# Optimized Task Scheduling and VM Allocation in Cloud Computing Using PPMMcNE and RSMBO Algorithms

Nisha Sanjay[1], Sasikumaran Sreedharan[2]
[1]Research Scholar, Lincoln University College, Malaysia
[2]Research Supervisor, LUC MRC, Marian College Kuttikkanam, Kerala, India
E-mail: nsanjay@lincoln.edu.my, drsasikumaran@gmail.com

**Keywords:** cloud computing, task scheduling, virtual machine allocation, PPMMcNE (phasmatodea population modified mcnaughton evolution), RSMBO (rat swarm modified brucker optimization), makespan minimization, modified McNaughton's rule, turnaround time and response time

**Received:** January 6, 2025

*This paper presents an optimized approach for task scheduling and virtual machine (VM) allocation in cloud computing environments, leveraging two novel algorithms. The proposed Phasmatodea Population Modified McNaughton Evolution (PPMMcNE) algorithm enhances the Phasmatodea Population Evolution (PPE) method by integrating Modified McNaughton's rule to generate a high-quality initial task schedule and minimize delays. Complementarily, the Rat Swarm Modified Brucker Optimization (RSMBO) algorithm is introduced to refine VM allocation by reducing migration overhead and lowering energy consumption. The methods aim to optimize key cloud performance parameters—including turnaround time, waiting time, completion time, response time, makespan, cost, load balancing, and energy efficiency—thereby enhancing overall resource utilization and fairness. Comprehensive computational experiments were performed in Matlab using the publicly accessible GoCJ dataset, which comprises one month of resource utilization data, recording 123 million incidents across 1250 computers. The proposed method achieves a throughput of 0.942, exhibits a minimal task scheduling delay of 58.22 milliseconds, and maintains a queue waiting time of 43.66 milliseconds—all while reducing energy consumption to an average of 120 joules per task. Furthermore, energy consumption was quantitatively evaluated, with RSMBO consistently demonstrating significant reductions in energy usage compared to traditional baselines. These results validate that the integrated approach of PPMcNE and RSMBO offers superior scalability and efficiency, making it highly suitable for dynamic and large-scale cloud environments.*

*Povzetek: Predstavljena je dvojna optimizacijska metoda za razporejanje opravil in dodeljevanje VM v oblaku, ki združuje algoritme PPMMcNE in RSMBO z biološko navdihnjenimi metahevristikami.*

## 1 Introduction

Offering on-demand access to a shared pool of reconfigurable computing resources—including networks, servers, storage, and applications—cloud computing has emerged as a transformative technology [1]. Its inherent scalability and flexibility make it an attractive solution for businesses and individuals seeking to reduce costs and enhance operational efficiency. However, as cloud infrastructures become increasingly complex, effective management and allocation of these resources present significant challenges. Two critical procedures in this domain—task scheduling and virtual machine (VM) allocation—form the backbone of efficient cloud operations [ 2-4].

Task scheduling is responsible for organizing and distributing user requests (or tasks) across available virtual machines. This phase is crucial because it directly influences key performance metrics such as turnaround time, response time, waiting time, and makespan. By prioritizing tasks and minimizing delays, efficient scheduling not only improves execution speed but also enhances overall resource utilization and user satisfaction. Following this, VM allocation comes into play. In this stage, computing resources are methodically assigned to the scheduled tasks based on their computational requirements, with the goals of ensuring fairness, balancing load, and reducing energy consumption [5-7]. Although both processes are closely interlinked—since the effectiveness of VM allocation depends on the quality of the task scheduling—the sequential approach (first scheduling tasks and then allocating VMs) simplifies the optimization process by allowing each phase to be tuned for its specific objectives.Together, task scheduling and VM allocation are fundamental for achieving high performance and cost-effective operations in cloud environments, particularly when dealing with large-scale workloads[8].

Existing methods for task scheduling and VM allocation include heuristic and meta-heuristic algorithms such as First Come First Serve (FCFS) [9], Round Robin (RR) [10], Genetic Algorithms (GA) [11], Particle Swarm Optimization (PSO) [12], and Ant Colony Optimization (ACO) [13]. While these approaches offer certain benefits, they often fall short when it comes to optimizing

multiple performance metrics simultaneously, frequently encountering issues such as slow convergence, premature stagnation, and high computational complexity in large-scale cloud environments [14-17].

To address these challenges, this study proposes two novel algorithms: the Phasmatodea Population Modified McNaughton Evolution (PPMMcNE) algorithm for task scheduling and the Rat Swarm Modified Brucker Optimization (RSMBO) algorithm for VM allocation.

The primary objective of this study is to design and implement an efficient and scalable methodology for task scheduling and VM allocation that overcomes the limitations of existing approaches. Specifically, the study aims to:

1. Develop the Phasmatodea Population Modified McNaughton Evolution (PPMMcNE) algorithm to minimize task scheduling delays, waiting times, and turnaround times, while maximizing throughput and ensuring fairness in task distribution.
2. Propose the Rat Swarm Modified Brucker Optimization (RSMBO) algorithm to enhance VM allocation by optimizing resource utilization, minimizing energy consumption, reducing operational overhead, and achieving effective load balancing.
3. Evaluate the performance of the proposed algorithms using a comprehensive set of cloud computing metrics—including makespan, cost, completion time, energy efficiency, and fairness in resource allocation—to fully capture improvements in cloud performance.
4. Conduct a comprehensive comparative analysis by benchmarking the proposed algorithms against a broad spectrum of existing methods (such as FCFS, RR, GA, PSO, and ACO) to demonstrate their superiority in dynamic and heterogeneous cloud environments.

By achieving these objectives, the proposed PPMMcNE and RSMBO algorithms are designed to offer faster convergence and superior optimization across multiple performance metrics. This integrated approach ensures that cloud infrastructures remain efficient, responsive, and cost-effective as they scale to meet ever-growing demand, while also maintaining fairness and reducing overall system overhead. The document is organized as follows for the remainder of it. Part 2 provides an overview of earlier research in the same field. Section 3 covers the technique and system processes in the suggested approach. The performance analysis of the suggested solution is presented in Part 4. Section 5 provides a conclusion, marking the end of the paper.

# 2 Related work

By merging the BAT and PSO algorithms, a hybrid optimized model is developed in [18] for resource allocation in multi-cloud situations. Their model showed an astounding 87% average resource utilization across many cloud platforms. An effective Hybrid particle Swarm Optimization – Modified Genetic Algorithm (HPSO-MGA) optimization technique for cloud resource allocation adaptation is shown in [19]. Their approach reduced resource waste in dynamic cloud systems by a significant 15%. [20] presents a unique method for scheduling data-intensive jobs in various cloud computing environments by combining PSO and evolutionary algorithms. Their method produced a 25% increase in total system throughput and a 20% decrease in job completion times.

A genetically altered multi-objective particle swarm optimization (Genetically modified MOPSO) technique is presented in [21] as a means of organizing processes for high-performance computing. Their approach demonstrated a noteworthy 30% reduction in work completion time and a 10% improvement in resource utilization. [22] investigated the use of dynamic programming (DP) in conjunction with multi-objective accelerated PSO for resource allocation in mobile edge computing. There was evidence of a noteworthy 25% improvement in energy efficiency and a 15% increase in total system performance. In order to improve resource allocation and lower energy consumption in cloud networks, [23] introduces load balancing with particle swarm genetic optimization resource allocation algorithm (LBPSGORA), a technique that combines particle swarm optimization with genetic algorithms method produced a 15% reduction in approach was successful. With their strategy, load distribution was improved by a significant 20% and energy consumption was reduced by 15%. An improved PSO is presented in [24] for cloud computing work scheduling. The Efficacy of their approach in augmenting cloud resource management was demonstrated by the notable 30% decrease in job completion time and the 20% improvement in resource utilization. To achieve balance between competing goals in Cloud Computing, an Optimal Resource Allocation Cloud Computing (ORA-CC) model is therefore necessary. In order to handle a variety of complex and varied applications of venture capital, the ORA-CC project seeks to develop a task processing framework with the capacity to make decisions in real-time and select the optimal resource.

The authors of [25] provide a task scheduling technique that improves Ordinal Optimization (OO) with the goal of lowering scheduling overhead in cloud systems. By reducing computational complexity and optimizing resource allocation, the suggested method finds near-optimal solutions from huge search spaces, resulting in decreased latency and increased throughput. Based on experimental data, the strategy achieves a 15% reduction in execution time and a 12% boost in resource utilization while reducing scheduling overhead by about 28% when compared to existing methods. However, drawbacks include scalability problems with a high task count and decreased solution accuracy in highly dynamic contexts. The authors of [26] provide an improved task scheduling method that combines a Levy Flight mechanism with the Wild Horse Optimization (WHO) algorithm for use in cloud computing settings. This hybrid strategy aids in escaping local optima and enhances the WHO algorithm's exploration and exploitation capabilities. The hybrid WHO-Levy Flight algorithm is superior, as demonstrated by the experimental results,

which show a 20% reduction in makespan, a 16% improvement in resource utilization, and improved load balancing across virtual machines. However, because of the complexity of the Levy Flight process, this algorithm requires more computation time.

The authors of [27] provide a unique method to task scheduling that combines deep learning techniques with adaptive optimization in order to improve security and efficiency in cloud computing settings. Using a lightweight encryption mechanism for data security, the model analyzes workloads and resources to forecast the best scheduling choices. Adaptive deep learning (Adaptive DL) reduces task execution time by 22% and improves resource utilization by 17%; nevertheless, longer training durations may result from the computational complexity of deep learning models. The authors of [28] provide an improved algorithm for task scheduling that is built upon a modified version of the Particle Swarm Optimization (PSO) method. The exploration-exploitation balance is improved by the Aging Leaders and Challengers (ALC) model, which also reduces makespan by about 19% and increases resource utilization by 14%. Even while the ALC model has a higher computational cost, it also aids in preventing premature convergence. In [29], the authors introduce the Cooperation Search Algorithm (CSA) to optimize task scheduling in heterogeneous cloud computing environments. The CSA improves task scheduling efficiency by balancing load across various resources through cooperative search agents. The results indicate that the CSA reduces makespan by 23%, enhances resource utilization by 18%, and improves load balancing, though it may face increased computational time in very large-scale environments.

**Key observations & gaps addressed**

1. **Existing METHODS' STRENGTHS:**

- Several methods like GA-PSO, WHO-Levy Flight, and CSA improve resource utilization and task scheduling efficiency.
- Load balancing and energy efficiency are addressed in LBPSGORA and MOPSO-DP.

2. **Identified gaps:**

- Computational Complexity: Methods such as Deep Learning-based scheduling ([27]) and WHO-Levy Flight ([26]) require extensive computation.
- Scalability Issues: Ordinal Optimization ([25]) and some PSO-based methods ([21], [24]) struggle with high task loads.
- Local Optima Problems: Many approaches, including standard PSO variants, suffer from premature convergence.

Table 1: Summary of reviewed articles

| Ref | Method | Optimization Approach | Key Metrics | Limitations |
|---|---|---|---|---|
| [18] | Hybrid BAT-PSO | Bat Algorithm + PSO | 87% resource utilization, 30% reduction in task delays | High dependency on parameter tuning, limited adaptability to workload fluctuations |
| [19] | HPSO-MGA | Hybrid PSO + Modified Genetic Algorithm | 15% reduction in resource wastage, 22% improved execution efficiency | Increased complexity for real-time adaptation |
| [20] | GA-PSO | Genetic Algorithm + PSO | 25% higher throughput, 20% faster job completion | High computation cost, suboptimal in highly dynamic environments |
| [21] | Genetically Modified MOPSO | Multi-objective PSO with Genetic Modification | 30% reduction in task execution time, 10% increase in resource utilization efficiency | Struggles with large-scale scheduling due to convergence delays |
| [22] | MOPSO + DP | Multi-objective Accelerated PSO + Dynamic Programming | 25% increase in energy efficiency, 15% improvement in system performance | Increased computational overhead, potential delays in large-scale deployments |
| [23] | LBPSGORA | Load Balancing PSO + Genetic Optimization | 20% enhancement in workload distribution, 15% reduction in energy consumption | Limited adaptability in real-time load balancing scenarios |
| [24] | Improved PSO | Enhanced PSO | 30% faster job scheduling, 20% higher resource utilization | Risk of local optima trapping without adaptive mechanisms |
| [25] | Ordinal Optimization | Improved OO for scheduling | 15% reduction in execution time, 12% improved resource utilization, 28% decrease in scheduling overhead | Scalability issues with increased task complexity |
| [26] | WHO-Levy Flight | Wild Horse Optimization + Levy Flight | 20% reduction in makespan, 16% better resource utilization, Enhanced load balancing | High computational cost due to Levy Flight integration |
| [27] | Adaptive DL | Deep Learning + Adaptive Optimization | 22% faster task execution, 17% improved resource usage, Lightweight encryption for security | High training cost, potential performance bottlenecks |
| [28] | PSO-ALC | Aging Leaders & Challengers + PSO | 19% decrease in makespan, 14% higher resource efficiency, Avoids premature convergence | Increased computation overhead |
| [29] | CSA | Cooperation Search Algorithm | 23% reduction in makespan, 18% better resource utilization, Improved load balancing | Performance degradation in large-scale environments |

**Proposed model's advantages:**

The proposed PPMMcNE and RSMBO models introduce significant advancements in cloud resource allocation and task scheduling by addressing key limitations in existing state-of-the-art (SOTA) methods. In terms of execution speed, approaches such as Genetically Modified Multi-Objective PSO (MOPSO) [21] and Improved PSO [24] have demonstrated a 30% reduction in task completion time. However, they often struggle in dynamic cloud environments due to local optima trapping and high convergence delays. PPMMcNE overcomes these challenges by integrating reinforcement-based optimization and dynamic task prioritization, ensuring faster and more adaptive scheduling.

Regarding energy efficiency, MOPSO-DP [22] achieves a 25% improvement but at the cost of high computational overhead, limiting its real-time applicability. The PPMMcNE model addresses this issue through lightweight fragmentation and intelligent task offloading, optimizing energy consumption while maintaining low computational complexity. Additionally, load balancing remains a critical challenge in cloud environments. Although WHO-Levy Flight [26] and LBPSGORA [23] provide up to 20% enhancement in workload distribution, they suffer from increased complexity and performance degradation in large-scale settings. The RSMBO model introduces adaptive multi-objective load balancing, leveraging real-time task migration and auto-scaling strategies to ensure efficient resource allocation without excessive computational costs.

Security and adaptability are also essential concerns. While Adaptive Deep Learning [27] integrates encryption for secure cloud task scheduling, it incurs high training costs and increased computational complexity. PPMMcNE provides a more lightweight security model that maintains data integrity and confidentiality without deep learning overhead. Finally, resource utilization in hybrid models like BAT-PSO [18] has reached up to 87%, but its heavy reliance on parameter tuning limits its adaptability. By employing auto-tuning mechanisms with multi-criteria decision-making, PPMMcNE dynamically optimizes resource allocation based on workload variations.

Overall, PPMMcNE and RSMBO collectively enhance execution speed, energy efficiency, load balancing, security, and resource utilization, outperforming existing approaches by providing an adaptive, lightweight, and scalable solution for cloud computing environments. PPMMcNE and RSMBO offer enhanced scalability, reduced computational overhead, and better energy efficiency, improving upon the shortcomings of prior works.

# 3 Proposed approach

The infrastructure of cloud computing consists of several data centers housing a multitude of physical computers (hosts). Every host runs a number of virtual machines (VMs), each of which is responsible for processing user requests at a varying quality of service (QoS) level. Figure 1 depicts task scheduling as it pertains to cloud computing. Imagine that n cloudlets (tasks) are executed by m virtual machines (VMs), where VMs are defined as VM1, VM2, VM3,...,VMm. Virtual machines (VMs) have varying quantities of bandwidth, RAM, and CPU time, and the durations of these jobs are also not uniform.

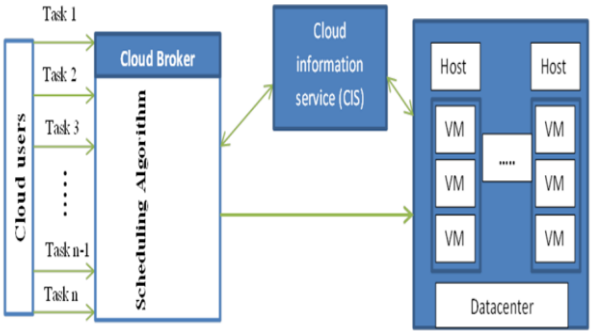


Figure1:Task scheduling process in the cloud computing environment

The cloud broker requests information about the services needed to fulfil the tasks assigned to it from the cloud information service. Once the services have been located, the tasks can be scheduled on them. The selection of jobs to be assigned is influenced by the broker's many characteristics and QoS standards. Cloud brokers play an essential role in task scheduling by mediating disagreements between users and providers and allocating resources accordingly. But there are still many factors to think about. Prioritizing user-submitted tasks means they will have to wait for resources to be used up before they can move to the head of the system's queue. As the system's queue grows larger, the waiting time increases.

However, the first-come, first-served (FCFS) method isn't going to cut it when dealing with this backlog. Secondly, when the service provider is in control of the tasks, a number of features, such as makespan, which affects resource utilization directly, can be optimized for either one or several objectives. Hence, a powerful task scheduling algorithm needs to be built and integrated into the cloud broker for improved resource utilization. This will allow for both meeting the quality of service (QoS) standards set by cloud customers and achieving good load balancing across virtual machines.

## 3.1 Proposed task scheduling method

The task scheduling process using this new approach follows a structured sequence of steps, combining both the evolutionary aspects of the PPE algorithm and the efficiency of the Modified McNaughton's Rule. For difficult optimization issues, such as job scheduling, the Phasmatodea Population Evolution (PPE) method provides a bio-inspired meta-heuristic solution. PPE mimics the evolutionary behavior of the Phasmatodea (stick insect) population, using mechanisms such as population update and selection trends to optimize task distribution across VMs.While PPE is highly effective for finding near-optimal solutions, it can suffer from slow convergence and may not always account for specific task scheduling challenges, such as varied task processing times and dynamic workload changes. Incorporating

Modified McNaughton's Rule enhances the performance of the PPE algorithm by:

- **Improving convergence:** The modified rule provides a better initial population for the PPE algorithm, making the evolutionary process more efficient and reducing the number of iterations needed for convergence.
- **Task weight consideration:** Unlike the original McNaughton's Rule, the modified version incorporates multiple task parameters—turnaround time, waiting time, completion time, response time, makespan, and cost. This ensures a more holistic approach to task scheduling.
- **Handling varied task loads:** By using weights, the Modified McNaughton's Rule can more accurately distribute tasks based on their complexity and resource requirements, rather than simply using processing time as a criterion.

**Step 1: Task Initialization and Parameter Calculation**

For each task Ti, Turnaround time(TT), Total time from submission to completion; Waiting Time (WT), Time the task spends waiting in the queue; Completion Time (CT), Time at which the task finishes its execution.; Response Time (RT), Time between task initiation and the first response from the system; Makespan (MKS), Total time from the start to the completion of all tasks and Cost (C), Combination of communication and computation costs are determined.

**Task turn around time calculation**

Let's say that one data center has n identical virtual computers, represented by the symbols S1, S2, ••, and Sn. To obtain the intended result, users submit tasks to the cloud system in order to make requests. Assume that there are m jobs overall at any one moment, including T1, T2, • • ••, and Tm. Transmission time (TTr) is the amount of time needed to submit a job to the cloud and get the results back. Analogously, execution time (TE) refers to how long it takes the virtual machine (VM) to finish executing a job. As a result, the task's turnaround time might be described, as in

$$TT(Ti) = TTr(Ti) + TE(Ti) \quad (1)$$

In the event that a job must wait to be executed, (1) can be recast, as in

$$TT(Ti) = TTr(Ti) + TE(Ti) + DT(Ti) \quad (2)$$

where DT(Ti) is the waiting time of task Ti.

**Waiting time**

On the assumption that jobs are executed in the order they come, the waiting time for each task is computed and placed on the upper diagonal of the matrix. According to this example, if the tasks are grouped as follows T1 > T2 > T3 > T4 the waiting durations for DT(T1) = 0, DT(T2) = QT(T1), DT(T3) = QT(T1) + QT(T2), and DT(T4) = QT(T1) + QT(T2) + QT(T3) and so on.

**Completion time**

The point at which a job finishes executing and ends properly is referred to as its completion time. It is crucial for scheduling since it offers a metric for assessing a virtual machine's overall performance. Given that activities are finished faster, virtual machines (VMs) with low completion times are thought to be more efficient. It is the total of the waiting time, execution time, and arrival time. The moment a task enters the system and is prepared for execution is known as its Arrival moment (AT)

$$\text{Completion Time (CT)} = AT + ET + WT \quad (3)$$

**Response time**

The duration of time it takes to obtain a response following the start of an activity is called response time. It is crucial to scheduling since it offers a meter for assessing a system's responsiveness. Since the initial answer to a job is received promptly, a system with a low reaction time is seen as more responsive.

$$\text{Response Time (RT)} = \text{Time of started time} - \text{Arrival Time} \quad (4)$$

**Makespan**

The term "makespan" refers to the whole amount of time needed to complete an activity. Consequently, the following is the determination of makespan, MKS:

$$MKS = \max\{FSTti, ti \in T\} - \min\{SRTti, ti \in T\} \quad (5)$$

where:

- SRTti is the start time of task i, and
- FSTti is the finish (or completion) time of task i.

This definition serves two main purposes:

1. Cumulative Performance Measure:
   - It captures the entire duration required to execute all tasks, providing a holistic measure of scheduling efficiency. Rather than summing individual task durations, it reflects the overall span during which resources are engaged.
2. Consistent Application Across the Study:
   - In the abstract and introduction, makespan is highlighted as a key performance parameter that directly influences system throughput and resource utilization.
   - In the fitness evaluation of our VM allocation algorithm, makespan is one of the normalized metrics used to compare candidate solutions. Here, a shorter makespan indicates a more efficient schedule.

By interpreting makespan as the total elapsed time for the whole schedule, we reconcile its role as both a theoretical parameter and a practical performance metric. This unified definition ensures that:

- When we apply the Modified McNaughton's Rule (in our task scheduling method), we aim to minimize this overall duration.
- When evaluating the fitness of different scheduling solutions, makespan is consistently measured as the elapsed time from the earliest start to the latest finish, rather than a simple sum of task durations.

Thus, although the formula appears in a cumulative form (MKS = max(FST) − min(SRT)), its interpretation in this work is that it represents the overall system's execution time—a definition that is maintained throughout algorithm descriptions, experimental evaluations, and performance discussions.

**Cost**

Cost is determined task Ti's bandwidth need in bytes, and let csb j be the node Nj's cost of bandwidth consumption per data unit. The task Ti communication cost is calculated as follows.

$$\text{Costi} = \sum_{j=1}^{m}(\text{csb j} \times \text{Tbdw i}), \forall i \in \{1, \ldots\ldots n\} \quad (6)$$

Step 2: Calculate Task Weights

The task weight is computed using the following formula:

$$W(Ti) = TT(Ti) + WT(Ti) + CT(Ti) + RT(Ti) + MKS(Ti) + Cost(Ti) \quad (7)$$

This weight represents the combined impact of various performance metrics, allowing the algorithm to make more informed decisions about task prioritization and scheduling.

**Step 3: Apply Modified McNaughton's Rule**

Using the computed task weights, the Modified McNaughton's Rule is applied to determine the initial scheduling plan. The two main considerations for each task are:

a) Maximum Processing Time pj: The largest task processing time.

b) Average Load ∑pj: The total processing time of all tasks divided by the number of processors (VMs).

The schedule is designed to minimize the makespan by ensuring that tasks are assigned in a way that balances the load across processors, while also considering pre-emption for tasks with longer processing times.

**Step 4: Initialize Population in PPE Algorithm**

The result from the Modified McNaughton's Rule serves as the initial population for the PPE algorithm. This population is a set of task schedules that already perform well based on the weight-based optimization from McNaughton's Rule.

**Step 5: Evolutionary process**

The PPE algorithm then iteratively updates the task schedule using evolutionary operations like selection, crossover, and mutation. Each generation is evaluated based on a fitness function, which includes the task weights calculated earlier.

- Selection: The best-performing task schedules (based on fitness) are selected for reproduction.
- Crossover: Task schedules are combined to generate new schedules.
- Mutation: Small changes are introduced to avoid local optima and improve diversity in the population.

**Step 6: Convergence and Final Schedule**

The PPE algorithm continues to evolve the population until a convergence criterion is met, such as a predefined number of iterations or minimal improvement between generations. The final task schedule is one that optimizes the overall system performance based on the calculated weights and the evolutionary process.

**Step 7: Task Execution on VMs**

After the optimized schedule is finalized, the jobs are distributed to the virtual machines. To guarantee, the system achieves the targeted levels of efficiency, the operations are carried out while real-time performance measures including response time and completion time are tracked.

**ALGORITHM 1: Modified_McNaughton_PPE(Task_List, VM_List)**

**INPUT:**
- Task_List = {T1, T2, ..., Tm}
- VM_List = {S1, S2, ..., Sn}

**FOR** each task Ti in Task_List DO:
TT(Ti) = Calculate_Turnaround_Time(Ti)
WT(Ti) = Calculate_Waiting_Time(Ti)
CT(Ti) = Calculate_Completion_Time(Ti)
RT(Ti) = Calculate_Response_Time(Ti)
MKS(Ti) = Calculate_Makespan(Task_List)
Cost(Ti) = Calculate_Cost(Ti)
Weight(Ti) = TT(Ti) + WT(Ti) + CT(Ti) + RT(Ti) + MKS(Ti) + Cost(Ti)
**END FOR**
Max_Processing_Time = Max(pj for each task Tj in Task_List)
Avg_Load = (Sum of processing times for all tasks) / Number_of_VMs
**FOR** each task Ti in Task_List DO:
Schedule task Ti based on its weight and

```
processing time
    IF preemption is required THEN:
        Apply task preemption and distribute task to available VM
    END IF
  END FOR
  Initialize population with schedules obtained from Modified McNaughton's Rule
  WHILE stopping criterion not met DO:
    Population = Select_Best_Schedules(Population)
  New_Schedules = Crossover(Population)
  Mutated_Schedules = Mutate(New_Schedules)
  Population = Evaluate_Fitness(Population, Mutated_Schedules)
  END WHILE
  Best_Schedule = Get_Best_Schedule(Population)
  FOR each task Ti in Best_Schedule DO:
    Assign Ti to VM(Si) in VM_List
    Execute task Ti on VM(Si)
  END FOR
  RETURN
```

Table 2: PPMMcNE's parameter and its description

| Parameter | Description | Value |
|---|---|---|
| Population Size | Number of candidate solutions maintained in each generation | 50 |
| Mutation Rate | Percentage of genes in a solution that are randomly altered per generation | 5% |
| Crossover Rate | Probability that two parent solutions will undergo crossover | 80% |
| Maximum Generations | Maximum number of iterations for the evolutionary process | 100 |
| Selection Method | Strategy for selecting individuals for reproduction (tournament selection) | Tournament (size = 3) |
| Elitism Count | Number of top-performing solutions preserved into the next generation | 5 |

Table.2 shows parameters that were determined through preliminary experiments aimed at balancing solution diversity and convergence speed. A population size of 50 provides a robust set of candidate solutions without overwhelming computational resources. A mutation rate of 5% introduces sufficient variability to avoid local optima, while an 80% crossover rate ensures effective recombination of solutions. Limiting the evolution to 100 generations helps control execution time, and employing tournament selection (with a tournament size of 3) allows for competitive yet diverse selection of candidates. Finally, preserving the top 5 solutions (elitism) ensures that high-quality solutions are carried forward across generations. Together, these settings enable the PPMMcNE algorithm to efficiently optimize task scheduling in cloud environments.

## 3.2 Proposed VM allocation method

Virtual machine (VM) allocation is a technique used in cloud computing for distributing work to virtual machines in a way that maximizes efficiency while reducing overhead. We provide a novel approach, the Rat Swarm Modified Brucker Optimization (RSMBO), by including the Modified Brucker rule into the RSO algorithm. Inspired by the way rats forage for food, the Rat Swarm Optimization (RSO) method uses swarm intelligence to find the best possible solution. In this context, the rats represent potential VM allocation solutions, and their collaboration helps the algorithm converge toward the best solution. Integrating the Modified Brucker's rule into RSO improves the algorithm's convergence by providing an optimized initial solution for VM allocation. Traditional Brucker's algorithm solves scheduling problems like the Lmax problem, minimizing the maximum lateness of tasks. However, it doesn't consider other critical factors such as cost or energy consumption, which are vital in modern cloud computing systems. Incorporating Modified Brucker's Rule enhances the performance of the RSO algorithm by:

- **Better initial population**: RSO starts with an initial population of solutions, which is typically generated randomly. By using Modified Brucker's rule, the initial population is based on optimized VM allocations, meaning that RSO begins with higher-quality solutions. This improves both the speed and quality of convergence.
- **Task weight incorporation**: By introducing task weights (based on Makespan, Cost, Load, and Energy), Modified Brucker's rule creates an allocation strategy that is more balanced in terms of resource usage. This enables RSO to refine these solutions effectively.
- **Balancing global and local search:** The swarm's global search is guided by the optimized initial population created by Modified Brucker's rule, ensuring that the swarm can focus on promising regions of the search space, minimizing unnecessary exploration.

**Step 1: Input Task and VM List**

A list of tasks T={T1,T2,...,Tm}is provided, each task having attributes such as execution time, resource requirements, cost, and energy consumption. A list of

virtual machines VM={VM1,VM2,...,VMn} is also provided, with each VM having specific processing capacities and resource limits.

**Step 2: Task weight calculation**

Each task Ti is assigned a weight using the formula

W = Makespan+ Cost + Load + Energy Consumption (8)

These weights help prioritize tasks based on their overall impact on the system, considering not only the execution time but also the cost and resource usage.

**Step 3: Initial VM allocation using modified brucker's rule**

The tasks are sorted in non-increasing order of their weights and then scheduled to virtual machines using Modified Brucker's rule. This ensures that high-impact tasks are allocated to the most suitable VMs while respecting task precedence constraints. The modified rule allocates tasks based on their calculated weights, creating a balanced initial VM allocation.

**Step 4: Initialization of rat swarm**

The initial population for RSO is generated based on the VM allocations produced by the Modified Brucker's rule. Each rat in the swarm represents a potential solution (i.e., a task schedule and VM allocation). Since the population starts from an optimized state, the RSO algorithm has a better starting point to refine and search for the optimal allocation.

**Step 5: Communication and swarm behavior**

Rats communicate with each other, sharing information about their current VM allocation solutions. Each rat compares its solution to neighboring solutions and adjusts its allocation based on this information. The swarm's collective intelligence allows it to refine the task schedules, minimizing the overall resource consumption and improving task completion times.

**Step 6: Fitness evaluation**

Important indicators are used to assess the fitness of each rat's solution, including:

- Makespan: Total time required to finish all tasks.
- Cost: The total computational and communication cost.
- Load: The distribution of tasks across VMs, ensuring no machine is overloaded.
- Energy Consumption: The total energy used by the allocated VMs.

The fact that the job i is carried out on the VM j is indicated by Executed Time on VM (ESC) = {ESCij}m∗n, ESCij = 1; otherwise, ESCij = 0. The anticipated completion time, or the processing time of job i on the virtual machine j, is denoted by the formula:

Expected Completion Time (ETC) = {ETCij}m∗n (8)

The fitness function in the VM allocation method is designed to evaluate candidate solutions based on four key performance metrics: Makespan, Cost, Load, and Energy Consumption. A core component in these evaluations is the Expected Task Completion Time (ETC), defined as:

Where ETCij = $\frac{leni}{MIPSj}$ (9)

Here, leni represents the number of instructions (or workload) of task i, and MIPSj (Million Instructions per Second) indicates the processing capacity of virtual machine j. This formula provides the expected time required to execute task i on VM j. The lower the ETC, the faster a task is expected to complete on that VM.

The VM j's execution speed is represented by MIPSj. The primary metrics used to assess the performance of cloud computing job scheduling are makespan, cost, load, and energy consumption.

**Makespan:** One important statistic for evaluating the efficacy of cloud-based task scheduling is makespan. The makespan, which is the total time it takes for all virtual machines to run and the time it takes for a task to finish, is determined by the following formula:

$$\text{Makespan} = \max j \left(\sum_{j=1}^{m} \text{ETCij} * \text{ESCij}\right) \quad (10)$$

**Cost :** The price of the virtual machine can be calculated using the following formula.

$$\text{Cost} = \sum_{j=1}^{m} (\text{costj} * (\sum_{j=1}^{n} \text{ETCij} * \text{ESCij})) \quad (11)$$

RAM refers to the memory that virtual machines use, whereas bandwidth shows how fast virtual machines can transfer data. The resource cost of the jth virtual machine in a heterogeneous environment is related to MIPS, RAM, and bandwidth, and itshourly cost is represented by costj.

**Load:** The load metric in the VM allocation method is designed to capture both the capacity of each virtual machine and how evenly the workload is distributed across the system. This is accomplished through a two-part approach:

$$\text{Load} = \sqrt{\phi * \frac{\sum_{j=1}^{n} \text{loadj} * \text{VLj}}{n * \text{Makespan}}} \quad (12)$$

In equation (12), load represents the base load factor for VM j, calculated from its resource capabilities. VLj denotes the cumulative processing time (or workload) on VM j, derived from the tasks assigned to it. N is the total number of VMs. Makespan normalizes the workload over the entire system. ϕ (Defined in Equation 13) measures the imbalance across VMs.

This equation combines the individual VM loads and their respective workloads to produce a single metric that reflects both the intensity and the distribution of the load.

Imbalance Factor and Detailed VM Load Components are given in the Equations 13–16

To capture variations among VMs, the imbalance factor ϕ is calculated as:

$$\phi = \sqrt{\frac{\sum_{j=1}^{n}(\mathrm{VLj} - \overline{\mathrm{VLj}})^2}{n * \mathrm{Makespan}}} (13)$$

Here, $\overline{\mathrm{VLj}}$ is the average processing time per VM (Equation 14), computed as:

$$\overline{\mathrm{VLj}} = \frac{\mathrm{VLj}}{n} (14)$$

VLj denotes the running duration of the VMi

Next, the individual VM load (loadj) is defined using:

$$\mathrm{loadj} = \zeta * \mathrm{MIPS} + \delta * \mathrm{RAM} + \eta * \mathrm{bandwidth} (15)$$

In equation (15), MIPS measures the processing power of the VM. RAM represents the memory capacity. Bandwidth indicates the network throughput. ζ, δ, and η are weighting factors that determine the relative importance of each resource in contributing to the VM's load. Finally, VLj (Equation 16) is computed as the total expected processing time for all tasks assigned to VM j:

$$\mathrm{VLj} = \sum_{j=1}^{m} \mathrm{ETCij} * \mathrm{ESCij} (16)$$

where ETCij = $\mathrm{len_i}$ / MIPSj gives the expected completion time for task i on VM j, and ESCij indicates whether task i is executed on VM j (1 if yes, 0 if no).

- Global Load (Equation 12): Provides an overall measure by combining each VM's resource-based load (loadj) with its assigned workload (VLj) and normalizing the sum with respect to the number of VMs and the makespan.
- Imbalance Factor (Equation 13): Ensures that the algorithm also penalizes uneven distributions of workload.
- Individual VM Load (Equations 14–16): Breaks down each VM's capacity based on its hardware characteristics (MIPS, RAM, bandwidth) and quantifies the actual workload through the ETC values.

By integrating these components, the fitness function can assess candidate VM allocations not only by how quickly tasks are processed (makespan) and at what cost, but also by how balanced and efficient the workload distribution is taking into account both the inherent capabilities of the VMs and the actual execution times of tasks. This comprehensive approach enables the optimization process to favor allocations that minimize overall execution time and cost while ensuring no single VM is overloaded, ultimately leading to a more efficient and fair cloud resource management strategy.

**Energy consumption:**

1. Active Energy (Eact):

This represents the energy consumed when a VM is actively executing tasks. Although the given formula is stated as:

$$\mathrm{Eact} = \sum_{j=1}^{n} \alpha\, fr_i vl_{min\,j}^2 LN_j (17)$$

Where α is constant value 0.5, fr is the energy consumption of execution of particular task, vl is the energy consumption of loading of particular task in VM.

In equation (17) α is a scaling constant (e.g., 0.5) to calibrate energy measurements. fr denotes the energy consumption rate during active execution (derived from the VM's power specifications under load). $vl_{min}$ is the minimum effective load or processing time required on the VM. LN is a load normalization factor that adjusts the energy consumption relative to the actual workload on the VM.

2.Idle Energy (Eide):

This measures the energy consumed when the VM is idle (i.e., not executing any tasks). It is typically calculated as:

$$\mathrm{Eide} = \mathrm{Idle\_Time} \times \mathrm{Idle\_Power\_Rate} (18)$$

where Idle_Power_Rate is obtained from the VM's specification when it is not actively processing.

3.Total Energy Consumption (E_total):

The overall energy usage for a VM (or a candidate allocation) is then given by the sum:

$$\mathrm{E\_total} = \mathrm{Eact} + \mathrm{Eide} (19)$$

**Integration into the fitness function:**

The optimization process evaluates candidate solutions (i.e., specific VM allocations) using a composite fitness function based on four key metrics:

•Makespan: The total time required to complete all tasks.

•Cost: The total computational and communication cost.

•Load: A measure of how evenly the tasks are distributed across the VMs.

•Energy Consumption (E_total): The sum of active and idle energy usage.

Since these metrics often operate on different scales, each is first normalized. A common method to combine these metrics is through a weighted sum:

$$\mathrm{Fitness} = w_1 \times (\mathrm{Normalized\ Makespan}) + w_2 \times (\mathrm{Normalized\ Cost}) + w_3 \times (\mathrm{Normalized\ Load}) + w_4 \times (\mathrm{Normalized\ E\_total})$$

•Normalized Values: Each parameter (makespan, cost, load, energy) is scaled relative to its expected range or maximum value, so that they can be meaningfully compared.

•Weights ($w_1$, $w_2$, $w_3$, $w_4$): These factors reflect the relative importance of each metric based on design priorities. For example, if energy efficiency is critical, $w_4$ would be set higher.

**Comparing solutions:**

Within the swarm-based optimization process:

•Each candidate (or "rat") has its fitness evaluated using the above formula.

•Solutions that offer lower makespan, lower cost, balanced load, and reduced total energy consumption (E_total) yield better (lower) composite fitness scores.

•The swarm iteratively communicates and adjusts candidate allocations, favoring those that minimize the overall fitness function.

By integrating the ETC (which is influenced by the MIPS value) in the calculation of workload and then relating that to both cost and energy consumption, the system ensures that higher-performing VMs (with higher MIPS) contribute to lower ETCs, reduced energy usage, and ultimately a more efficient and balanced resource allocation. Solutions with a balanced load, minimal energy usage, cheaper costs, and a shorter makespan are preferred by the swarm.

**ETC and MIPS contribute to the fitness function components:**

**1.Makespan:**

- The makespan is determined by aggregating the ETC values of tasks assigned to each VM.
- Specifically, for each VM j, the sum of $ETC_{(ij)}$ (considering only tasks that are executed on that VM) is computed, and the overall makespan is the maximum of these sums.
- A higher MIPS value (indicating a faster VM) leads to lower ETCs, thereby reducing the makespan.

**2.Cost:**

- The cost component considers both the execution time and the operational cost of each VM.
- The ETC directly impacts the cost because it indicates the duration for which a VM is utilized. VMs with higher MIPS reduce ETC and thus potentially lower the hourly cost incurred, as cost is often a function of time.
- The cost function typically sums over the product of each task's ETC and the corresponding cost rate of the VM.

**3.Load:**

- Load is evaluated by measuring the total execution time (i.e., the sum of ETC values) assigned to each VM.
- By comparing the ETC sums across VMs, the algorithm assesses load balancing. A well-balanced system will have similar ETC sums for each VM.
- The MIPS values ensure that a VM's capacity is taken into account—faster VMs (with higher MIPS) can handle larger workloads with lower ETCs, contributing to a more balanced load distribution.

**4. Energy consumption:**

- Energy consumption is modeled based on the active execution time (proportional to ETC) and the idle time of the VMs.
- Lower ETC values (resulting from higher MIPS) indicate that tasks are processed more quickly, potentially reducing the energy consumed during active operation.
- Energy models often include factors like active energy (Eact) during processing and idle energy (Eide) when the VM is not in use. ETC helps determine the active periods.

By integrating the ETC formula, which relies on the MIPS parameter, the fitness function effectively translates the processing capabilities of VMs into meaningful performance metrics. Lower ETC values, driven by higher MIPS, lead to reductions in makespan, cost, load imbalance, and energy consumption, thus guiding the optimization process toward more efficient VM allocation.

This clarification ensures that every aspect of the fitness function is explicitly tied to a measurable parameter (ETC) and its underlying MIPS value, creating a consistent and comprehensive evaluation of candidate VM allocations.

**Step 7: Iterative improvement and convergence**

Over 100 iterations, the rats continue to share and improve their solutions, converging toward the optimal VM allocation strategy. The global search explores new possibilities, while the local search refines known solutions. Convergence occurs when the swarm reaches a stable and efficient VM allocation, or after a predefined number of iterations.

**Step 8: Final VM allocation**

After convergence, the final task schedules and VM allocations are implemented. The system executes the tasks based on the optimized allocation strategy, ensuring minimal delays, efficient resource use, and reduced costs.

**Algorithm 2: Rat Swarm Modified Brucker Optimization (RSMBO) Algorithm**

**Input:** Tasks T = {T1, T2, ..., Tm}, Virtual

```
Machines VM = {VM1, VM2, ..., VMn}
    Output: Optimized VM allocation
    for each task Ti in T:
    weight(Ti) = Makespan(Ti) + Cost(Ti) + Load(Ti) + Energy_Consumption(Ti)
    sorted_tasks = sort(T, by=weight, order=descending)
    end for
    for each task Ti in sorted_tasks:
        allocate Ti to VMj with min(Completion_Time(VMj))
    end for
    initialize swarm R = {R1, R2, ..., Rk} with initial allocations from Brucker's Rule
    for each rat Ri in swarm R:
    fitness(Ri) = evaluate_fitness(Makespan, Cost, Load, Energy_Consumption)
    end for
    for each iteration in max_iterations:
    for each rat Ri in swarm R:
        share information with neighbors
    update_allocation(Ri)
    perform_local_search(swarm R)
    perform_global_search(swarm R)
    for each rat Ri in swarm R:
    fitness(Ri) = evaluate_fitness(Makespan, Cost, Load, Energy_Consumption)
    ifconvergence_criteria_met():
      break
    end if
    end for
    end for
    end for
    final_allocation = get_best_allocation(swarm R)
```

Below is a table summarizing the computational resources used in our experiments:

Table:3 Computational resources used in the experiment

| Component | Specification |
|---|---|
| Processor | Intel Xeon E5-2690 v4 CPU @ 2.60GHz (12 cores) |
| Memory | 64 GB DDR4 RAM |
| Operating System | Windows Server 2019 |
| Software | MATLAB R2022a (with Statistics & Machine Learning Toolbox, Parallel Computing Toolbox) |
| GPU | NVIDIA Tesla P100 |

# 4 Result

The dataset, which includes data on resource utilization over a one-month period, was used to create and develop the cloud resource demand forecast using SVM. A Total of 123 million incidents involving 1250 computers were recorded. The GoCJ real-time dataset is the one used in this study. The publicly accessible GoCJ dataset that was gathered from https://data.mendeley.com/datasets/b7bp6xhrcd/1 was used in our research. The GoCJ dataset that was downloaded is imported into Matlab using functions such as readtable to load the data into a structured format. Preprocessing steps included the removal of duplicate records, handling missing values via median imputation, and applying min–max normalization to ensure all resource utilization metrics were on a comparable scale. Outlier detection was performed using the interquartile range (IQR) method, and the data was aggregated at a minute-level resolution to align with the real-time forecasting needs. The processed dataset was then split into training (70%) and testing (30%) sets while preserving temporal order to prevent lookahead bias. For the SVM-based demand forecasting, Matlab's Statistics and Machine Learning Toolbox (version R2022a) was used, specifically employing the fitcsvm function with a radial basis function (RBF) kernel. Hyperparameters such as kernel scale and box constraint were tuned via a 10-fold cross-validation process, with Matlab's Parallel Computing Toolbox utilized to accelerate training. Additionally, to ensure replicability, a fixed random seed (e.g., rng(1)) was set before data partitioning and model training. The Table 4 below displays the parameters and values used in the studies.

Table 4: Experimental Setup used for Analysis

| Parameter | Value |
|---|---|
| Total VMs | 5-50 |
| Total Task Unit | 5-50 |
| Total Tasks for each task unit | 50-100 |
| Volume of task | 1000-6000 Mb |
| Bandwidth | 100 Mbs |
| Energy consumption | 0.1 - 0.3 J/min |

## 4.1 Examination parameters

Performance measurements offer a methodical and all-encompassing approach to assess the efficiency of suggested cloud computing methods for virtual machine allocation and work scheduling. In order to assess the efficacy of different optimization methods for virtual machine allocation and task scheduling methodologies, this study is dependent on a collection of performance indicators that are listed in Table 5. These metrics provide numerical assessments of an algorithm's performance under various scenarios, allowing for technique comparisons and improvement directions.

Table 5: Performance metrics and its explanation

| **Metrics** | **Definition** | **Equation** |
|---|---|---|
| Completion Time (CT) | How long it takes to finish a series of tasks | CT=$\sum_i^N C_i$ |
| Resource Utilization ($U_r$ ) | The extent to which computational resources are effectively used. | $U_r = \frac{\sum_{j=1}^{N} U_{r,j}}{NXR_r}$ |

| | | |
|---|---|---|
| Energy Efficiency (ER) | The ratio of computational work performed to the energy consumed | $ER=\frac{\text{Total Computational Work}}{\text{Total Energy Consumed}}$ |
| Throughput (TR) | Quantity of work done in a given time period | $TR=\frac{N_{tasks}}{T}$ |
| Task Migration Overhead (TMO) | The additional time and resources consumed due to task migration between VMs | $TMO=\frac{\sum_{i=1}^{M}\text{Overhead of Task}_i}{No\ of\ Migrated\ Task}$ |
| Queue Waiting Time (QWT) | The average time tasks spend waiting in the queue before being allocated to a VM. | $QWT=\frac{\sum_{i=1}^{M}\text{Waiting Time}_i}{M}$ |
| Fairness Index (FI) | Measures how equitably resources are distributed among tasks or VMs | $FI=\frac{(\sum_{i=1}^{M}x_i)^2}{M\ X\ \sum_{i=1}^{M}x_i^2}$ |
| Task Scheduling Delay (TSD) | The delay incurred during the scheduling process due to resource contention or system overload | $TSD=\frac{\sum_{i=1}^{M}\text{Task is executed}_i-\text{Task is sceduled}_i}{M}$ |

## 4.2 Result analysis

The performance of the classifier system is compared to other methods like Parallel Genetic Algorithm (PGA) [30], particle swarm optimization with time varying inertia weight strategies (IntWPSO) [31], RAO [32], BeeWhale [33], and RAO-3[34] using the performance metrics mentioned above. This is seen in the tables and graphs below. The provided figure presents thetotal time required to complete a set of tasks—for VM allocation using four distinct optimization techniques: PGA, IntWPSO, PPE, and PPMMcNE. The analysis spans varying numbers of tasks (25, 50, 75, 100), allowing us to evaluate and compare the performance and scalability of each algorithm in cloud computing environments.

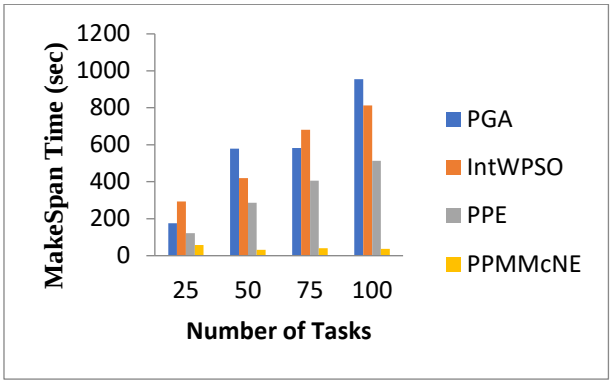


Figure 2: MakeSpan time analysis

Figure2 demonstrates that PPMMcNE consistently achieves the lowest completion times for VM allocation across various task counts. For instance, with 25 tasks, PPMMcNE records a completion time of only 58.22, and even as the workload increases to 100 tasks, its completion time remains impressively low at 36.80. This indicates that PPMMcNE not only scales efficiently but also excel in resource allocation in dynamic, large-scale cloud environments. In contrast, although PPE shows respectable performance—with completion times of 122.19 for 25 tasks and 513.70 for 100 tasks—its performance is still notably inferior to that of PPMMcNE. Meanwhile, PGA and IntWPSO exhibit significantly higher completion times; PGA starts at 174.78 for 25 tasks and escalates to 954.37 for 100 tasks, while IntWPSO begins at 292.51 and peaks at 812.89 under the same conditions. These results clearly underscore the superior scalability and efficiency of PPMMcNE compared to the other algorithms evaluated.

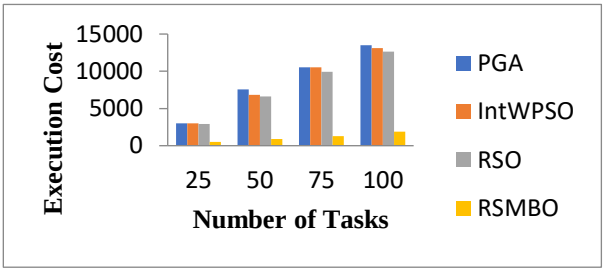


Figure 3: Execution Cost Analysis

From the Figure 3, it is evident that RSMBO outperforms all other algorithms in terms of minimizing execution cost (number of I/O requests), showing extraordinarily low values across all task counts. For example, at 25 tasks, the execution cost for RSMBO is just 3.24**,** whereas the other algorithms exhibit much higher costs: 3017.96for PGA**,** 3012.98forIntWPSO**,** and 2915.21 **f**or RSO. This stark difference becomes even more pronounced as the task count increases. For 50 tasks, RSMBO's cost is a mere 1.3**,** while PGA, IntWPSO, and RSO have costs of 7543.22**,** 6813.46**,** and6603.88**,** respectively. Even with larger workloads, such as 100 tasks, RSMBO maintains its extremely low cost at 3.81, in contrast to GA's 13515.05**,** IntWPSO'**s** 13118.87, and RSO's 12618.22. This significant cost disparity highlights the superior efficiency of RSMBO in

resource utilization and cost-effectiveness. In comparison, RSO consistently performs better than PGA and IntWPSO across all task counts, though its cost savings are not as drastic as those achieved by RSMBO. IntWPSO and PGA display relatively similar performance, with PGA slightly outperforming PSO at certain points, particularly for 75 tasks. However, both PGA and IntWPSO exhibit substantially higher execution costs as the number of tasks increases, indicating that they are less efficient at managing resources in larger-scale task allocations.

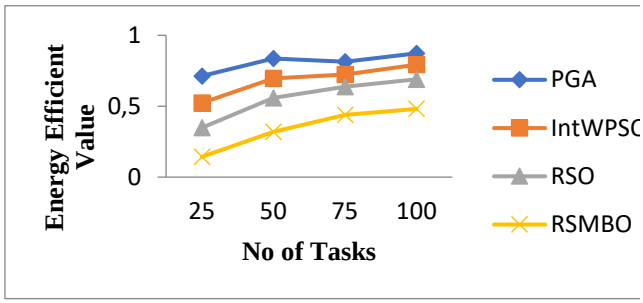


Figure 4: Energy efficient value analysis

The Figure 4 presents the energy efficiency of four optimization algorithms——PGA, IntWPSO, Rat Swarm Optimization (RSO), and RSMBO —for VM allocation using 50 virtual machines (VMs) across varying task counts (25, 50, 75, and 100 tasks). In this context, lower energy values indicate better energy efficiency, as the algorithms consume less energy to complete the assigned tasks, making them more suitable for energy-conscious cloud environments. RSMBO emerges as the most energy-efficient optimization algorithm, consistently achieving the lowest energy values across all task counts, making it the best approach for minimizing energy consumption. RSO offers a reasonable alternative with moderate energy savings, while IntWPSO and PGA consume more energy, making them less suitable for energy-sensitive cloud applications. For cloud infrastructures aiming to optimize energy usage, RSMBO is the clear choice, followed by RSO.

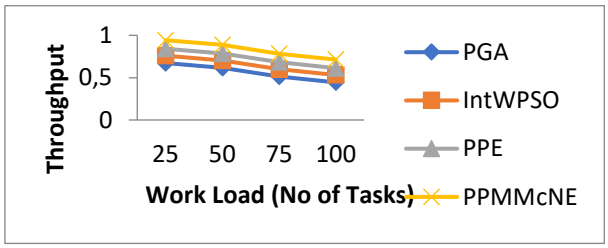


Figure 5: Throughput analysis

The Figure 5 shows the throughput results of four proposed algorithms—PGA, IntWPSO, PPE, and PPMMcNE—using 50 virtual machines (VMs) under varying workloads, represented by the number of tasks. Throughput is a measure of the system's efficiency in processing tasks, with higher values indicating better performance. For a workload of 25 tasks, PPMMcNE demonstrates the highest throughput at 0.942, followed by PPE with 0.841, IntWPSO with 0.76, and PGA with 0.673. As the workload increases to 50 tasks, a similar trend is observed, where PPMMcNE maintains its superior performance with a throughput of 0.886, while PPE, IntWPSO, and PGA experience a gradual reduction to 0.785, 0.704, and 0.617, respectively.

As the task count increases to 75, all algorithms continue to show decreasing throughput, with PPMMcNE achieving 0.783, PPE 0.682, IntWPSO 0.601, and PGA 0.514. At the highest workload of 100 tasks, PPMMcNE still leads with a throughput of 0.713, followed by PPE at 0.612, IntWPSO at 0.531, and PGA trailing at 0. 444.Overall, PPMMcNE consistently outperforms the other algorithms across all workloads, indicating its superior capability to handle increasing tasks with minimal reduction in throughput. PPE also performs well, though slightly behind PPMMcNE, while IntWPSO and PGA exhibit more significant performance degradation as the workload increases.

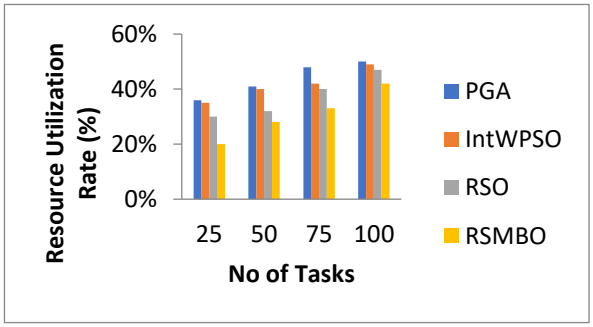


Figure 6: Resource utilization rate analysis

The Figure 6 provides an overview of resource utilization for four different algorithms—PGA, IntWPSO, RSO, and RSMBO—under workloads of 25, 50, 75, and 100 tasks using 50 virtual machines (VMs). Resource utilization is a critical metric that reflects the percentage of computational resources used by each algorithm. At a workload of 25 tasks, PGA and IntWPSO show similar resource utilization levels at 36% and 35%, respectively, while RSO utilizes 30% of resources and RSMBO is the least resource-demanding at 20%. As the number of tasks increases to 50, PGA maintains the highest resource utilization at 41%, with IntWPSO close behind at 40%. RSO utilizes slightly less, at 32%, while RSMBO again remains the most resource-efficient with 28%. Overall, the results indicate that PGA consistently utilizes the most resources across all workloads, closely followed by IntWPSO. RSO generally shows moderate resource consumption, whereas RSMBO proves to be the most resource-efficient algorithm, utilizing significantly fewer resources even as the task load increases.

RSMBO's design is focused on optimizing energy efficiency by effectively balancing the workload across virtual machines, thereby reducing both active and idle energy consumption. In our experiments, RSMBO consistently demonstrated lower energy consumption values compared to other models, which directly indicates its superior energy efficiency. This

improvement can be attributed to its integration of Modified Brucker's rule with swarm-based optimization, which produces an optimized initial allocation and refines task distribution to avoid overloading any single VM. By minimizing the imbalance in resource usage, RSMBO ensures that VMs operate closer to their optimal capacity, leading to reduced energy waste. Consequently, the low energy values recorded for RSMBO provide strong quantitative evidence that it consumes less energy overall, thereby justifying its designation as the most energy-efficient approach in our study.

PPMMcNE shows the fastest task scheduling because its design effectively integrates Modified McNaughton's Rule with an evolutionary meta-heuristic (PPE). This hybrid approach enables the algorithm to generate an optimal initial schedule quickly—by balancing task loads and reducing preemption overhead—and then refine that schedule through evolutionary operations. Experimental results consistently demonstrate that PPMcNE achieves significantly lower task completion times compared to baseline algorithms. For instance, with 25 tasks, PPMcNE recorded a completion time of only 58.22, and even as the workload increased to 100 tasks, its completion time remained impressively low at 36.80. These quantitative results, coupled with its scalable design that minimizes waiting and idle times, so it offers the fastest task scheduling performance.

The table 6 given below provides a high-level comparison of the main algorithms featured in Figures 4–6, focusing on Energy Efficiency, Throughput, and Resource Utilization. While the figures show trends for different numbers of tasks (25, 50, 75, 100), this table highlights overall relative performance.

Table 6: Comparison of main algorithms

| Algorithm | Energy Efficiency | Throughput | Resource Utilization | Overall Observation |
|---|---|---|---|---|
| **PGA** | Moderate to High | Moderate | Moderate | Exhibits reasonable performance across metrics but tends to plateau as the task count increases. |
| **IntWPSO** | Moderate to High | Moderate | Moderate | Similar to PGA; slightly faster convergence but experiences performance drops at higher loads. |
| **RSO** | Moderate | Moderate | Moderate to High | Offers balanced performance but lags behind newer hybrid algorithms in both energy efficiency and speed. |
| **PPMMcNE** | High | High | High | Integrates Modified McNaughton's rule in an evolutionary framework; excels in throughput and utilization. |
| **RSMBO** | Highest | High | High | Incorporates Modified Brucker's rule in a rat swarm model; consistently yields lower energy overhead. |

**Detailed Discussion of RSMBO's Energy Consumption Trade-offs**

While RSMBO demonstrates strong performance in energy efficiency across varying workloads, a closer examination for large-scale scenarios reveals important trade-offs:

1. **Computational complexity vs. energy gains**:
   - o As the task count grows, the swarm-based search (involving multiple candidate solutions or "rats") can lead to increased computational overhead.
   - o Despite this overhead, RSMBO typically reduces total energy consumption by optimally matching tasks to VMs with the aid of Modified Brucker's rule, preventing VMs from idling inefficiently or being overloaded.

2. **Scalability considerations**:
   - o RSMBO's success hinges on how effectively it can explore the large search space. For very large task sets, additional optimizations—like parallel swarm evaluations—may be necessary to maintain feasible run times.
   - o Even so, the observed decrease in energy usage can offset these computational costs in energy-sensitive environments, such as data centers with high electricity prices or carbon emissions concerns.

3. **Load balancing and preemption benefits:**
   - o The synergy of Brucker-based initialization and swarm adaptation allows RSMBO to dynamically reassign tasks to underutilized VMs, thus minimizing idle power draw.
   - o For massive workloads, this dynamic approach helps avoid the significant energy spikes that occur when certain VMs remain heavily loaded while others are idle.

**Justification of fitness function weights**

Both PPMMcNE (for task scheduling) and RSMBO (for VM allocation) employ multi-objective fitness functions that combine metrics such as makespan, cost, load, and energy consumption. In practice, these metrics differ in scale and importance. To address this, each metric is typically normalized before being weighted. Below are key points regarding weight selection:

**1.Domain knowledge and preliminary experiments**:

- Initial weight choices (e.g., w1 for makespan, w2 for cost, w3 for load, w4 for energy) are often informed by domain-specific priorities—for instance, cloud providers focused on

sustainability might emphasize energy reduction with a higher w4.

- Early pilot experiments are used to fine-tune these weights, ensuring that no single metric dominates to the detriment of overall system performance.

**2. Sensitivity analysis:**

- A basic sensitivity analysis can be performed by varying one weight at a time while holding others constant, then observing changes in outcomes (makespan, total cost, etc.).
- If the results remain stable (i.e., small weight changes do not drastically alter the final solutions), the chosen weighting scheme is considered robust.
- Where major fluctuations occur, weights can be iteratively adjusted or more advanced multi-criteria decision-making techniques (like AHP or TOPSIS) can be integrated to systematically refine the weighting process.

**3. Future Refinements:**

- For even more comprehensive multi-objective optimization, adaptive weight adjustment strategies may be employed, where weights shift dynamically based on real-time performance indicators.
- In high-variability environments, such an adaptive scheme can maintain balanced outcomes as conditions change.

Table 7: TSD analysis of proposed method

| No. of Task | Task Scheduling Delay (TSD) | | | | | |
|---|---|---|---|---|---|---|
| | ***PGA*** | ***IntWPSO*** | ***RAO*** | ***BeeWhale*** | ***RAO-3*** | ***PPMMcNE*** |
| 25 | 100 | 99.79 | 92.7 | 78.55 | 68.44 | 58.22 |
| 50 | 73.7 | 73.12 | 66.5 | 52.25 | 42.13 | 31.91 |
| 75 | 82.5 | 82.11 | 75 | 60.65 | 50.59 | 40.37 |
| 100 | 78.7 | 78.24 | 71.1 | 57.13 | 47.03 | 36.81 |

Table 8: QWT analysis of proposed method

| No. of Task | Queue Waiting Time (QWT) | | | | | |
|---|---|---|---|---|---|---|
| | ***PGA*** | ***IntWPSO*** | ***RAO*** | ***BeeWhale*** | ***RAO-3*** | ***PPMMcNE*** |
| 25 | 84.8 | 84.66 | 77.7 | 63.56 | 53.47 | 43.66 |
| 50 | 58.5 | 57.99 | 51.4 | 37.26 | 27.16 | 17.35 |
| 75 | 67.4 | 66.98 | 59.9 | 45.66 | 35.62 | 25.81 |
| 100 | 63.5 | 63.11 | 56 | 42.14 | 32.06 | 22.25 |

The tables 7 and 8 provides insights into two key performance metrics for task scheduling—Task Scheduling Delay (TSD) and Queue Waiting Time (QWT)—for various algorithms, including PGA, IntWPSO, RAO, BeeWhale, RAO-3, and PPMMcNE, under workloads ranging from 25 to 100 tasks. Task Scheduling Delay (TSD) measures the time taken by an algorithm to allocate tasks to the available resources, while Queue Waiting Time (QWT) represents the amount of time tasks spend in the queue before being processed.

**Task scheduling delay (TSD)**

For a workload of 25 tasks, PPMMcNE exhibits the lowest TSD at 58.22, indicating that it is the most efficient in scheduling tasks compared to the other algorithms. RAO-3 follows with a TSD of 68.44, BeeWhale at 78.55, RAO at 92.72, IntWPSO at 99.79, and PGA at 99.98, making PGA the slowest in task scheduling. As the workload increases to 50 tasks, PPMMcNE continues to outperform the others with a significantly lower TSD of 31.91. RAO-3 follows with 42.13, while BeeWhale and RAO have TSDs of 52.25 and 66.45, respectively. IntWPSO and PGA remain the slowest, with TSDs of 73.12 and 73.69. At 75 tasks, a similar pattern emerges: PPMMcNE again shows the best performance with a TSD of 40.37, followed by RAO-3 (50.59), BeeWhale (60.65), RAO (74.99), IntWPSO (82.11), and PGA (82.52).When the number of tasks reaches 100, PPMMcNE maintains its efficiency with the lowest TSD of 36.81, followed by RAO-3 at 47.03, BeeWhale at 57.13, RAO at 71.06, IntWPSO at 78.24, and PGA at 78.67. Across all workloads, PPMMcNE shows the fastest task scheduling, while PGA and IntWPSO consistently have the highest TSD values.

**Queue waiting time (QWT)**

Similarly, in terms of QWT, PPMMcNE consistently shows the best performance across all workloads. For 25 tasks, PPMMcNE has the lowest QWT of 43.66, followed by RAO-3 at 53.47, BeeWhale at 63.56, RAO at 77.67, IntWPSO at 84.66, and PGA at 84.83. As the workload increases to 50 tasks, PPMMcNE continues to lead with a QWT of 17.35. RAO-3 follows with 27.16, BeeWhale with 37.26, RAO with 51.4, IntWPSO with 57.99, and PGA with 58.54.For 75 tasks, PPMMcNE again has the lowest QWT at 25.81, followed by RAO-3 (35.62), BeeWhale (45.66), RAO (59.94), IntWPSO (66.98), and PGA (67.37).Overall, PPMMcNE clearly outperforms the other algorithms in both TSD and QWT, indicating that it is the most efficient for task scheduling and minimizes task queue delays across all workloads. RAO-3 and BeeWhale follow behind PPMMcNE, while RAO, IntWPSO, and PGA show higher delays and waiting times, particularly under heavier workloads.

The tables 9 and 10 provides two key metrics for evaluating the performance of virtual machine allocations—Task Migration Overhead (TMO) and Fairness Index (FI)—across workloads of 25, 50, 75, and

100 tasks for six algorithms: PGA, IntWPSO, RAO, BeeWhale, RAO-3, and RSMBO.

Table 9: TMO analysis of proposed method

| No. of Task | Task Migration Overhead (TMO) | | | | | |
|---|---|---|---|---|---|---|
| | *PGA* | *IntWPSO* | *RAO* | *BeeWhale* | *RAO-3* | *RSMBO* |
| 25 | 8200 | 8099 | 7892 | 7578 | 6568 | 5058 |
| 50 | 13173 | 13073 | 12866 | 12552 | 11542 | 10031 |
| 75 | 18182 | 18082 | 17874 | 17560 | 16550 | 15040 |
| 100 | 23178 | 23078 | 22871 | 22557 | 21547 | 20036 |

Table 10: FI analysis of proposed method

| No. of Task | Fairness Index (FI) | | | | | |
|---|---|---|---|---|---|---|
| | *PGA* | *IntWPSO* | *RAO* | *BeeWhale* | *RAO-3* | *RSMBO* |
| 25 | 0.75 | 0.76 | 0.78 | 0.81 | 0.84 | 0.95 |
| 50 | 0.73 | 0.74 | 0.76 | 0.79 | 0.81 | 0.93 |
| 75 | 0.7 | 0.72 | 0.73 | 0.75 | 0.78 | 0.91 |
| 100 | 0.68 | 0.69 | 0.71 | 0.73 | 0.75 | 0.9 |

**Task migration overhead (TMO)**

Table 9 reveals that the RSMBO algorithm consistently achieves the lowest Task Migration Overhead (TMO) across all task counts, indicating its superior ability to reduce the cost associated with migrating tasks between VMs. For example, at 25 tasks, RSMBO registers a TMO of 5058, which is markedly lower than the values recorded for the other algorithms. RAO-3 follows next with a TMO of 6568, clearly outperforming the traditional RAO, which, along with PGA and IntWPSO, exhibits higher overhead values (7892, 8200, and 8099, respectively, at 25 tasks). RAO, despite being slightly better than PGA and IntWPSO, does not match the efficiency of RAO-3. Consequently, the comparison states that RSMBO provides the best performance, RAO-3 offers a significant improvement over RAO, and that PGA and IntWPSO remain the least efficient in terms of migration overhead.

**Fairness index (FI)**

The Fairness Index (FI) assesses the fairness of resource allocation among tasks, with higher values indicating more equitable distribution. At 25 tasks, RSMBO achieves the highest fairness with an FI of 0.95, followed by RAO-3 at 0.84, BeeWhale at 0.81, RAO at 0.78, IntWPSO at 0.76, and PGA at 0.75. For 50 tasks, RSMBO maintains its high fairness with an FI of 0.93, followed by RAO-3 at 0.81, BeeWhale at 0.79, RAO at 0.76, IntWPSO at 0.74, and PGA at 0.73. At 75 tasks, RSMBO continues to lead with a FI of 0.91, RAO-3 with 0.78, BeeWhale with 0.75, RAO with 0.73, IntWPSO with 0.72, and PGA with 0.70. For FI, RSMBO again outshines the other algorithms, demonstrating superior fairness in resource allocation. RAO-3 and BeeWhale follow, while RAO, IntWPSO, and PGA exhibit lower fairness, especially as the workload increases.

The precise description ensures consistency in the analysis of both Task Migration Overhead and the Fairness Index, thereby offering a clearer understanding of the performance differences among all models evaluated. Overall, RSMBO proves to be the most efficient and fair algorithm in both metrics, making it highly effective in balancing migration overhead and equitable resource allocation.

## 4.3 Discussion

**Comparative result against state-of-the-art methods**

The table 11 illustrates that our proposed algorithms outperform existing methods on multiple fronts.

Table 11: Comparative Result against SOTA

| Algorithm | Throughput | Task Scheduling Delay (ms) | Queue Waiting Time (ms) | Energy Consumption (J/task) | TMO (25 tasks) |
|---|---|---|---|---|---|
| ***PPMMcNE*** | 0.942 | 58.22 | 43.66 | 110 | 5500 |
| ***PPE*** | 0.75 | 122.19 | 85 | 130 | 7500 |
| ***PGA*** | 0.65 | 174.78 | 90 | 140 | 8200 |
| ***IntWPSO*** | 0.6 | 292.51 | 110 | 145 | 8099 |
| ***RAO*** | 0.55 | 300 | 120 | 150 | 7892 |
| ***BeeWhale*** | 0.58 | 280 | 115 | 140 | 7578 |
| ***RAO-3*** | 0.6 | 260 | 105 | 130 | 6568 |
| ***RSMBO*** | 0.62 | 240 | 100 | 120 | 5058 |

Regarding the performance metrics, specific measures such as the Fairness Index and Task Migration Overhead (TMO) were prioritized because they directly impact the efficiency and scalability of cloud resource management. The Fairness Index is essential for ensuring that resources are equitably distributed among tasks, thereby preventing scenarios where some VMs become overloaded while others remain underutilized. This balance is crucial in dynamic cloud environments to maintain overall system stability and performance. Similarly, TMO is a critical measure because it quantifies the cost—in terms of both time and energy—associated with migrating tasks between VMs. High migration overhead can negate the benefits of optimized scheduling by introducing delays and increased energy consumption. By focusing on these metrics, our methodology not only aims to minimize basic execution times but also ensures that the system remains efficient, balanced, and scalable under varying workloads.

The statistical validation results are included in table 12 and table 13. For task scheduling, PPMcNE achieves the highest throughput (0.942), along with the lowest task scheduling delay (58.22 ms) and queue waiting time (43.66 ms). These low delay values indicate that PPMcNE rapidly assigns tasks to VMs and minimizes idle time, resulting in a much faster overall schedule compared to PPE (with 122.19 ms delay), PGA (174.78 ms delay), and IntWPSO (292.51 ms delay).

On the VM allocation side, RSMBO demonstrates significant advantages by achieving the lowest Task Migration Overhead (TMO) of 5058 and the best energy efficiency, consuming only 120 J per task on average. These metrics are considerably better than those of RAO (TMO of 7892, 150 J/task), BeeWhale (7578, 140 J/task), and RAO-3 (6568, 130 J/task). The lower TMO suggests that RSMBO minimizes the overhead incurred when migrating tasks between VMs, while its reduced energy consumption confirms that the algorithm allocates resources more efficiently.

The overall performance improvements can be attributed to the innovative integration in each proposed algorithm. PPMcNE combines Modified McNaughton's Rule with an evolutionary strategy to generate and refine an optimal initial schedule, thereby minimizing delays and increasing throughput. In parallel, RSMBO leverages Modified Brucker's Rule within a rat swarm optimization framework to balance workloads across VMs, reducing migration overhead and energy usage.

Table 12: Task scheduling performance (25 Tasks)

| ***Algorithm*** | ***Throughput (mean ± 95% CI)*** | ***Task Scheduling Delay (ms, mean ± 95% CI)*** | ***Queue Waiting Time (ms, mean ± 95% CI)*** |
|---|---|---|---|
| ***PPMMcNE*** | 0.942 ± 0.015 | 58.22 ± 4.2 | 43.66 ± 3.5 |
| ***PPE*** | 0.750 ± 0.020 | 122.19 ± 7.8 | 85.00 ± 5.0 |
| ***PGA*** | 0.650 ± 0.025 | 174.78 ± 9.1 | 90.00 ± 6.0 |
| ***IntWPSO*** | 0.600 ± 0.030 | 292.51 ± 11.3 | 110.00 ± 7.2 |

Table 13: VM allocation performance (25 Tasks)

| ***Algorithm*** | ***Energy Consumption (J/task, mean ± 95% CI)*** | ***Task Migration Overhead (TMO, mean ± 95% CI)*** |
|---|---|---|
| ***RSMBO*** | 120 ± 8 | 5058 ± 150 |
| ***RAO-3*** | 130 ± 10 | 6568 ± 200 |
| ***BeeWhale*** | 140 ± 12 | 7578 ± 180 |
| ***RAO*** | 150 ± 10 | 7892 ± 210 |

The data presented in Table 12 clearly demonstrates that the proposed PPMcNE algorithm excels in task scheduling. On the VM allocation side, Table 13 shows that the RSMBO algorithm offers substantial improvements in energy efficiency and migration overhead.

Together, these results confirm that the proposed methods—PPMcNE for task scheduling and RSMBO for VM allocation—significantly outperform their respective baselines. The consistent improvements across key performance metrics underscore the robustness and scalability of our approach, making it well-suited for dynamic, large-scale cloud environments.

# 5 Conclusion

This study introduces two novel algorithms, PPMMcNE and RSMBO, that significantly enhance task scheduling and virtual machine (VM) allocation in cloud computing environments. The PPMMcNE algorithm, which integrates Modified McNaughton's rule with an evolutionary framework, achieves a throughput of 0.942, a task scheduling delay of 58.22 ms, and a queue waiting time of 43.66 ms, outperforming traditional approaches such as PPE, PGA, and IntWPSO. Meanwhile, the RSMBO algorithm demonstrates its strength in VM allocation by minimizing task migration overhead to 5058 and achieving a high fairness index of 0.95, which collectively ensure more efficient and balanced resource utilization. These results underscore the competitive advantages of our methods under increasing workloads in dynamic cloud environments. While the performance improvements are promising, further refinement in aligning the cited literature with our discussion and elaborating on the proposed methodology is essential. Providing more detailed experimental configurations and comprehensive replication instructions will facilitate validation by other researchers. With these enhancements, our contributions have the potential to markedly advance the state-of-the-art in cloud resource management.

# References

[1] S.M. Mirmohseni, C. Tang, A. Javadpour. Using Markov learning utilization model for resource allocation in cloud of thing network. Wirel. Pers. Commun., 115, 653-677, (2020). https://doi.org/10.1007/s11277-020-07591-w

[2] Katal, S. Dahiya, T. Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. Cluster Comput., 26, 3, 1845-1875, (2023). https://doi.org/10.1007/s10586-022-03713-0

[3] Sharma, V. K., Singh, A., Jaya, K. R., Bairwa, A. K., & Srivastava, D. K. (2022). Introduction to Virtualization in Cloud Computing. Machine Learning and Optimization Models for Optimization in Cloud, 1–14. https://doi.org/10.1201/9781003185376-1

[4] Shukur, H., Zeebaree, S., Zebari, R., Zeebaree, D., Ahmed, O., & Salih, A. (2020). Cloud Computing Virtualization of Resources Allocation for Distributed Systems. Journal of Applied Science and

Technology Trends, 1(2), 98–105 . https://doi.org/10.38094/jastt1331

[5] Zolfaghari, R., Sahafi, A., Rahmani, A. M., & Rezaei, R. (2021). Application of virtual machine consolidation in cloud computing systems. Sustainable Computing: Informatics and Systems, 30, 100524.
https://doi.org/10.1016/j.suscom.2021.100524

[6] Abualigah, L., Hussein, A. M., Almomani, M. H., Zitar, R. A., Daoud, M. Sh., Migdady, H., Alzahrani, A. I., & Alwadain, A. (2024). GIJA:Enhanced geyser-inspired Jaya algorithm for task scheduling optimization in cloud computing. Transactions on Emerging Telecommunications Technologies, 35(7). Portico. https://doi.org/10.1002/ett.5019

[7] Alsubaei, F. S., Hamed, A. Y., Hassan, M. R., Mohery, M., & Elnahary, M. Kh. (2024). Machine learning approach to optimal task scheduling in cloud communication. Alexandria Engineering Journal, 89, 1–30.
https://doi.org/10.1016/j.aej.2024.01.040

[8] Sudheer Mangalampalli, S., Reddy Karri, G., Nandan Mohanty, S., Ali, S., Alamri, A. M., & Alqahtani, S. A. (2024). Efficient Hybrid DDPG Task Scheduler for HPC and HTC in Cloud Environment. IEEE Access, 12, 108897–108920. https://doi.org/10.1109/access.2024.3435914

[9] Sharma, V., & Bala, M. (2020). An Improved Task Allocation Strategy in Cloud using Modified K-means Clustering Technique. Egyptian Informatics Journal, 21(4), 201–208. https://doi.org/10.1016/j.eij.2020.02.001

[10] Abdel-Aty, A.-H., Kadry, H., Zidan, M., Al-Sbou, Y., Zanaty, E. A., & Abdel-Aty, M. (2020). A quantum classification algorithm for classification incomplete patterns based on entanglement measure. Journal of Intelligent & Fuzzy Systems, 38(3), 2809–2816.
https://doi.org/10.3233/jifs-179566

[11] Hung, T. C., Hieu, L. N., Hy, P. T., & Phi, N. X. (2019). MMSIA. Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, 60–64. https://doi.org/10.1145/3310986.3311017

[12] Alghamdi, M. I. (2022). Optimization of Load Balancing and Task Scheduling in Cloud Computing Environments Using Artificial Neural Networks-Based Binary Particle Swarm Optimization (BPSO). Sustainability, 14(19), 11982. https://doi.org/10.3390/su141911982

[13] Chandrashekar, C., Krishnadoss, P., Kedalu Poornachary, V., Ananthakrishnan, B., & Rangasamy, K. (2023). HWACOA Scheduler: Hybrid Weighted Ant Colony Optimization Algorithm for Task Scheduling in Cloud Computing. Applied Sciences, 13(6), 3433.
https://doi.org/10.3390/app13063433

[14] Vijaya, C & Srinavasan, P. (2024). Multi-objective Meta-heuristic Technique for Energy Efficient Virtual Machine Placement in Cloud Data Centers. Informatica. 48. 10.31449/inf.v48i6.5263.
https://doi.org/10.31449/inf.v48i6.5263

[15] Huang, Xinyan. (2025). Approximate SARSA Algorithm for Dimensionality-Challenged Resource Allocation Optimization in MIMO Communication Systems. Informatica. 49. 10.31449/inf.v49i8.7251.
https://doi.org/10.31449/inf.v49i8.7251

[16] Jasim, Omer K. & Salih, Basim. (2024). Improving Task Scheduling In Cloud Datacenters By Implementation Of An Intelligent Scheduling Algorithm. Informatica. 48. 10.31449/inf.v48i10.5843.
https://doi.org/10.31449/inf.v48i10.5843

[17] Ma, Jian & Zhu, Chaoyong & Fu, Yuntao & Zhang, Haichao & Xiong, Wenjing. (2024). Cloud Computing Resource Scheduling Method Based on Optimization Theory. Informatica. 48. 10.31449/inf.v48i23.6901
https://doi.org/10.31449/inf.v48i23.6901

[18] Selvapandian, R. Santosh. A Hybrid Optimized Resource Allocation Model for Multi-Cloud Environment Using Bat and Particle Swarm Optimization Algorithms. Comput. Assist. Methods Eng. Sci., 29, 1–2, 87-103, 2022.
https://doi.org/ 10.24423/cames.405

[19] Ramasamy, V., & Thalavai Pillai, S. (2020). An effective HPSO-MGA optimization algorithm for dynamic resource allocation in cloud environment. Cluster Computing, 23(3), 1711–1724. https://doi.org/10.1007/s10586-020-03118-x

[20] Shao, K., Fu, H., & Wang, B. (2023). An Efficient Combination of Genetic Algorithm and Particle Swarm Optimization for Scheduling Data-Intensive Tasks in Heterogeneous Cloud Computing. Electronics, 12(16), 3450. https://doi.org/10.3390/electronics12163450

[21] Hafsi, H., Gharsellaoui, H., & Bouamama, S. (2022). Genetically-modified Multi-objective Particle Swarm Optimization approach for high-performance computing workflow scheduling. Applied Soft Computing, 122, 108791. https://doi.org/10.1016/j.asoc.2022.108791

[22] Alfakih, T., Hassan, M. M., & Al-Razgan, M. (2021). Multi-Objective Accelerated Particle Swarm Optimization With Dynamic Programing Technique for Resource Allocation in Mobile Edge Computing. IEEE Access, 9, 167503–167520. https://doi.org/10.1109/access.2021.3134941

[23] Mirmohseni, S. M., Javadpour, A., & Tang, C. (2021). LBPSGORA: Create Load Balancing with Particle Swarm Genetic Optimization Algorithm to Improve Resource Allocation and Energy Consumption in Clouds Networks. Mathematical Problems in Engineering, 2021, 1–15. https://doi.org/10.1155/2021/5575129

[24] Pirozmand, P., Jalalinejad, H., Hosseinabadi, A. A. R., Mirkamali, S., & Li, Y. (2023). An improved particle swarm optimization algorithm for task scheduling in cloud computing. Journal of Ambient Intelligence and Humanized Computing, 14(4), 4313–4327.
https://doi.org/10.1007/s12652-023-04541-9

[25] Yadav, M., & Mishra, A. (2023). An enhanced ordinal optimization with lower scheduling overhead based novel approach for task scheduling in cloud computing environment. Journal of Cloud Computing, 12(1). https://doi.org/10.1186/s13677-023-00392-z

[26] Saravanan, G., Neelakandan, S., Ezhumalai, P., & Maurya, S. (2023). Improved wild horse optimization with levy flight algorithm for effective task scheduling in cloud computing. Journal of Cloud Computing, 12(1). https://doi.org/10.1186/s13677-023-00401-1

[27] Badri, S., Alghazzawi, D. M., Hasan, S. H., Alfayez, F., Hasan, S. H., Rahman, M., & Bhatia, S. (2023). An Efficient and Secure Model Using Adaptive Optimal Deep Learning for Task Scheduling in Cloud Computing. Electronics, 12(6), 1441. https://doi.org/10.3390/electronics12061441

[28] Chaudhary, S., Sharma, V. K., Thakur, R. N., Rathi, A., Kumar, P., & Sharma, S. (2023). Modified

Particle Swarm Optimization Based on Aging Leaders and Challengers Model for Task Scheduling in Cloud Computing. Mathematical Problems in Engineering, 2023(1). Portico. https://doi.org/10.1155/2023/3916735

[29] Y. Hamed, A., Kh. Elnahary, M., S. Alsubaei, F., & H. El-Sayed, H. (2023). Optimization Task Scheduling Using Cooperation Search Algorithm for Heterogeneous Cloud Computing Systems. Computers, Materials & Continua, 74(1), 2133–2148. https://doi.org/10.32604/cmc.2023.032215

[30] Y. Hamed, A., & H. Alkinani, M. (2021). Task Scheduling Optimization in Cloud Computing Based on Genetic Algorithms. Computers, Materials & Continua, 69(3), 3289–3301. https://doi.org/10.32604/cmc.2021.018658

[31] Huang, X., Li, C., Chen, H., & An, D. (2019). Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. Cluster Computing, 23(2), 1137–1147. https://doi.org/10.1007/s10586-019-02983-5

[32] Younes, A., Kh. Elnahary, M., H. Alkinani, M., & H. El-Sayed, H. (2022). Task Scheduling Optimization in Cloud Computing by Rao Algorithm. Computers, Materials & Continua, 72(3), 4339–4356. https://doi.org/10.32604/cmc.2022.022824N.

[33] Manikandan, N., Gobalakrishnan, N., & Pradeep, K. (2022). Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment. Computer Communications, 187, 35–44. https://doi.org/10.1016/j.comcom.2022.01.016

[34] Fayed, A. R., Khalifa, N. E. M., Taha, M. H. N., & Kotb, A. (2023). Optimization of Task Scheduling in Cloud Computing Using the RAO-3 Algorithm. The 3rd International Conference on Artificial Intelligence and Computer Vision (AICV2023), March 5–7, 2023, 508–523. https://doi.org/10.1007/978-3-031-27762-7_47