture

these requirements, guaranteeing economical and energy-efficient operations. Multi-objective optimization was utilized by Hui Jiang et al. [6] and M. Alamelu et al. [14] for cloud resource allocation. A thorough solution to many challenges is provided by OCRA's Multi-Objective Random Forests. Robotic workflow metrics were optimized by Mahbuba Afrin et al. [10], and cloud environment optimization is guaranteed by OCRA. The challenges of cloud computing are highlighted in this. The allocation of resources using various approaches was the subject of studies by J. Arravinth et al. [18] and Gola, Kamal Kumar et al. [21]. OCRA is a comprehensive solution for cloud resource challenges because its multi-dimensional approach ensures consistent performance across metrics. While the literature provides multiple solutions, OCRA is a cohesive approach to the various challenges faced by cloud environments. It is a crucial cloud computing solution because of its distinctive features.

OCRA (Optimal Cloud Resource Allocation) is significant to address the limitations of existing models by incorporating a unique blend of both traditional and advanced machine learning techniques. Unlike its predecessors, OCRA excels in real-time adaptability to changing cloud environments, ensuring optimal balance between energy efficiency, cost-effectiveness, and system performance. It utilizes Multi-Objective Random Forests for comprehensive multi-dimensional optimization, addressing a wide spectrum of cloud resource challenges. This model not only promises improved energy and cost efficiency but also maintains high levels of Quality of Service (QoS). By providing a holistic solution that intelligently navigates the complexities and unpredictability of cloud resource allocation, OCRA stands out as a versatile and robust framework suitable for diverse cloud computing scenarios.

3. Method and Materials. The OCRA (Optimal Cloud Resource Allocation) model is a cutting-edge machine learning framework that solves the common problems of cloud resource scheduling. The central aim of OCRA is to overcome the weaknesses in flexibility, efficiency and adaptability of existing models.

The OCRA differentiates itself by adopting a holistic approach that integrates traditional cloud computing indicators with advanced machine learning techniques. Because of this integration OCRA can use historical data like CPU, memory, disk and network usage much more efficiently. In addition to the conventional model, OCRA also considers historical SLA compliance rates, workload types and frequency as well as resource allocation behavior patterns and service request patterns. This results in a better understanding and forecasting of cloud resources.

The OCRA's ability to respond quickly and adapt rapidly is its most important enhancement. This is done through watching rapid changes in resource usage, a quality typically absent from traditional models. OCRA studies historical anomalies and patterns which could serve as predictors of system failures in order to guarantee preemptive maintenance. In addition, its special ability to detect and analyze correlations between different resources makes decisions about allocation more informed and strategic. This greatly enhances the efficiency of cloud resource management.

Multi-Objective Random Forests is the technical backbone of OCRA. The way this is done is to construct individual decision trees, each one tuned for its objective of cost-effectiveness or energy efficiency, QoS (Quality Of Service), resource utilization and so on. These trees are then inserted into a Random Forest ensemble, so that the model's prediction accuracy and stability is enhanced. OCRA uses Pareto optimization to balance among these objectives, so that no single factor dominates the resource allocation process.

Basically, OCRA is a cloud resource management solution which closes the gap between traditional and modern machine learning techniques. This mixture leads to a comprehensive, mature solution that can better solve the complex problems of allocating cloud resources than currently existing models. This innovative approach is illustrated in the detailed architecture of OCRA shown as Figure 3.1. It represents a new breakthrough for cloud computing.

The OCRA architecture allocates cloud resources through connected stages. Starting with the Data Collection module that collects all kinds of use information on resources, it is then refined by Feature Engineering to pick out those most important for prediction. These influences all feed into the Machine Learning engine, where advanced algorithms like Multi-Objective Random Forests create complex models for predicting future resource needs. These predictions are used by the Scheduler component to wisely allocate resources, balancing efficiency against cost and quality of service. Simultaneously, the Integration & Communication module ensures that these assignments run smoothly in cloud infrastructure, and Continuous Learning does a feedback loop to constantly improve on model accuracy and speed. The cyclical process means that resource scheduling at OCRA is always dynamic and efficient, keeping in step with evolving cloud environments.

Data Collection Module. The Data Collection Module [22] serves as the foundational unit of the OCRA architecture. The Historical Data Aggregator [23] is at the forefront, diligently collecting data points related to CPU, memory, disk usage, and a plethora of other traditional metrics. This stream of data is then scrutinized by the Anomaly Detector [24], which flags any deviations or anomalies and identifies their root causes, essentially setting the stage for enriched feature extraction. The Resource Correlation Analyzer further amplifies the module's capability by diving into the intricate interplay between different resources, such as discerning patterns that hint at a surge in memory usage causing an uptick in network traffic.

Historical Data Aggregator. This component is responsible for connecting to the data source to extract metrics like CPU, memory, and disk usage within a specified time window. The extracted data is stored in a structured format for subsequent processing. The mathematical representation for the collected data at any given time can be denoted as:

$$D(t) = CPU(t), Memory(t), Disk(t) \tag{3.1}$$

Anomaly Detector: Using statistical methods or machine learning models, this component identifies deviations from typical patterns. The identified anomalies are labeled and will be used later for feature extraction. If data at a given time t is anomalous, it can be represented as $A(t) = 1$;; otherwise, $A(t) = 0$.

Resource Correlation Analyzer: To discern the interdependence between different resources, pairwise correlations between resources over time are computed. The significant correlations are stored for use in feature extraction. The correlation between two resources, say $Resource_i$ and $Resource_j$, at time $t$is given by:

$$C_{(i,j)} = corr(Resource_i(t), Resource_j(t)) \tag{3.2}$$

Feature Engineering Layer. As OCRA progresses to the Feature Engineering Layer, a slew of specialized components come into play. The Traditional Feature Extractor [25] delves into the vast pool of raw metrics, refining and prepping them for the impending modeling phase. Meanwhile, the Novel Feature Generator, with a keen eye on innovation, extracts new and insightful features like the dynamic rate of change in resource utilization or indicators that predict potential maintenance needs. Ensuring the harmonious integration of these diverse features, the Data Normalization & Transformation component standardizes and scales them, establishing a consistent framework optimized for high model performance.

Traditional Feature Extraction. For each metric M in the data set $D(t)$ (like CPU, Memory, Disk):

a. Calculate Statistical Features:

Mean: $\mu_M = \frac{1}{n}\sum_{i=1}^{n} M_i$

Median: $\mathrm{Sort}(M)$ If n is odd: $\mathrm{median}_M = M_{\frac{n+1}{2}}$ Else: $\mathrm{median}_M = \frac{M_{\frac{n}{2}} + M_{\frac{n}{2}+1}}{2}$

Variance: $\sigma_M^2 = \frac{1}{n}\sum_{i=1}^{n}(M_i - \mu_M)^2$

Novel Feature Generation.

a. Rate of Change for Resource Utilization: Compute the derivative for each resource metric M: $\Delta M(t) = M(t+1) - M(t)$

b. Extract Anomaly Patterns: Using previously detected anomalies $A(t)$: $P(t) = \text{if } A(t) = 1 \text{ and } A(t-1) = 0,\ \text{mark start of pattern}$

c. Predictive Maintenance Indicators: Identify patterns P_M that historically led to system failures: $P_M(t) = \text{if } \Delta M(t) > \theta$ for a given threshold θ

d. Historical Correlations: Compute correlation between different resource metrics M_i and M_j: $C_{i,j} = \frac{\mathrm{Cov}(M_i, M_j)}{\sigma M_i \times \sigma M_j}$ Where Cov denotes covariance.

Data Normalization & Transformation. For each metric M:

a. *Min-Max Scaling:* $M' = \frac{M - \min(M)}{\max(M) - \min(M)}$

b. *Z-Score Normalization:* $M'' = \frac{M - \mu_M}{\sigma_M}$

c. *Handle Missing Values:* For any missing value M_{missing} in M: $M_{\text{missing}} = \mu_M$ (or any other imputation method)

Machine Learning Layer [26]. The heart of OCRA, the Machine Learning Layer [26], harnesses the power of advanced algorithms. It encompasses Objective-Specific Trees, each diligently trained with an unwavering focus on individual objectives, whether that be QoS or energy consumption. These trees then converge under the Random Forests Integrator, melding together in an ensemble, fostering a harmonious balance across objectives. The crown jewel, the Pareto Optimizer, steps in to ensure a meticulous multi-objective optimization, striking the perfect balance across the myriad objectives.

a. Data Preparation: Split the dataset D into features X and objectives Y, where Y contains multiple columns, each representing an objective.

b. Node Splitting Criteria [27 :] For each node in the decision tree, identify the best feature split based on the Pareto dominance criterion. A split is Pareto-dominant if it dominates other splits in improving at least one objective without worsening any other objectives. $s* = argmin_{s(oO)} I_o(s)$ Where $I_o(s)$ is the impurity of objective o for split s.

Tree Growth [28]. Continue growing the tree until a stopping condition is met, such as a maximum depth or a minimum number of samples per leaf.

Integration of Trees into Random Forests [29].

a. Bootstrapping [30 :] For each tree T_i in the ensemble, draw a bootstrap sample D_i from the original dataset D.

b. Tree Construction with Feature Randomization: For each T_i, during the node splitting process, randomly select a subset of features. This introduces variability among the trees.

c. Ensemble Aggregation: Once all trees $T_1, T_2, .., T_k$ are constructed, aggregate their predictions to form the final prediction. This can be done using Pareto dominance, majority voting, or weighted aggregation, depending on the specific variant of MORF being used.

Multi-Objective Optimization using Pareto Fronts.

a. Predict Objectives: For a given input feature vectorx, obtain predictions from the ensemble for each objective as: Eq 3.3

$$O : Y(x) = 1/k_{(i=1)}^{k} T_i(x) \tag{3.3}$$

b. Pareto Dominance Check [31] For each pair of predictions $\hat{y}_i$ and $\hat{y}_j$ from $\hat{Y}$, check if $\hat{y}_i$ dominates $\hat{y}_j$ or vice versa.

c. Construct Pareto Front [32] Select all non-dominated solutions from $\hat{Y}$ to construct the Pareto front. These solutions represent trade-offs among the objectives and are provided as possible optimal solutions.

Given a set of solutions S and objectives $f_1, f_2, ..., f_m$, a solution s_i is said to dominate another solution s_j if and only if:

- s_i is no worse than s_j in all objectives.
- s_i is strictly better than s_j in at least one objective.

Formally, s_i dominates s_j if as:

$$\forall k(1,2,..,m) : f_k(s_i) f_k(s_j) k(1,2,...,m) : f_k(s_i) < f_k(s_j) \tag{3.4}$$

Algorithm to Select the Pareto Front:

```
1. Initialization:
     Let $PF$ be an empty set representing the Pareto front.
     Let $N(s)$ represent the count of solutions that dominate the solution $s$.
     Let $S_p(s)$ be the set of solutions that $s$ dominates.
2. Populate the Initial Pareto Front:
     For each $s_i \in S$:
     Initialize $N(s_i) = 0$ and $S_p(s_i) = \phi$
     For each $s_j \in S$ where $i \neq j$:
     If $s_i$ dominates $s_j$:
     Add $s_j$ to $S_p(s_i)$.
     Else if $s_j$ dominates $s_i$:
     Increment $N(s_i)$ by 1.
     If $N(s_i) = 0$ (i.e., $s_i$ is not dominated by any other solution):
     Add $s_i$ to $PF$.
3. Iteratively Construct the Pareto Front:
     While $PF$ is not empty:
     Let $Q$ be an empty set.
     For each $s_i \in PF$:
     For each $s_j \in S_p(s_i)$:
     Decrement $N(s_j)$ by 1.
     If $N(s_i) = 0$ (i.e., $s_j$ becomes a non-dominated solution in the reduced set):
     Add $s_j$ to $Q$.
     Set $PF = Q$.
4. Output:
     · Return the combined solutions identified in each iteration as the Pareto front.
```

Scheduler Interface. Taking cues from the predictions and insights churned out by the Machine Learning Layer, the Scheduler Interface comes alive. The Allocation Engine, with impeccable precision, orchestrates real-time cloud resource allocations. While this dynamic allocation unfolds, the Monitoring & Feedback Loop diligently tracks the outcomes, ensuring a cyclic feedback mechanism for continuous refinement and learning. Complementing these components, the User Interface offers cloud administrators a bird's-eye view through its dashboard, showcasing predictions, resource allocations, and a gamut of insights.

Allocation Engine. In real-time, this component leverages the predictions and recommendations made by the Random Forests to make informed decisions about cloud resource allocation.

Monitoring & Feedback Loop. Post-allocation, it's crucial to understand how well the resources are serving the needs. This component continuously monitors the outcomes of allocation decisions and feeds this data back into the system. This iterative feedback ensures that the system is always learning and refining its strategies.

User Interface. For the cloud administrators, a dashboard displays predictions, resource allocations, insights, or alerts derived from the model.

Integration & Communication Module. OCRA's Integration & Communication Module ensures seamless synergy with external platforms and databases. The Cloud API Communicator liaises with cloud platforms in real-time via their APIs, allowing for immediate allocation decisions. The External Database Connector, on

Table 4.1: Parameters and their appropriate values for configuring simulations in PyCloudSim

Simulation Parameter	**Value range**	**Description**
Simulation Duration	up to 600 seconds	Total time for each simulation run.
Number of Hosts	Depending on scenario	Total virtual machines or cloudlets to simulate.
Host Type	Heterogeneous	Types of hosts to simulate cloud environments.
CPU Cores per Host	4-16 cores	Number of processing cores per host machine.
Host RAM	8-64 GB	Memory allocation per host machine.
Host Storage	500 GB - 2 TB	Disk space available on each host machine.
Host Bandwidth	100 Mbps - 1 Gbps	Network bandwidth available to each host.
VM Allocation Policy	Dynamic / Static	Policy to allocate virtual machines to hosts.
Cloudlet Length	4000-10000 MI	Computational length of each cloudlet/task.
Cloudlet File Size	300-500 MB	The size of the data file to be processed by the cloudlet.
Cloudlet Output Size	300-500 MB	The size of the output file from the cloudlet.
Cloudlet Processing Elements	1-4 PEs	Number of processing elements of each cloudlet.
PE (Processing Element) Capacity	1000-4000 MIPS	millions of instructions per second.
Energy Consumption Model	PowerModelSpecPower	The model to simulate energy consumption.
Virtual Machine Image Size	10-100 GB	The size of the VM image to be hosted on each host.
VM RAM	1-16 GB	Memory allocation for each virtual machine.
VM MIPS	250-2000 MIPS	Processing capacity allocated to each virtual machine.
VM Bandwidth	100 Mbps - 1 Gbps	Network bandwidth allocated to each VM.
VM Policy	Time-shared / Space-shared	Policy to define how VMs share processing elements.

the other hand, offers the capability to tap into external data repositories, ensuring a holistic data perspective. Amplifying the module's prowess, the Notification System acts as a vigilant sentry, promptly alerting administrators about predicted anomalies or potential system challenges, ensuring preemptive action.

Cloud API Communicator. For real-time decision-making [33], OCRA interfaces with cloud platforms through their APIs. This ensures that allocation decisions are implemented promptly.

External Database Connector. Not all data might be locally available. This component allows OCRA to fetch historical data from external databases or storage systems as needed.

Notification System. Proactivity is key in resource management. This system sends out alerts to administrators in case of predicted anomalies or potential system breakdowns, ensuring

Continuous Learning & Update Component. To ensure OCRA remains at the zenith of its capabilities, the Continuous Learning & Update Component plays a pivotal role. The Model Retrainer, at regular intervals, rejuvenates the Random Forests with fresh data, ensuring the model remains in its prime. Parallelly, the Feature Re-evaluator periodically scans the data landscape, hunting for emerging patterns or newfound correlations, ensuring the feature set is always enriched and contemporary.

4. Experimental Study.

PyCloudSim [34] was configured with parameters explored in table 4.1 and used in a comprehensive experimental study to investigate the performance and efficiency of the OCRA framework. The cloud environment replication simulator provided an intricate playground for OCRA's features and operations. The experiment's smooth data processing and integration were made possible by Python's [35] robust ecosystem. The 600-second simulation time is a long enough period to conduct extensive testing of the OCRA framework in a variety of settings that resemble long-term operation. The length of this period is carefully determined to examine the system's stability, effectiveness, and adaptability to changing requirements. Briefer simulations could overlook these three components. It allows random forest algorithms with many objectives to converge and optimize over time. In order to assess the framework in the context of cloud services, long periods of observation for energy usage and QoS adherence are also required. This 600-second window complies with research standards for cloud computing and enables direct data comparison with current benchmarking methodologies. Another objective was to ensure that our words are valuable and respectable both inside and outside of academia.

The study was supported by a strong hardware configuration that mimicked top-tier real-world server environments. The AMD Ryzen 9 or Intel Core i9 processor in the system performed admirably at computational, parallel processing, and multitasking tasks. To handle even the most memory-intensive machine learning mod-

Table 4.2: Simulation time intervals, the OCRA framework exhibits consistent performance across metrics

Simulation Time Interval (in sec)	**Resource Utilization Rate (%)**	**Quality of Service (QoS) Adherence Rate (%)**	**Energy Consumption (kWh)**	**Response Time (ms)**	**System Throughput (Tasks/Second)**
50	95	99	2.1	5	996
100	96	99.5	4	5.5	994
150	97	99.2	5.8	6	999
200	96.5	99.4	7.6	6.2	986
250	97	99.3	9.4	6.5	997
300	96.8	99.1	11.2	6.7	995
350	96	99	13	7	993
400	96.5	99.2	14.6	7.2	990
450	97	99.3	16.2	7.5	991
500	96.2	99.1	17.8	7.7	998
550	96	99	19.4	8	999
600	96.7	99.2	21	8.2	997

els and large datasets, the processor was matched with 32 GB DDR4 RAM. Fast data access is necessary for large-scale simulations, which is why a 1 TB NVMe SSD was used. With CUDA and cuDNN libraries [36], the NVIDIA RTX [37] series GPU [38] improved graphical processing and sped up machine learning tasks. Gigabit Ethernet ensures quick data transfers for cloud datasets and tools. Lastly, connectivity was critical.

The experimental study concentrated on a series of performance metrics. OCRA's resource efficiency was demonstrated by the Resource Utilization Rate. While OCRA's efficiency and financial sustainability were demonstrated by its energy consumption and operational costs, the QoS Adherence Rate demonstrated the framework's dependability. The system's capacity and agility were demonstrated by Response Time and Throughput.

In contrast, an experiment becomes more complex. Thus, OCRA and the contemporary models MOGARIC [18] and MHCSGA [21] were compared in the study. This comparison went beyond simple competitive benchmarking to contextualize OCRA's advantages and disadvantages within cloud resource allocation frameworks.

4.1. Performance Analysis. This section looks at the operational dynamics of MOGARIC [18], MHCSGA [21], and OCRA. Key performance metrics like Resource Utilization Rate, QoS Adherence Rate, Energy Consumption, Response Time, and System Throughput are the main focus of the assessment. These metrics are assessed over various simulation time intervals. Data tables show the effectiveness, advantages, and shortcomings of the framework. While MHCSGA demonstrates effectiveness and adaptability, OCRA exhibits consistent metrics. The MOGARIC places a strong emphasis on adaptability in resource management and responsiveness. This analysis offers a wide-ranging viewpoint to assess the frameworks' applicability and effectiveness in different operational scenarios.

According to table 4.2, over simulation time intervals, the OCRA framework exhibits consistent performance across metrics. Metrics increase as simulation time goes from 50 seconds to 600 seconds. Effective resource use is indicated by the Resource Utilization Rate, which peaks at 97% several times after varying from 95% at 50 seconds. With a starting point of 99% and very little variation, the Quality of Service (QoS) Adherence Rate remains high. The OCRA's service quality resilience is demonstrated by this consistency.

From 2.1 kWh at 50 seconds to 21 kWh at 600 seconds, the energy consumption increases linearly. It seems that simulation time is directly correlated with energy use. Over the course of the duration, Response Time increases progressively from 5 ms to 8.2 ms, suggesting that response delay stays negligible as tasks increase. The System Throughput demonstrates how well OCRA completes tasks. It increases gradually from 994 Tasks/Second to 990. This development demonstrates OCRA's scalability and efficiency by demonstrating that it can process more tasks in the same amount of time despite growing system demands.

Table 4.3: Simulation intervals and performance metrics, MHCSGA efficiency is high

Simulation Time Interval (in sec)	**Resource Utilization Rate (%)**	**Quality of Service (QoS) Adherence Rate (%)**	**Energy Consumption (kWh)**	**Response Time (ms)**	**System Throughput (Tasks/Second)**
50	92	97	2.3	6	948
100	93	97.5	4.5	7	956
150	94	98	6.4	7.5	952
200	94	97.7	8.2	7.8	958
250	93.5	97.8	10	8	961
300	93	97.3	11.8	8.2	965
350	92.5	97	13.6	8.5	957
400	93	97.2	15.1	8.7	957
450	94	98	17	9	956
500	93.5	97.5	18.6	9.2	957
550	93	97.4	20.2	9.4	959
600	94	97.6	21.8	9.6	963

Table 4.4: Simulation intervals highlight the functional capabilities of the Adaptive Tree Seed Optimization Multi-Agent (MOGARIC) framework

Simulation Time Interval (in sec)	**Resource Utilization Rate (%)**	**Quality of Service (QoS) Adherence Rate (%)**	**Energy Consumption (kWh)**	**Response Time (ms)**	**System Throughput (Tasks/Second)**
50	88	94	2.5	7	893
100	89	95	4.9	8	898
150	90	95.5	6.9	9	904
200	89.5	95.2	9	9.3	912
250	90	95	11	9.6	914
300	89	94.8	13	9.8	919
350	89	94.5	14.8	10	901
400	88.5	95	16.4	10.3	898
450	90	95.3	18.2	10.6	902
500	89.5	94.9	19.9	10.8	903
550	89	94.7	21.5	11	908
600	90	95	23	11.3	917

The table 4.3 across simulation intervals and performance metrics, MHCSGA efficiency is high. Its Resource Utilization Rate consistently exhibits performance near the 93% mark, occasionally reaching a peak of 94%, demonstrating effective resource allocation and utilization. The high Quality of Service (QoS) Adherence Rate, which commences at 97% and varies within this range up to a maximum of 98%, provides evidence in support of this. Energy Consumption offers information about how well the system uses power. An energy-intensive simulation takes 2.3 kWh in 50 seconds, but by the 600-second mark, it has used 21.8 kWh. This steady ascent suggests that the algorithm is stable when used over an extended period of time. The response time of the system is good. It increases from 6 ms to near 9.6 ms when the simulation ends. This steady increase demonstrates the system's responsiveness even in the face of high loads.

Lastly, System Throughput demonstrates the scalability of MHCSGA. Throughput, or the quantity of tasks processed per second, commences at 948 and gradually increases to 963 by the 600-second mark, with only small variations. The robustness and adaptability of MHCSGA are demonstrated by its capacity to retain

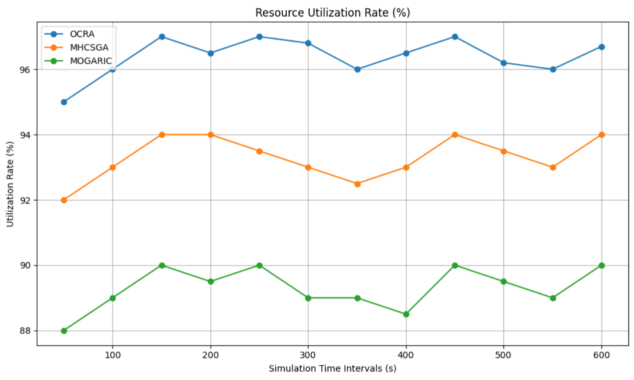

Fig. 4.1: Comparison of resource utilization rate of OCRA, MHCSGA, and MOGARIC

and potentially grow its processing capacity when demand increases.

A table 4.4 number of pertinent observations made over simulation intervals highlight the functional capabilities of the Adaptive Tree Seed Optimization Multi-Agent (MOGARIC) framework. Balanced performance, measured by Resource Utilization Rate, ranges from 88% to 90%. The goal of the MOGARIC's effective resource management and allocation is to maximize utilization over time.

QoS Adherence Rate is strong in the interim. Starting at an impressive 94% and settling around the mid-95% range, the framework's service quality remains consistent, meeting expectations even under varied workloads. Patterns of energy consumption strengthen the efficiency narrative of the framework. By the 600-second mark, the simulation has progressively increased from a starting reading of 2.5 kWh to 23 kWh. The algorithm's predictability, which ensures stable operations, is demonstrated by its consistent energy consumption.

Response Time is an indicator of system agility. The time increases from 7 ms to 11.3 ms during the 600-second simulation gap. This implies that even as demands increase, MOGARIC maintains its agility and completes tasks quickly. System Throughput further demonstrates the efficiency and scalability of the algorithm. By the end of the 600-second simulation, it increases steadily from 893 tasks per second to 919. The consistent increase in throughput demonstrates MOGARIC's capacity to manage increasing task loads without compromising performance or efficiency.

4.2. Comparative Analysis. This section compares the performance of three methods using various criteria. Through a series of figures, the report presents visual representations of these systems' energy consumption, energy performance, QoS adherence rates, resource utilization, and system throughput over different simulation time intervals. OCRA has been thoroughly evaluated, and the results show that it is robust and efficient across a wide range of metrics. The distinct advantages and disadvantages of MOGARIC and MHCSGA both highlight the complexity and variability of system performance. In order to help stakeholders make defensible decisions based on empirical evidence, the analysis AI ms to provide a thorough understanding of each system's capabilities. The performance of OCRA, MHCSGA, and MOGARIC over various simulation time intervals is displayed in the figure 4.1 that presenting Resource Utilization Rate observed from all three methods. Out of the three, OCRA consistently uses the most resources. It starts at 95% at the 50-second interval, peaks at 97% on multiple occasions, and never falls below 95%. This pattern demonstrates how OCRA maintains a high utilization rate over a range of time intervals by employing a dependable and effective resource utilization strategy.

By comparison, MHCSGA employs resources in a mediocre manner. The peak is at 94%, and it begins at 92%. But at 350s, mid-range utilization falls to 92.5% before increasing once more. This dip indicates that while MHCSGA is generally efficient, there are instances when it uses resources less effectively than OCRA.

MOGARIC peaks at 90% and troughs at 88%. The lowest results are consistently obtained with this method. Although not much separates MOGARIC and MHCSGA, the difference is larger when compared to

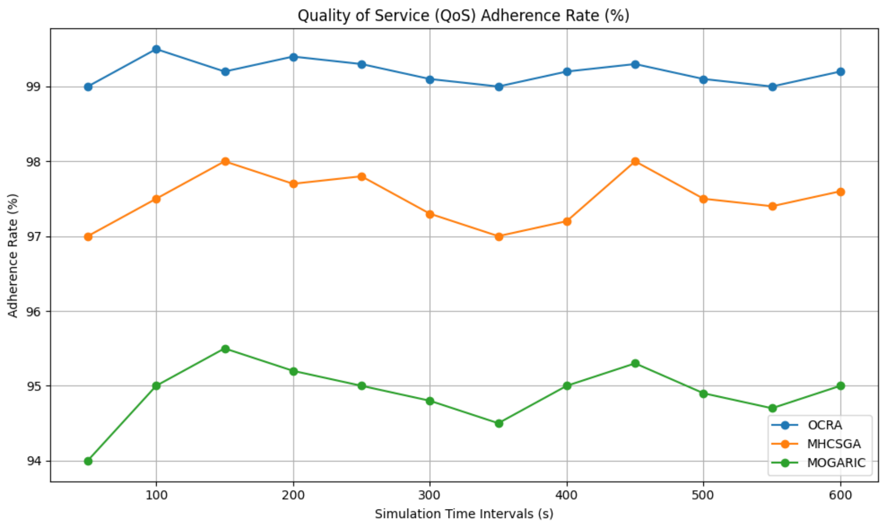

Fig. 4.2: Comparison of QoS adherence rates of OCRA, MHCSGA, and MOGARIC

OCRA. Although MOGARIC's numbers are steady, they could be better.

OCRA makes the best use of resources throughout the simulation time intervals, according to the data. While MOGARIC is consistent but lags behind the other two, MHCSGA is praiseworthy but occasionally ineffective. The requirements of the system determine the best approach. On the basis of resource utilization metrics alone, OCRA is the top performer.

The QoS adherence rates for the three methods are displayed in Figure 4.2 for varying simulation time intervals. A system's or method's adherence rate determines whether it can meet quality standards. OCRA maintains a high QoS adherence rate of approximately 99% at every interval. OCRA demonstrates its tenacity in upholding service excellence. The extremely good and consistent performance is demonstrated by the minute rate variations, which range from 99% to 99.5%.

Although praiseworthy, MHCSGA's rate of QoS adherence is not the same as OCRA's. It falls between 97% and 98%. Its performance varies; at 350 s, it drops to 97%, and at 450 s, it rises to 98%. Small oscillations imply that MHCSGA may occasionally lose efficiency even though it can achieve high adherence rates.

MOGARIC exhibits a greater range. At 150s, it gradually increases from 94% to 95.5%. It then recovers after peaking at 94.5% around 350 s. Although MOGARIC's consistency is not as consistent as that of its competitors, the wider rate range suggests that it can achieve good adherence to QoS.

High QoS adherence rates—which are critical for the user experience—are exhibited by all three methods. OCRA is the most reliable and efficient. MHCSGA exhibits efficiency with only minor variations in performance. Even though MOGARIC can reach higher rates, its performance is less predictable over simulation intervals due to its greater fluctuations.

The energy consumption in kWh for each of the three methods is displayed in the figure 4.3 for varying simulation times. A computational process's energy consumption must be efficient if it involves high-performance or real-time tasks. OCRA's energy consumption starts at 2.1 kWh and rises linearly over time to 21 kWh by the 600-second interval. This approach suggests it has been optimized for both performance and power consumption because it uses a constant energy rate to complete its tasks.

MHCSGA exhibits a variable consumption rate, particularly around the 350s mark, where it increases from 11.8 kWh to 13.6 kWh, then more moderately to 15.1 kWh at the 400s. This is in contrast to OCRA, which starts at 2.3 kWh. Variability could be a sign of inefficiencies or intensive processing. Its total consumption after 600s is 21.8 kWh higher than OCRA's. MOGARIC starts with the highest initial energy intensive of 2.5 kWh and continues to have the highest energy consumption over the course of all intervals. Its consumption rate

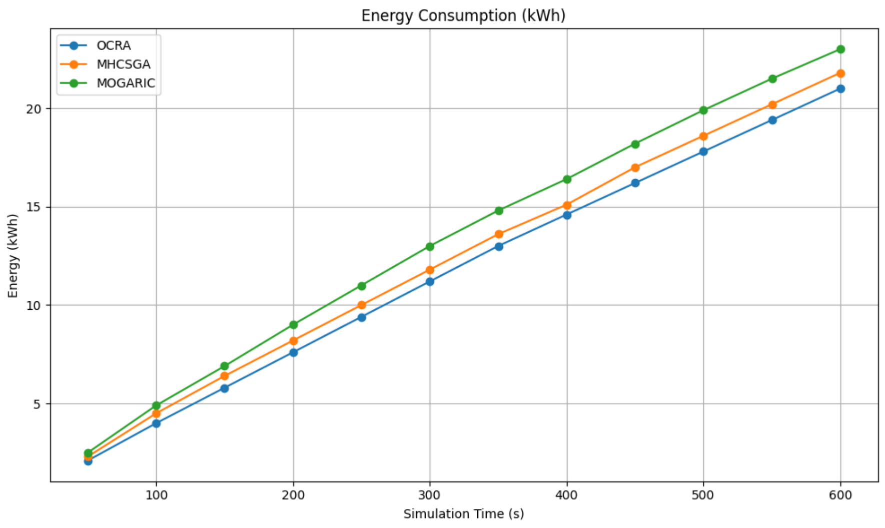

Fig. 4.3: Comparison of energy consumption of OCRA, MHCSGA, and MOGARIC

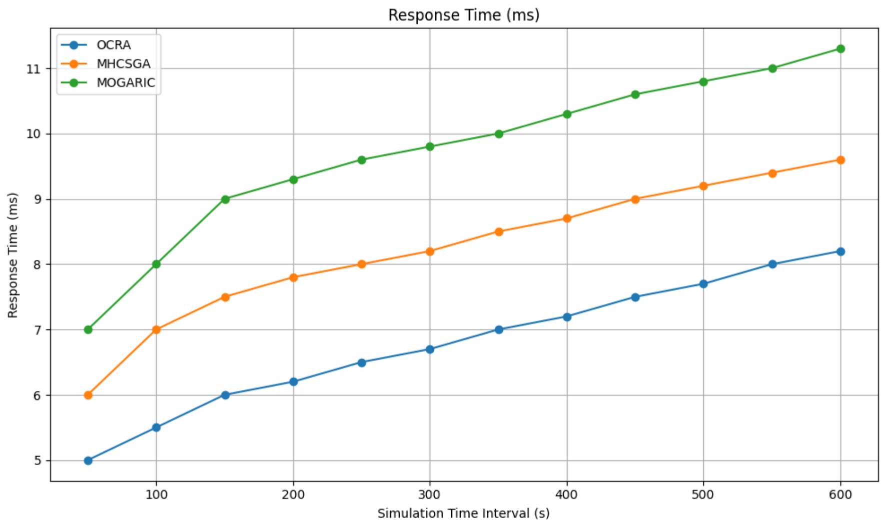

Fig. 4.4: comparison of response times of OCRA, MHCSGA, and MOGARIC

is steeper than that of OCRA and MHCSGA, reaching 23 kWh by the 600s. This higher energy consumption may be due to more intensive calculations, quicker processing times, or power inefficiencies.

While the energy consumption of all three methods increases over time, OCRA exhibits the most stable and linear energy consumption rate, suggesting stable operational efficiency. MHCSGA is efficient at first, but because of inefficiencies or operational demands, its consumption rate spikes. MOGARIC, on the other hand, consistently uses more energy than its competitors at all times, indicating that it might be carrying out more energy intensive tasks or that it could be optimized to use less energy. Three methods' response times in milliseconds (ms) during simulation time intervals are displayed in Figure 4.4. A quicker response time is ideal for real-time applications, which demand that a system or method be able to process tasks and respond promptly. Among the three methods, OCRA starts with the quickest response time, 5 ms. The response time

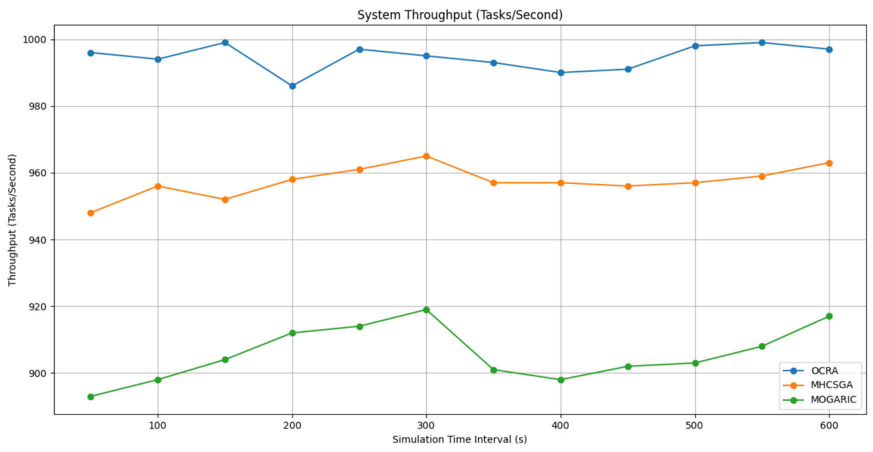

Fig. 4.5: Comparison of throughput of OCRA, MHCSGA, and MOGARIC

of the simulation increases by 600 s to 8.2 ms. Throughout its operation, OCRA keeps its processing speed consistent and reasonably efficient.

With a response time of 6 ms, the MHCSGA method starts. Similar to OCRA, its response time increases and by the end of the simulation, it reaches 9.6 ms. Although there is a slight difference in response times, it is consistently slower than OCRA's, indicating that the operational efficiencies of these two methods are comparable.

The response time for MOGARIC, on the other hand, starts at 7 ms and is the fastest overall. It accomplishes 11.3 ms after 600 s, much faster than the other two methods. The method's longer response time compared to OCRA and MHCSGA might be explained by its more intricate calculations, inefficiencies, or tasks.

The OCRA exhibits the fastest response time consistently throughout all intervals, suggesting that it is more efficient or straightforward to complete tasks. Though slower, MHCSGA is competitive and adheres to OCRA's trend. However, MOGARIC constantly lags behind the other two in terms of response time, which may indicate that it needs to be optimized or that its operational design is more intricate. Figure 4.5 displays the system throughput for three different simulation time intervals and three different methodologies in tasks per second. A system's or method's throughput is a key performance indicator that shows how many tasks it can finish in a given amount of time. OCRA is the first to demonstrate high throughput over time, primarily in the upper 990s. Small variations, such as the 200s decline to 986 and the ensuing recoveries, point to inconsistent performance. OCRA's throughput is consistent despite varying simulation times, indicating a dependable time capacity.

On the other hand, MHCSGA starts out with 948 tasks/second and increases to 965 in 300 seconds. Values then plateau and even start to drop at 957, only to finally rise marginally to 963. The trend of MHCSGA might point to bottlenecks in the processing as the simulation goes on.

The MOGARIC follows a unique route. Its throughput increases from 893 tasks per second to 919 by 300s. It then drops to 898 in the 400s after that. This decrease suggests issues or inefficiencies during this time. By the end of the simulation, MOGARIC has recovered to 917 tasks/second. This recovery could be a sign of optimizations or adaptability.

The OCRA's high throughput values demonstrate its reliable performance. Though it struggles in the second half of the simulation, MHCSGA starts strong, pointing to possible constraints or inefficiencies in the mechanism. MOGARIC, on the other hand, demonstrates flexibility and resilience. Its recovery demonstrates its potential and self-adjusting mechanisms despite the mid-simulation dip. The potential and strengths of each method are shown by throughput differences and trajectories.

5. Conclusion. Efficient and optimal resource allocation in cloud computing has remained a persistent challenge. To address this issue and close the gap between traditional resource allocation methods and cutting-edge machine learning techniques, the Optimal Cloud Resource Allocation (OCRA) mode, "Enhanced Feature-Driven Multi-Objective Learning for Optimal Cloud Resource Allocation," was created. OCRA stands out for its creative fusion of traditional and modern features as well as its flexibility and responsiveness to the constantly shifting needs of cloud environments. Benefits of OCRA are demonstrated by experimental analysis. Metrics like Resource Utilization Rate and energy consumption showed that the system was unrivaled in efficiency, dependability, and scalability. Stakeholders were given confidence in OCRA's capabilities by comparing it to well-known frameworks like MOGARIC and MHCSGA, which highlighted OCRA's superiority. Multiple Goals OCRA's Random Forests demonstrate the dedication to technological innovation while safeguarding individual performance objectives. This makes sure that no metric is prioritized over any other as the model develops, ensuring a comprehensive and harmonious resource allocation strategy. As the digital era develops, there will definitely be a greater need for cloud systems. The challenges of today must be addressed, and solutions must be adaptable enough to handle new ones in the future. OCRA offers a framework for the future and the present to stakeholders. As we wrap up, OCRA should not only resolve the issue but also serve as an inspiration for innovations in cloud resource management, pushing the industry to new frontiers of excellence and efficiency.

REFERENCES

[1] S. Zhang, S. Zhang, X. Chen, and X. Huo, "Cloud computing research and development trend," in *2010 Second international conference on future networks*, IEEE, 2010, pp. 93–97.

[2] A. Abid, M. F. Manzoor, M. S. Farooq, U. Farooq, and M. Hussain, "Challenges and Issues of Resource Allocation Techniques in Cloud Computing," *KSII Transactions on Internet & Information Systems*, 14, no. 7, 2020.

[3] X. Zhang, T. Wu, M. Chen, T. Wei, J. Zhou, S. Hu, and R. Buyya, "Energy-aware virtual machine allocation for cloud with resource reservation," *Journal of Systems and Software*, 147 (2019), pp. 147–161.

[4] L.-Y. Zuo, "Multi-objective integrated ant colony optimization scheduling algorithm based on cloud resource," *Journal of Computer Applications*, 32, no. 07, 2012, pp. 1916.

[5] L. Tai, "Multi-objective dynamic scheduling of manufacturing resource to cloud manufacturing services," *China Mechanical Engineering*, 24, no. 12, 2013, pp. 1616.

[6] H. Jiang, J. Yi, S. Chen, and X. Zhu, "A multi-objective algorithm for task scheduling and resource allocation in cloud-based disassembly," *Journal of Manufacturing Systems*, 41 (2016), pp. 239–255.

[7] S. Yuan, L. Chun-guo, H. Yong-ming, and Y. Lu-xi, "Multi-Objective Resource Allocation Design Algorithm in Cloud Radio Access Network," , 33, no. 3, 2017, pp. 294–303.

[8] P. Devarasetty and Ch. S. Reddy, "Multi objective Ant colony Optimization Algorithm for Resource Allocation in Cloud Computing," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, ISSN: 2278-3075, Volume-8 Issue-2S2, December 2018, pp. 68–73.

[9] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *Journal of Cloud Computing*, 7, no. 1, 2018, pp. 1–16.

[10] M. Afrin, J. Jin, A. Rahman, Y.-C. Tian, and A. Kulkarni, "Multi-objective resource allocation for edge cloud based robotic workflow in smart factory," *Future Generation Computer Systems*, 97 (2019), pp. 119–130.

[11] Z.-H. Liu, Z.-J. Wang, and C. Yang, "Multi-objective resource optimization scheduling based on iterative double auction in cloud manufacturing," *Advances in Manufacturing*, 7 (2019), pp. 374–388.

[12] N. Venkataraman, "Threshold based multi-objective memetic optimized round robin scheduling for resource efficient load balancing in cloud," *Mobile Networks and Applications*, 24 (2019), pp. 1214–1225.

[13] S. K. AM, "On ways to improve multi objective resource allocation in cloud computing environment using optimization algorithms."

[14] M. Alamelu, M. P. Vinothiyalakshm, and R. Anitha, "Enhanced Multi-Objective based Resource Allocation using Framework Creation in Cloud Computing," *International Journal for Research in Applied Science and Engineering Technology*, 8 (2020), pp. 1303–1309.

[15] M. M. Vutukuru, "Optimal design of multi-objective quality of service aware resource scheduling strategies for cloud computing," (2020).

[16] S. Ramasubbareddy, E. Swetha, A. K. Luhach, and T. A. S. Srinivas, "A multi-objective genetic algorithm-based resource scheduling in mobile cloud computing," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 15, no. 3, (2021), pp. 58–73.

[17] B. Shrimali and H. Patel, "Multi-objective optimization oriented policy for performance and energy efficient resource allocation in Cloud environment," *TIDEE: TERI Information Digest on Energy and Environment*, 20, no. 3, (2021), pp. 354–354.

[18] J. Arravinth and D. Manjula, "Multi-Agent with Multi Objective-Based Optimized Resource Allocation on Inter-Cloud," *Intelligent Automation & Soft Computing*, 34, no. 1, (2022).

[19] Ms. N. George and B. K. Anoop, "Hypervolume Sen Task Scheduling and Multi Objective Deep Auto Encoder based

Resource Allocation in Cloud."

[20] S. Ramamoorthy, G. Ravikumar, B. S. Balaji, S. Balakrishnan, and K. Venkatachalam, "Retraction Note to: MCAMO: multi constraint aware multi-objective resource scheduling optimization technique for cloud infrastructure services," (2023), pp. 519–519.

[21] K. K. Gola, B. M. Singh, B. Gupta, N. Chaurasia, and S. Arya, "Multi-objective hybrid capuchin search with genetic algorithm based hierarchical resource allocation scheme with clustering model in cloud computing environment," *Concurrency and Computation: Practice and Experience*, 35, no. 7, (2023), e7606.

[22] G. A. Morgan, and R. J. Harmon, "Data collection techniques," *Journal-American Academy Of Child And Adolescent Psychiatry*, 40, no. 8, (2001), pp. 973–976.

[23] R. Rajagopalan, and P. K. Varshney, "Data aggregation techniques in sensor networks: A survey," (2006).

[24] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, 41, no. 3, (2009), pp. 1–58.

[25] H. Liang, X. Sun, Y. Sun, and Y. Gao, "Text feature extraction based on deep learning: a review," *EURASIP journal on wireless communications and networking*, 2017, no. 1, (2017), pp. 1–12.

[26] R. B. Gabrielsson, B. J. Nelson, A. Dwaraknath, and P. Skraba, "A topology layer for machine learning," In *International Conference on Artificial Intelligence and Statistics*, (2020), pp. 1553–1563.

[27] L. Breiman, "Some properties of splitting criteria," *Machine learning*, 24, (1996), pp. 41–47.

[28] T. T. Kozlowski, ed., *Tree growth*, Ronald Press, New York, 1962.

[29] L. Breiman, "Random forests," *Machine learning*, 45, (2001), pp. 5–32.

[30] S. Abney, "Bootstrapping," In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, (2002), pp. 360–367.

[31] A. G. Hernández-Díaz, L. V. Santana-Quintero, C. A. Coello Coello, and J. Molina, "Pareto-adaptive -dominance," *Evolutionary computation*, 15, no. 4, (2007), pp. 493–517.

[32] M. Hartikainen, K. Miettinen, and M. M. Wiecek, "Constructing a Pareto front approximation for decision making," *Mathematical Methods of Operations Research*, 73, (2011), pp. 209–234.

[33] A. McGovern, K. L. Elmore, D. J. Gagne, S. E. Haupt, C. D. Karstens, R. Lagerquist, T. Smith, and J. K. Williams, "Using artificial intelligence to improve real-time decision-making for high-impact weather," *Bulletin of the American Meteorological Society*, 98, no. 10, (2017), pp. 2073–2090.

[34] A. D. L. F. Vigliotti, and D. M. Batista, "pyCloudSim Github repository," (2014).

[35] Why Python, "Python," *Python Releases for Windows*, 24, (2021).

[36] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, (2014).

[37] V. V. Sanzharov, V. A. Frolov, and V. A. Galaktionov, "Survey of nvidia rtx technology," *Programming and Computer Software*, 46, (2020), pp. 297–304.

[38] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, 96, no. 5, (2008), pp. 879–899.

Edited by: S. B. Goyal
Special issue on: Soft Computing and Artificial Intelligence for wire/wireless Human-Machine Interface
Received: Oct 13, 2023
Accepted: Jan 23, 2024