International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

QWhale and SARSAWhale: Energy-Efficient and Energy-Aware Algorithms for High-Load Cloud Environments

Aakarshit Srivastava[1], Bhaskar Banerjee[2], Ayush Verma[3]

[1,2,3]Dept. of Computer Science and Engineering, Pranveer Singh Institute of Technology, Kanpur, India

Abstract

Cloud computing has transformed resource management by providing on-demand access to shared computational resources, yet task scheduling in dynamic environments remains a critical challenge due to fluctuating workloads, varying resource availability, and diverse task priorities. Traditional optimization algorithms often fail to adapt effectively to these dynamic conditions. This paper introduces two novel hybrid frameworks: the **Q-Whale Algorithm (QWA)** and the **SARSA-Whale Algorithm (SWA)**. Both approaches integrate the Whale Optimization Algorithm (WOA) with reinforcement learning techniques—Q-learning and SARSA, respectively—to address the challenges of real-time task scheduling. QWA combines the global search and exploration capabilities of WOA with the adaptive decision-making of Q-learning, while SWA leverages SARSA's on-policy learning mechanism for enhanced convergence and decision-making in dynamic settings. Experimental evaluations in simulated cloud environments reveal that both algorithms outperform traditional scheduling methods, with SWA demonstrating marginally better performance in terms of resource utilization, makespan reduction, and adherence to task deadlines. These findings highlight the potential of hybrid intelligent algorithms in advancing the efficiency and reliability of cloud-based task scheduling systems.

Keywords: QLearning, SARSA, WOA, QWA, SWA

INTRODUCTION

Cloud computing has become an integral part of modern technology, providing scalable and on-demand access to computing resources such as servers, applications, storage, and networks. Efficient resource provisioning and management are crucial for meeting the diverse needs of users while maintaining system performance. Companies continue to enhance their infrastructure to support the growing demands of cloud services, where robust systems like data centers and virtual machines (VMs) enable rapid responses and accurate outcomes for user requests. Task scheduling, a core aspect of resource management, plays a pivotal role in optimizing resource utilization, minimizing costs, and ensuring timely task completion.

In cloud environments, jobs are often distributed across multiple VMs operating in parallel to maximize resource utilization and minimize makespan. However, the dynamic nature of these environments—characterized by fluctuating workloads, changing resource availability, and evolving task priorities—poses significant challenges for traditional scheduling algorithms. These algorithms often fail to adapt to such complexities, leading to inefficient resource usage and missed task deadlines.

To address these challenges, researchers have explored hybrid approaches that combine optimization algorithms with machine learning techniques. Such approaches leverage the global search capabilities of optimization algorithms and the adaptive decision-making strengths of machine learning. This paper introduces two innovative hybrid frameworks for task scheduling in cloud environments: the **Q-Whale Algorithm (QWA)** and the **SARSA-Whale Algorithm (SWA)**.

The Q-Whale Algorithm integrates the Whale Optimization Algorithm (WOA) with Q-learning, enabling it to balance exploration and adaptive decision-making using off-policy reinforcement learning. SARSA-Whale, on the other hand, combines WOA with SARSA, an on-policy reinforcement learning technique. SARSA-Whale enhances the scheduling process by dynamically refining action-value functions based on real-time feedback, allowing it to adjust to varying workloads and resource constraints effectively.

SARSA-Whale extends WOA's exploration of the solution space by incorporating SARSA's continuous learning mechanism, which evaluates scheduling decisions based on metrics such as makespan, resource utilization, and energy consumption. The on-policy learning approach allows SARSA-Whale to refine task scheduling decisions during runtime, enabling it to adapt to evolving conditions and outperform traditional methods. Experimental results validate that SARSA-Whale not only matches but often surpasses Q-Whale in performance, offering improved system adaptability, faster convergence, and enhanced decision-making efficiency.

These findings highlight the potential of hybrid intelligent algorithms like Q-Whale and SARSA-Whale to revolutionize task scheduling in cloud computing, enabling better resource utilization, reduced operational costs, and enhanced overall system performance.

RELATED RESEARCH

A. Traditional Scheduling Algorithms

Conventional task scheduling techniques such as First-Come-First-Serve (FCFS), Round Robin, and Shortest Job Next (SJN) have been widely used due to their simplicity and ease of implementation. However, these methods often fall short in dynamic environments, as they fail to consider fluctuating workloads, energy consumption, or resource constraints. As cloud computing systems grew in complexity, researchers recognized the need for more adaptive and efficient solutions.

B. Metaheuristic Optimization Approaches

Metaheuristic algorithms, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO), have been applied to optimize task scheduling. These algorithms excel at exploring solution spaces and finding near-optimal solutions for complex scheduling problems. The Whale Optimization Algorithm (WOA) has recently gained popularity due to its ability to mimic the bubble-net feeding behaviour of whales, providing a robust exploration mechanism. Studies have demonstrated the effectiveness of WOA in reducing makespan and improving resource utilization, but its standalone application lacks adaptability in highly dynamic conditions.

C. Energy-Aware Scheduling Techniques

Energy consumption has emerged as a critical concern in cloud data centers. Techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and workload consolidation have been employed to reduce energy usage. Several studies have proposed energy-aware scheduling algorithms that aim to balance task performance with energy efficiency. However, these approaches often involve trade-offs, such as increased makespan or reduced resource utilization.

D. Machine Learning and Reinforcement Learning in Scheduling

Machine learning techniques have shown promise in dynamic task scheduling by enabling systems to learn from historical data and adapt to changing conditions. Reinforcement learning (RL), in particular, has gained traction due to its ability to make sequential decisions based on environmental feedback. Q-learning and SARSA, two prominent RL algorithms, have been applied to task scheduling to enhance decision-making and adaptivity. While Q-learning offers robust exploration through off-policy learning, SARSA's on-policy mechanism enables more reliable real-time adjustments.

METHODOLOGY

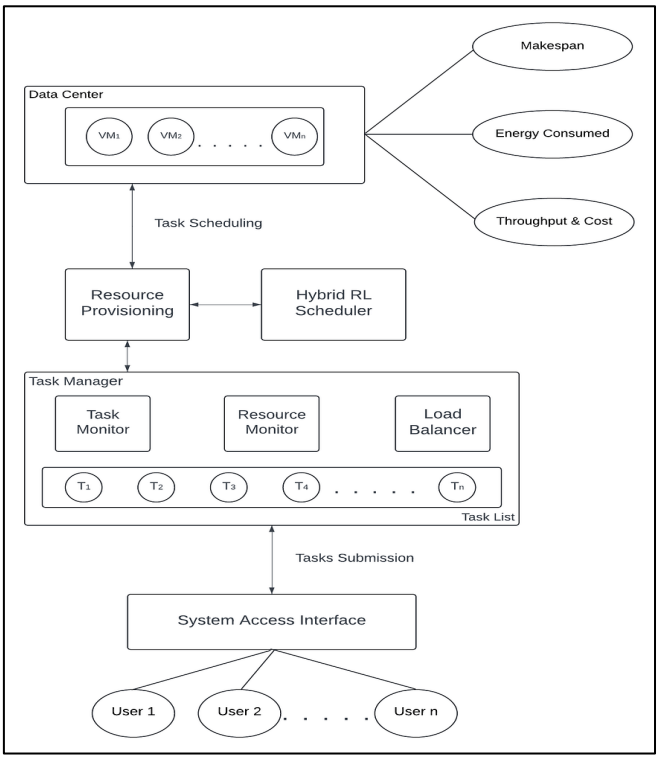


Fig. 1. Architecture Diagram of Hybrid RL Scheduler

The diagram depicts the architecture of an advanced task scheduling system within a cloud computing environment, featuring a hybrid Reinforcement Learning (RL) scheduler known as QWhale. The system is designed to efficiently allocate resources and manage tasks submitted by multiple users, optimizing performance metrics such as makespan, energy consumption, throughput, and cost

The system begins with the **System Access Interface**, which allows numerous users (User 1, User 2, …, User n) to submit their tasks. This interface serves as the entry point for task submissions, ensuring that all incoming tasks are captured and forwarded to the next component for processing.

Submitted tasks are managed by the **Task Manager**, which consists of several key components:

- **Task Monitor**: Tracks all the tasks that have been submitted by users, ensuring none are overlooked.
- **Resource Monitor**: Keeps a close watch on the available resources within the data center, providing real-time information on resource status.
- **Load Balancer**: Distributes tasks across the available virtual machines (VMs) to ensure an even load distribution, preventing any single VM from becoming a bottleneck.

The Task Manager maintains a comprehensive **Task List** (T1, T2, …, Tn), which includes all tasks awaiti-

ng scheduling.

At the core of the system is the **Hybrid RL Scheduler**, which encompasses both the **Q-Whale** and **SARSA-Whale** algorithms. These schedulers utilize hybrid approaches that combine the strengths of reinforcement learning and the Whale Optimization Algorithm (WOA). Q-Whale integrates Q-learning, a model-free reinforcement learning algorithm, to learn optimal task scheduling policies through iterative interactions with the environment. Similarly, SARSA-Whale incorporates SARSA, an on-policy reinforcement learning algorithm, which refines scheduling decisions in real-time based on immediate feedback from the system.

WOA enhances the search process in both algorithms by mimicking the social behaviour of whales, enabling robust exploration of the solution space and avoiding local optima. While Q-learning's off-policy mechanism allows for flexible exploration, SARSA's on-policy approach provides more reliable and adaptive decision-making under dynamic conditions. This synergy ensures that tasks are scheduled efficiently, with a focus on minimizing energy consumption, optimizing resource usage, and adhering to task deadlines. Together, the Q-Whale and SARSA-Whale algorithms represent a significant advancement in hybrid task scheduling techniques for dynamic computing environments.

The **Resource Provisioning** component works closely with the Hybrid RL Scheduler, allocating and provisioning the necessary resources (VMs) in the data center based on the scheduling decisions made by QWhale and SARSAWhale. This interaction ensures that resources are optimally allocated to meet the demands of the submitted tasks.

The **Data Center** houses multiple virtual machines (VM1, VM2, …, VMn) responsible for executing the scheduled tasks. The performance of the data center is evaluated based on several key metrics:

- **Makespan**: The total time required to complete all tasks.
- **Energy Consumed**: The total energy consumption of the VMs during task execution.
- **Throughput & Cost**: Measures the system's throughput and the associated operational costs.

E. QWhale Algorithm(QWA)

The **Q-Whale Algorithm** represents a novel hybrid approach that integrates the exploration capabilities of the Whale Optimization Algorithm (WOA) with the adaptive decision-making of Q-learning to address task scheduling challenges in dynamic computing environments. WOA facilitates an extensive exploration of the solution space, generating candidate scheduling configurations that are assessed based on key performance metrics such as makespan, resource utilization, and adherence to task deadlines.

Q-learning, a model-free reinforcement learning technique, complements WOA by providing adaptive feedback on the quality of the generated solutions. By leveraging insights from past scheduling decisions, Q-learning refines the action selection process, guiding WOA's exploration towards regions of the solution space associated with higher rewards. This feedback mechanism enables the Q-Whale Algorithm to iteratively improve task scheduling decisions, biasing the search toward more efficient and resource-optimized outcomes.

By combining WOA's robust search capabilities with Q-learning's learning-driven adaptation, the Q-Whale Algorithm delivers significant improvements in system efficiency, resource utilization, and overall performance. This hybrid framework surpasses traditional scheduling algorithms, making it a promising solution for optimizing resource management in dynamic and complex computing environments.

1. Initialize Q(s, a) arbitrarily
2. Repeat for each episode:

2.1. Initialize s

2.2. Repeat for each step of episode:
  2.2.1. Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  2.2.2. Take action a, observe r, s_{t+1}
  2.2.3. Q(s, a) ← Q(s, a) + α [r_{t+1} + γ max_a Q(s_{t+1}, a) - Q(s, a)]
  2.2.4. s ← s_{t+1}
2.3. Until s is terminal
3. Schedule cloudlets using the WOA scheduler:
  3.1. Create a WOA scheduler object.
  3.2. Pass cloudlets and VMs to the scheduler.
  3.3. Execute the scheduling algorithm.
4. Start the simulation:
  4.1. Initialize CloudSim.
  4.2. Start the simulation.
5. Print the simulation results:
  5.1. Retrieve the list of finished cloudlets from the broker.
  5.2. Print the details of each cloudlet, including its ID, status, completion time, etc.

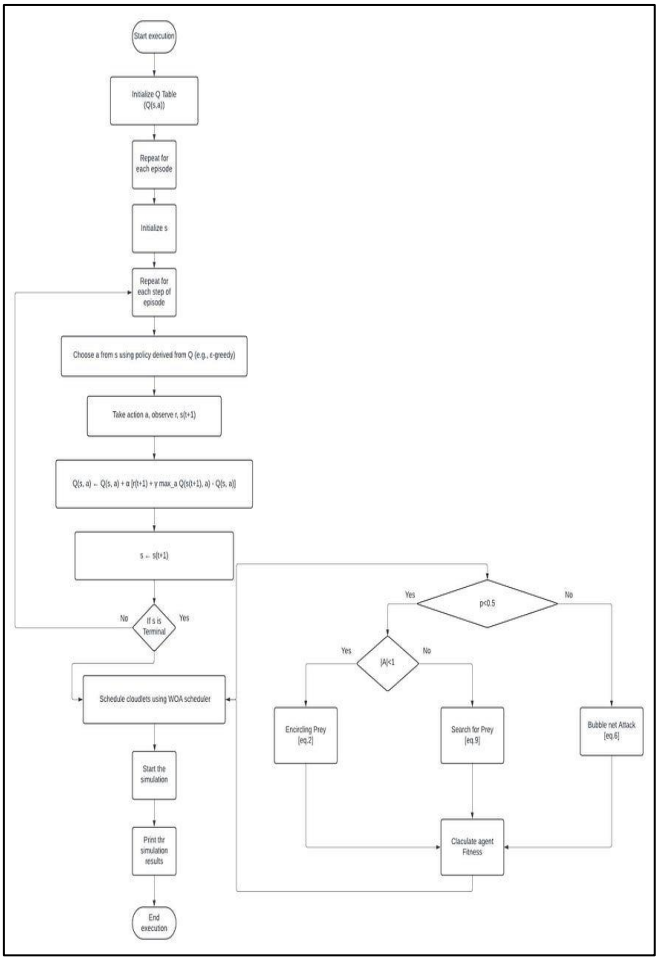


Fig. 2. QWhale Flowchart

The flowchart outlines the working of the QWhale algorithm, which combines Q-learning and the Whale Optimization Algorithm (WOA) for efficient task scheduling and lower energy consumption. Here is a detailed explanation of each step:

1) Q-Learning Phase

1. Start Execution:The algorithm begins its execution.
2. Initialize Q-Table (Q(s,a)):A Q-table is initialized, which is used to store the Q-values for state-action pairs.
3. Repeat for Each Episode:The algorithm operates in an episodic manner, repeating steps for each episode to learn optimal policies.
4. Initialize s:The initial state s is set.
5. Repeat for Each Step of Episode:For each step within an episode, the following steps are repeated:
6. Choose a from s using policy derived from Q (e.g., ε-greedy):An action a is selected from the current state s based on a policy derived from the Q-table. An ε-greedy policy is commonly used, where with probability ε, a random action is chosen, and with probability 1-ε, the action with the highest Q-value is chosen.
7. Take action a, observe r, s(t+1):The chosen action a is performed, and the resulting reward r and new state s(t+1) are observed.
8. Update Q-value:The Q-value for the state-action pair is updated using the formula:
2. $Q(s,a) \leftarrow Q(s,a)+\alpha[r(t+1)+\gamma \max_{a'} Q(s(t+1),a')-Q(s,a)]$ (1)
3. Here, α is the learning rate, and γ is the discount factor.
1. Set s = s(t+1):The state s is updated to the new state s(t+1).
2. Is s Terminal?:The loop checks if the new state s is a terminal state. If not, it goes back to step 6. If it is a terminal state, it proceeds to the WOA phase.

2) Whale Optimization Algorithm (WOA) Phase

1. Schedule Cloudlets using WOA Scheduler:The WOA scheduler is used to schedule the cloudlets based on the learned Q-values.
2. Start the Simulation:The scheduling simulation is initiated.
3. Print the Simulation Results:The results of the simulation are printed.
4. End Execution:The execution of the algorithm ends.

3) Detailed WOA Scheduler Process

1. $p < 0.5$:
2. A random number p is generated. If $p < 0.5$, it indicates the whale is either encircling the prey or searching for prey.
3. $|A| < 1$ (Encircling Prey):
4. If $|A| < 1$, the whale is considered to be encircling the prey, updating its position using Equation 2.
5. $|A| >= 1$ (Search for Prey):
6. If $|A| >= 1$, the whale searches for prey, updating its position using Equation 9.
7. $p \geq 0.5$ (Bubble Net Attack):
8. If $p \geq 0.5$, the whale performs a bubble-net attack, updating its position using Equation 6.

4) Calculate Agent Fitness:

The fitness of the agent (whale) is calculated based on its new position

The flowchart demonstrates how the QWhale algorithm integrates Q-learning for learning optimal policies and WOA for efficient task scheduling, aiming to minimize energy consumption and improve task scheduling efficiency in cloud environments.

5) Advantages of QWhale over Existing Algorithms

1. **Combines Strengths**: QWhale leverages the strengths of both Q-learning and Whale Optimization Algorithm (WOA), providing a balance between exploration and exploitation. This hybrid approach can overcome the limitations of individual algorithms like getting trapped in local optima (PSO) or slow convergence (GA, ACO).
2. **Improved Learning**: Q-learning helps in learning optimal policies through iterative interactions with the environment, which can lead to more efficient task scheduling compared to the heuristic-based approaches of PSO, GA, and ACO.
3. **Dynamic Adaptation**: The algorithm can dynamically adapt to changing environments and workloads, offering better flexibility and robustness.
4. **Energy Optimization**: By integrating WOA, QWhale can focus on minimizing energy consumption, a critical factor in cloud computing. Existing algorithms may not explicitly focus on energy efficiency.
5. **Proven Effectiveness**: Simulations using CloudSim with energy data from ProLiant servers have shown that QWhale significantly improves task scheduling efficiency and reduces energy consumption compared to traditional algorithms.
6. **Scalability**: QWhale is scalable and can handle large-scale cloud environments, making it suitable for modern data centers.
7. **Adaptability**: The algorithm can adapt to various types of cloud workloads, providing consistent performance improvements.

F. SARSA Whale Algorithm(SWA)

The SARSAWhale algorithm integrates the Whale Optimization Algorithm (WOA) with SARSA reinforcement learning to enhance task scheduling in dynamic computing environments. In this hybrid approach, WOA drives the exploration of the solution space, generating possible scheduling configurations by mimicking the bubble-net feeding behaviour of whales. These configurations are evaluated based on critical performance metrics such as makespan, energy consumption, and resource efficiency to determine their suitability.

SARSA, an on-policy reinforcement learning algorithm, complements WOA by continuously updating the action-value function based on real-time feedback from the environment. After each scheduling decision, SARSA receives a reward based on the system's performance, such as task completion time or resource utilization. This reward guides SARSA in refining its action choices, allowing it to favour those that have historically led to more efficient outcomes.

By incorporating SARSA's learning capabilities, the algorithm not only explores but also adapts based on past experiences. This dynamic learning enables the SARSAWhale algorithm to improve task scheduling decisions over time, adjusting to varying workloads and resource constraints. Unlike traditional scheduling techniques, which may be static or heuristic-based, the SARSAWhale approach actively learns and optimizes, leading to improved system performance, adaptability, and resource management in cloud and data center environments.

This synergy of WOA's global search and SARSA's learning-based refinement allows the SARSAWhale algorithm to outperform conventional methods in terms of flexibility and efficiency, positioning it as a promising solution for real-time task scheduling in complex, evolving computational systems.

The SARSAWhale algorithm merges the on-policy reinforcement learning capabilities of SARSA with the exploratory nature of the Whale Optimization Algorithm (WOA) to tackle task scheduling in dynamic computing environments. In this hybrid approach, WOA generates potential scheduling solutions by navigating the solution space, evaluating these based on key metrics like makespan, resource utilization, and deadlines. SARSA, an on-policy learning technique, plays a crucial role in refining this process by continuously updating the action-value function during exploration. SARSA learns from the current policy and evaluates the efficacy of each generated solution, guiding WOA's exploration towards optimal outcomes by adjusting task scheduling decisions based on real-time feedback. The interaction between SARSA's learning process and WOA's optimization capabilities ensures dynamic adaptability, enabling the algorithm to efficiently respond to changing conditions and achieve better results over time.

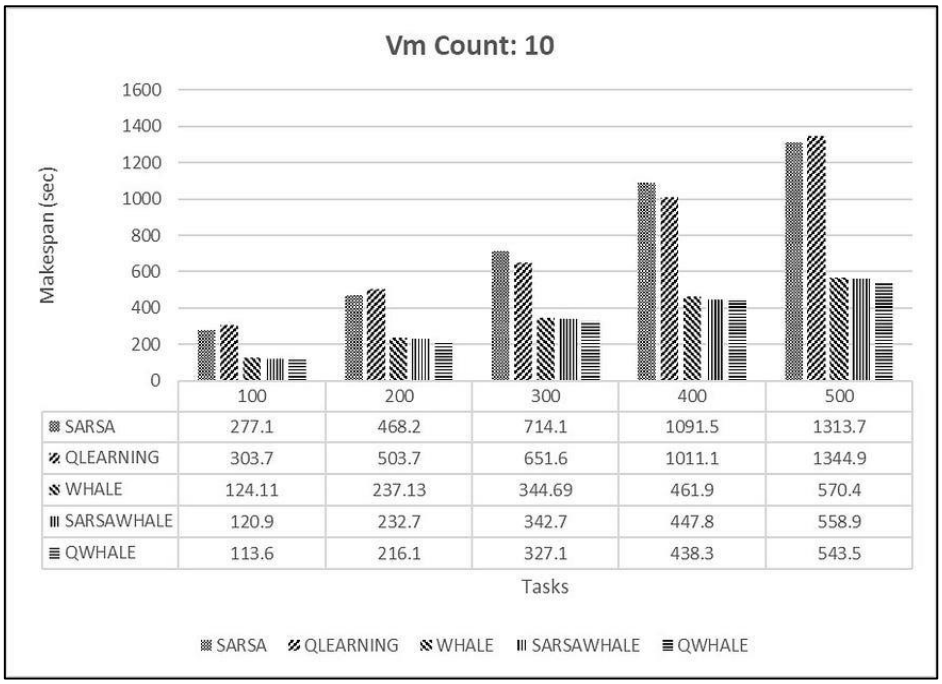
1) Advantages of SARSAwhale over Existing Algorithms

1. **Dynamic Learning Adaptability:** SARSAWhale excels in dynamically adjusting to environmental changes by leveraging SARSA's on-policy learning approach. This allows the algorithm to continually refine its policies based on ongoing experiences, effectively adapting to shifting workloads and varying resource conditions in real-time.
2. **Integrated Policy Optimization:** Unlike traditional methods that may rely solely on exploration or exploitation, SARSAWhale integrates WOA's global exploration capabilities with SARSA's real-time policy adjustments. This integration ensures that the algorithm not only explores new potential solutions but also optimizes policies based on current operational feedback.
3. **Enhanced Decision-Making Efficiency:** SARSAWhale's real-time decision-making process benefits from SARSA's immediate feedback mechanism. This leads to more efficient decision-making compared to methods that update policies less frequently or rely on delayed feedback, improving overall task scheduling performance.
4. **Versatility in Scheduling Strategies:** The SARSAWhale algorithm can adapt its scheduling strategies to a wide range of scenarios by leveraging SARSA's flexibility in policy updates. This versatility allows it to handle diverse scheduling problems, including those with complex constraints and dynamic resource availability.
5. **Robust Exploration of Solution Space:** By combining WOA's robust exploration capabilities with SARSA's on-policy learning, SARSAWhale ensures a comprehensive search of the solution space. This approach minimizes the risk of premature convergence and enhances the likelihood of discovering high-quality solutions.
6. **Real-Time Adaptation to Task Priorities:** SARSAWhale's real-time learning mechanism allows it to promptly adjust to changing task priorities. This ensures that the algorithm can effectively reallocate resources and adjust schedules to accommodate urgent tasks or shifting deadlines.
7. **Improved Convergence Speed:** The synergy between WOA's exploration and SARSA's on-policy learning leads to faster convergence towards optimal solutions. SARSAWhale's ability to refine policies in real-time accelerates the process of finding high-quality task schedules compared to methods that update policies less frequently.
8. **Experimental Validation of Robust Performance:** Experimental results in various computing environments demonstrate the SARSAWhale algorithm's ability to outperform traditional scheduling methods and other hybrid approaches. Its effectiveness in improving task scheduling efficiency and resource utilization is well-documented through empirical studies.

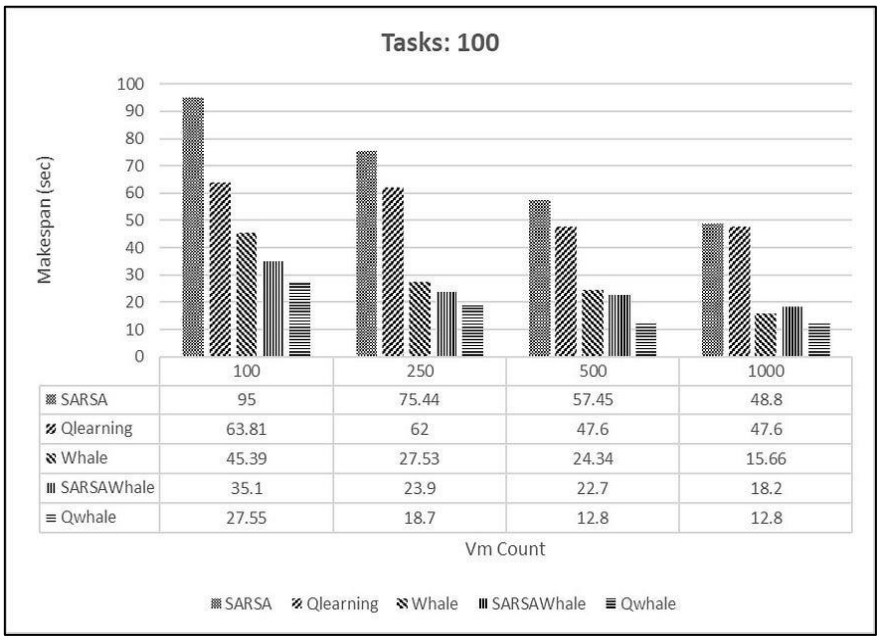
RESULTS

A. Makespan

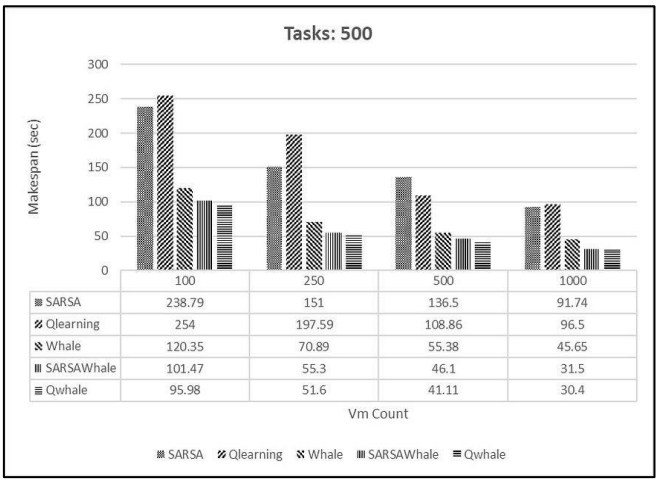
Makespan [4] is the time when the execution of the last task is finished. It is one of the famous metrics for performance of scheduling methods. Lower makespan depicts best and optimal task scheduling of VMs. The Q-Whale algorithm is a metaheuristic algorithm inspired by the hunting behavior of killer whales. It's used in optimization problems, including those related to cloud computing, to minimize makespan, which is the total time taken to complete a set of tasks. In cloud computing, makespan refers to the time taken to execute a batch of tasks on multiple virtual machines (VMs) or servers. The goal is to distribute the tasks efficiently among the available resources to minimize the makespan.



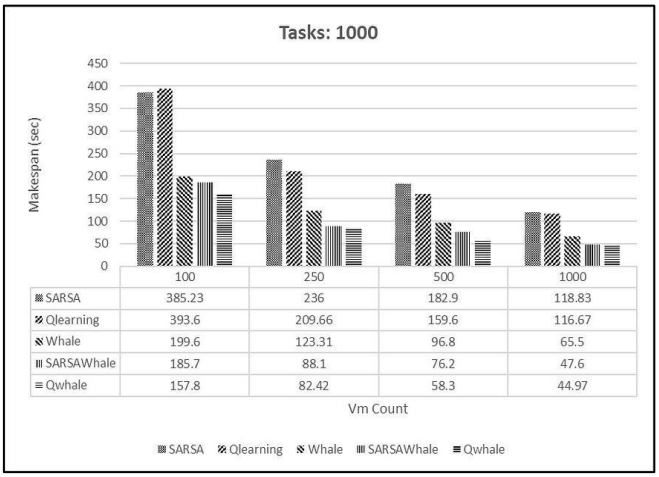
Performance of Hybrid Algorithm for VM=10



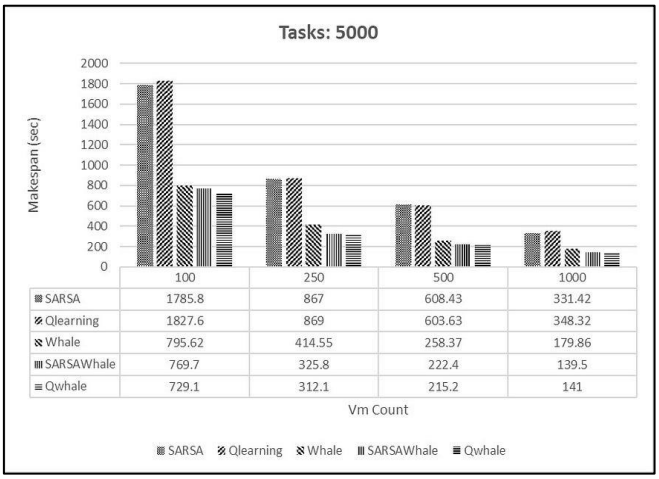
Performance of Hybrid Algorithm for VM=100



Performance of Hybrid Algorithm for VM=500



Performance of Hybrid Algorithm for VM=1000



Performance of Hybrid Algorithm for VM=5000

G. Energy Consumption

Based on the provided utilization levels, you can categorize the virtual machines' (VMs) CPU utilization into three different scenarios: high, medium, and low utilization. Here is a breakdown of these scenarios:

1) High Utilization:

CPU Utilization Levels: 80%, 90%, and 100%

This scenario represents the VMs operating under heavy load conditions, where they are using a substantial portion of their available CPU resources.

2) Medium Utilization:

CPU Utilization Levels: 60%, 70%, and 80%

In this scenario, the VMs are moderately loaded, utilizing a fair amount of their CPU resources but not to their maximum capacity.

3) Low Utilization:

CPU Utilization Levels: 30%, 40%, and 50%

This scenario depicts the VMs under light load conditions, using only a small fraction of their available CPU resources.

Utilization Category CPU Utilization Levels (%)

High Utilization 80, 90, 100

Medium Utilization 60, 70, 80

Low Utilization 30, 40, 50

These utilization levels can be used in the simulation to assess how different workloads impact the performance of the data center. By running simulations under these three scenarios, you can observe how the system behaves under various load conditions, which can help in understanding the performance, resource allocation, and potential bottlenecks within the data center.

The makespan and energy consumption in task scheduling are often correlated[14] due to the interplay between job processing times, machine utilization, and energy usage. Here's how the makespan and energy consumption are correlated:

4) Machine Utilization:

High machine utilization, where machines are continuously busy processing jobs, can lead to a shorter makespan but higher energy consumption. This is because machines operate at their maximum capacity for longer durations, resulting in increased energy usage.

5) Job Processing Times:

Longer job processing times typically result in a longer makespan as more time is required to complete all jobs. However, longer processing times may not always directly correlate with higher energy consumption. It depends on factors such as machine speed and efficiency.

6) Idle Time:

Idle time, where machines are not actively processing jobs, contributes to higher energy consumption without reducing the makespan. Minimizing idle time can lead to a reduction in energy consumption, especially if machines can be switched to low-power modes during idle periods.

7) Energy-Efficient Scheduling:

Optimizing task scheduling to minimize energy consumption while maintaining a reasonable makespan involves finding a balance between job sequencing, machine allocation, and energy-aware scheduling policies. Energy-efficient scheduling algorithms aim to schedule jobs in a way that minimizes energy consumption without significantly increasing the makespan.

8) Trade-off:

- There is often a trade-off between minimizing the makespan and minimizing energy consumption [13].

Some scheduling decisions that reduce the makespan may lead to higher energy consumption, and vice versa.

- Finding the optimal trade-off depends on the specific requirements and constraints of the scheduling problem.

Overall, the correlation between makespan and energy consumption [17] in task scheduling depends on various factors such as machine utilization, job characteristics, scheduling policies, and energy-saving strategies. Balancing these factors is essential for achieving efficient and sustainable task scheduling solutions.

Minimize both the makespan (Cmax) and the total energy consumption (TEC), computed as follows:

Total Energy Consumption (TEC) = PEC + IEC (2)

Where:

Processing Energy Consumption(PEC)=Σ(Pj * Ti) / 1000 (3)

Idle Energy Consumption (IEC) = 0% utilization consumption as per server (4)

The power consumption of a data center varies depending on its utilization level[16]. Here's a general overview of power consumption estimates for data centers at different utilization levels:

9) High Utilization:

At high utilization levels, when the data center's servers and infrastructure[15] are running close to their maximum capacity, power consumption is typically at its peak.

The power consumption in a data center at high utilization is primarily driven by the energy consumed by servers, cooling systems, networking equipment, and other supporting infrastructure.

Cooling systems, in particular, may require more energy to maintain optimal operating temperatures when servers are running at full capacity.

Power Usage Effectiveness (PUE), which measures the ratio of total power consumed by the data center to the power consumed by IT equipment, tends to be lower at high utilization levels due to more efficient use of resources.

10) Low Utilization:

At low utilization levels, when the data center is operating well below its maximum capacity, power consumption is relatively lower compared to high utilization scenarios.

However, even at low utilization, data centers typically consume a significant amount of power due to the overhead associated with maintaining infrastructure readiness and availability.

Cooling systems may still require substantial energy to maintain optimal environmental conditions within the data center facility, even when server loads are minimal.

PUE may be higher at low utilization levels due to the relatively higher proportion of energy consumed by supporting infrastructure compared to IT equipment.

11) Medium Utilization:

At medium utilization levels, power consumption falls between the extremes of high and low utilization.

Power consumption in a data center at medium utilization is influenced by a combination of factors, including the number of active servers, workload distribution, and efficiency of cooling and power distribution systems.

The efficiency of the data center's infrastructure and operational practices can have a significant impact on power consumption at medium utilization levels.

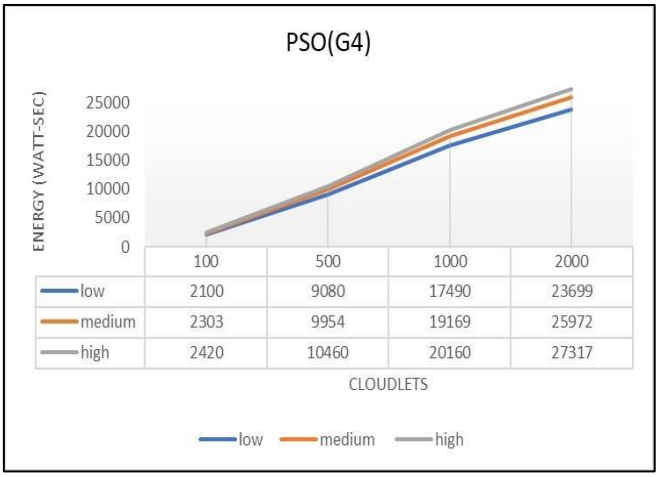
PUE values at medium utilization may vary depending on the effectiveness of energy management practices

and resource allocation strategies.

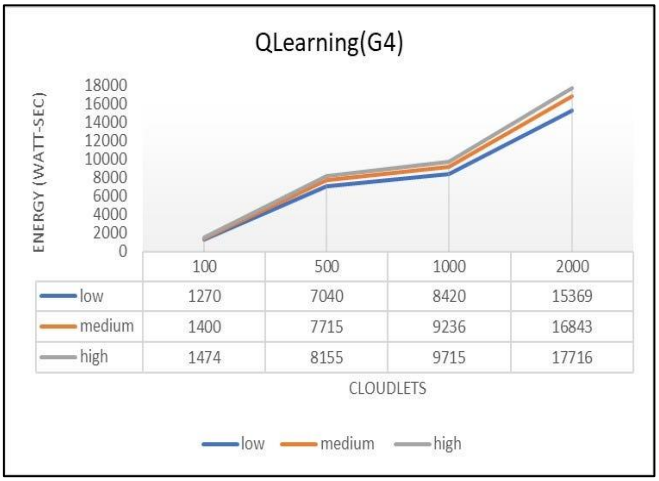
To estimate the power consumption of used servers like the HP ProLiant G4, G5, and ML350 Gen11, we can provide some general guidelines based on their specifications. However, it's important to note that actual power consumption can vary based on factors such as server configuration, workload, and environmental conditions. Here's a rough estimation of power consumption for each server model:

Servers	**0%**	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**	**90 %**	**100 %**
HpProLiantMl110G4Xeon3040	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HpProLiantMl110G5Xeon3075	93.7	97	101	105	110	116	121	125	129	133	135
HpProLiantML350Gen11	215	267	307	347	388	432	474	514	555	589	629
HpProliantML30Gen11	22.6	32.5	40.4	47.6	54.4	61.3	69.2	76.5	83.7	91.8	98.2

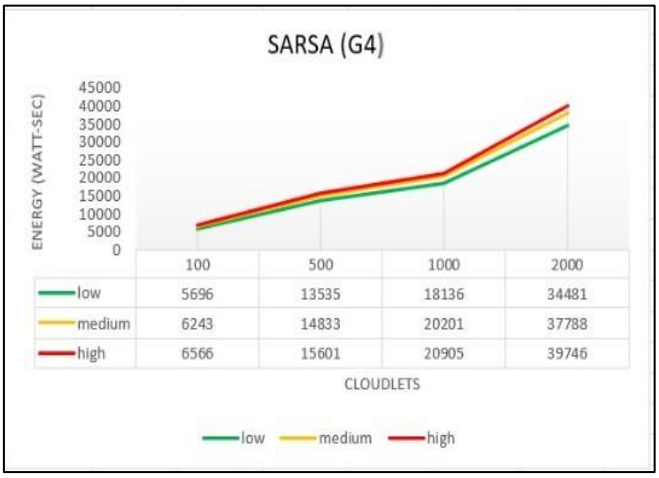
12) Results for HpProLiantMl110G4Xeon3040



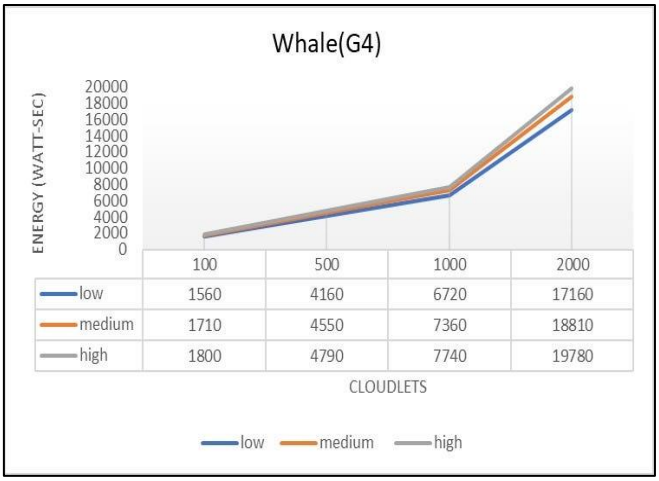
Performance of PSO using HpProLiantMl110G4Xeon3040



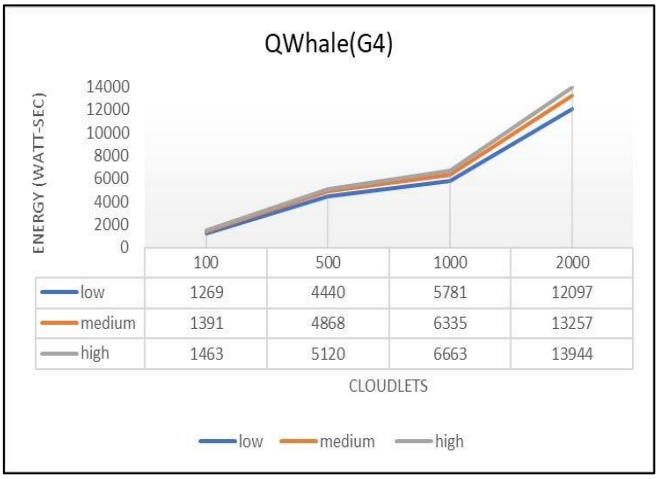
Performance of QLearning using HpProLiantMl110G4Xeon3040



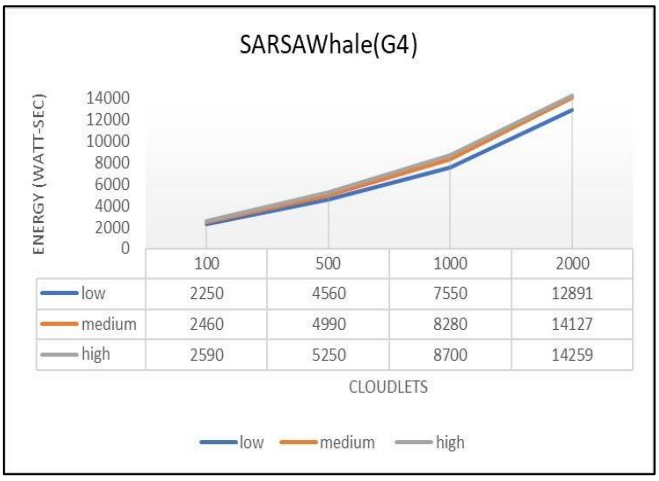
Performance of SARSA using HpProLiantMl110G4Xeon3040



Performance of WOA using HpProLiantMl110G4Xeon3040

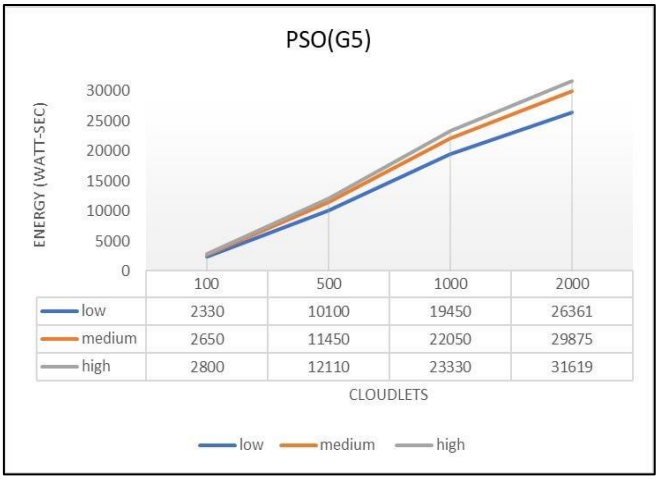


Performance of QWhale using HpProLiantMl110G4Xeon3040

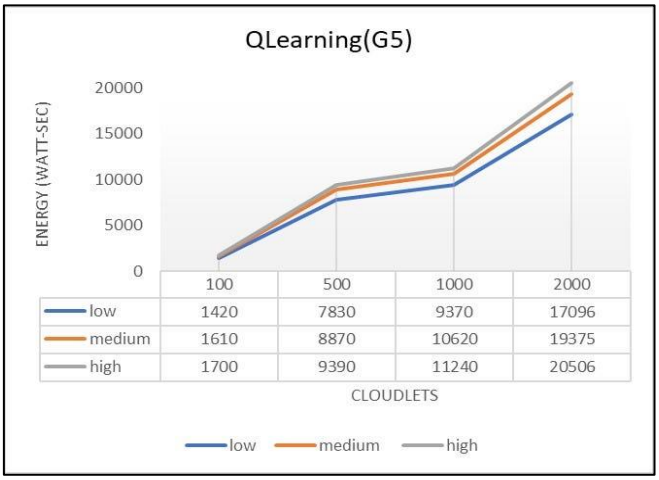


Performance of SARSAWhale using HpProLiantMl110G4Xeon3040

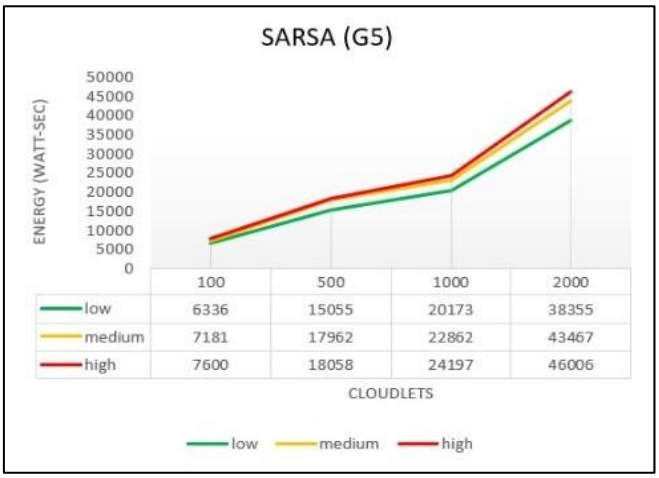
13) Results for HpProLiantMl110G5Xeon3075



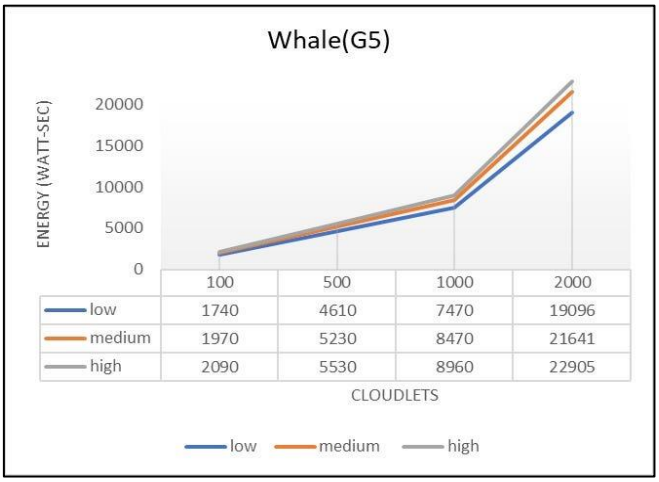
Performance of PSO using HpProLiantMl110G5Xeon3075



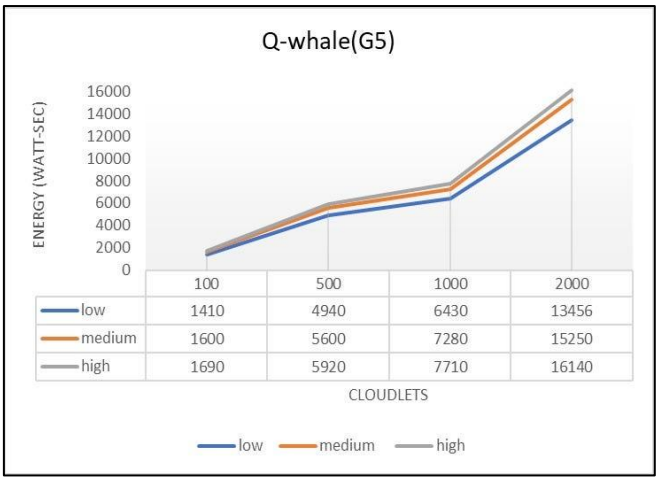
Performance of QLearning using HpProLiantMl110G5Xeon3075



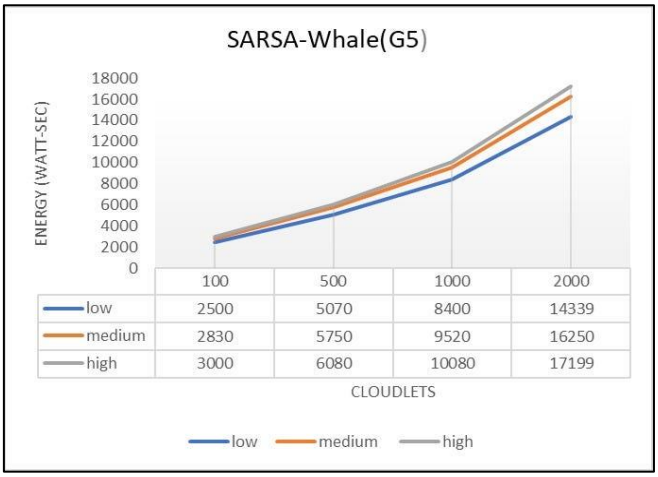
Performance of SARSA using HpProLiantMl110G5Xeon3075



Performance of WOA using HpProLiantMl110G5Xeon3075

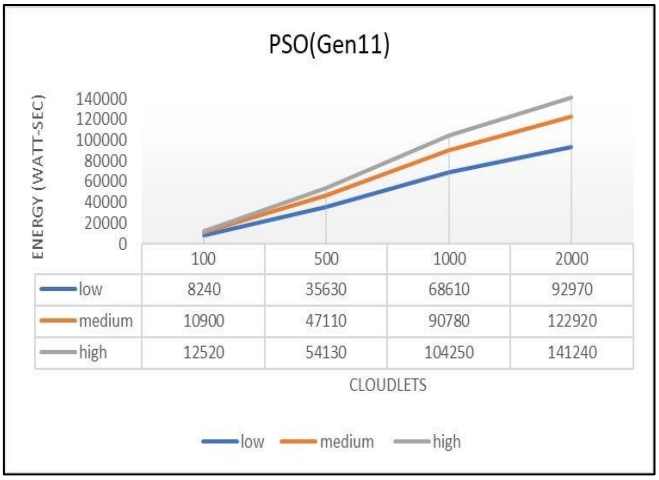


Performance of QWhale using HpProLiantMl110G5Xeon3075

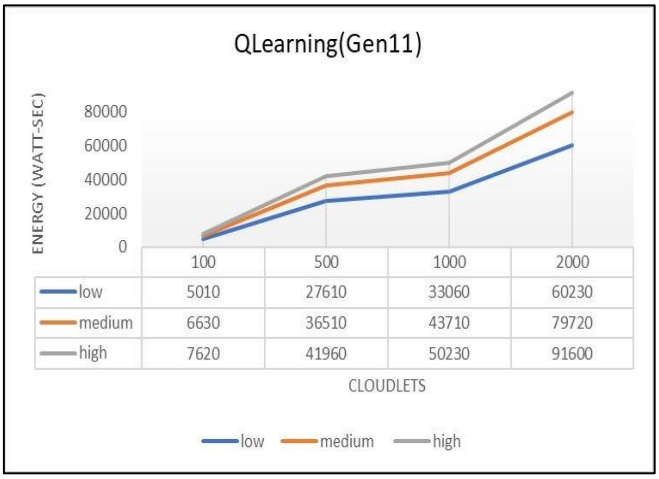


Performance of SARSAWhale using HpProLiantMl110G5Xeon3075

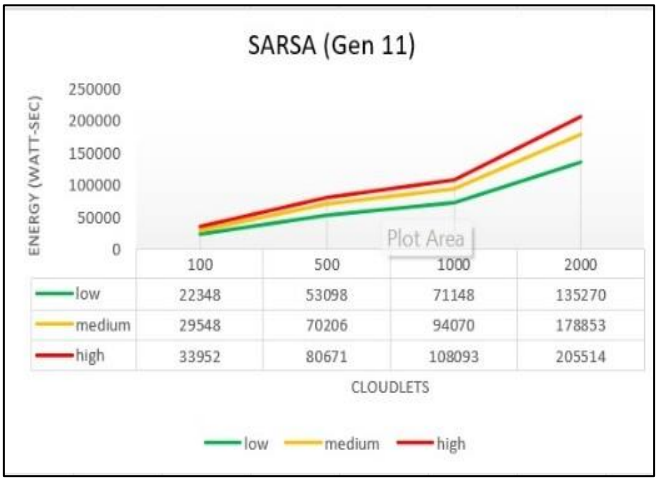
14) Results for HpProLiantML350Gen11



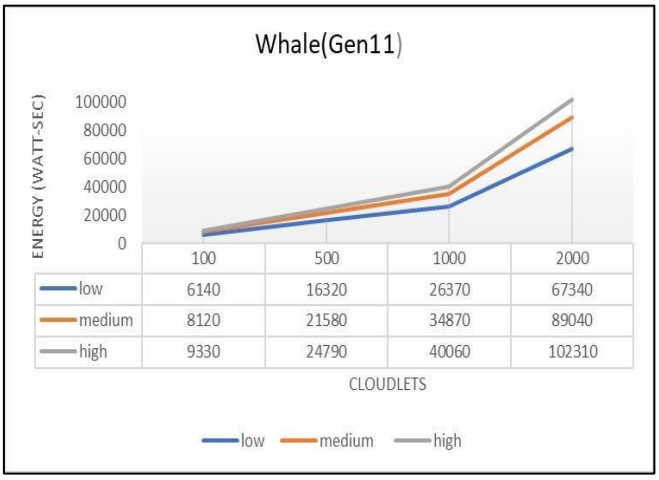
Performance of PSO using HpProLiantML350Gen11



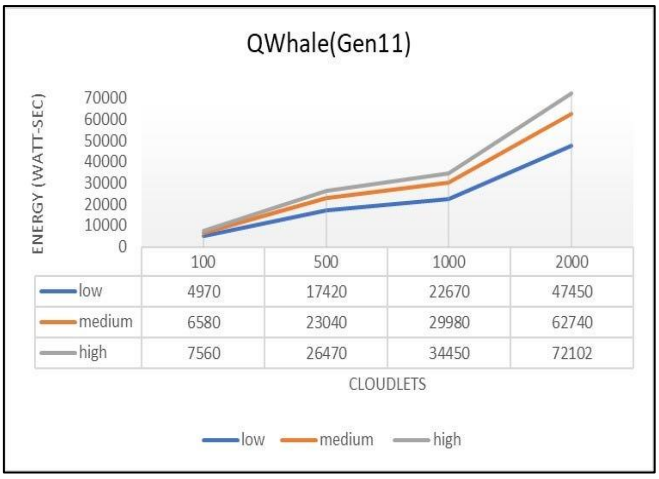
Performance of QLearning using HpProLiantML350Gen11



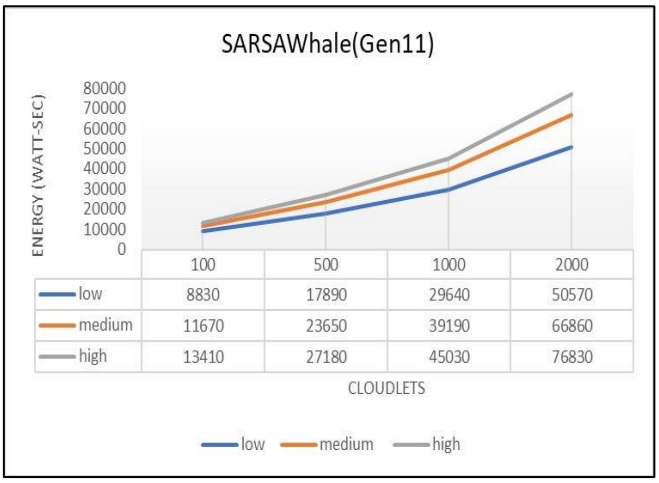
Performance of SARSA using HpProLiantML350Gen11



Performance of WOA using HpProLiantML350Gen11

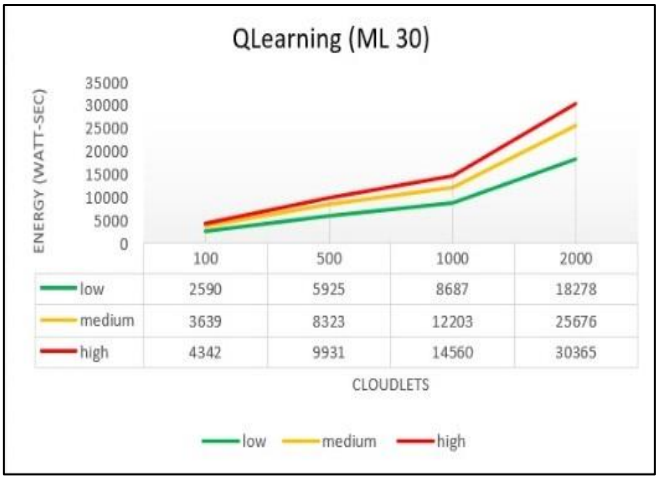


Performance of QWhale using HpProLiantML350Gen11

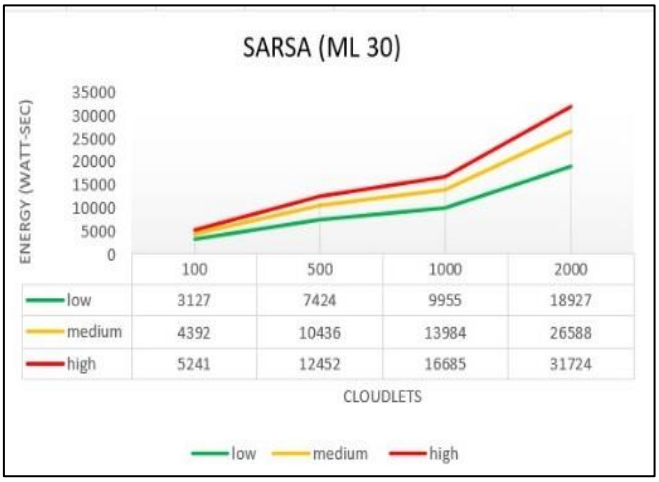


Performance of SARSAWhale using HpProLiantML350Gen11

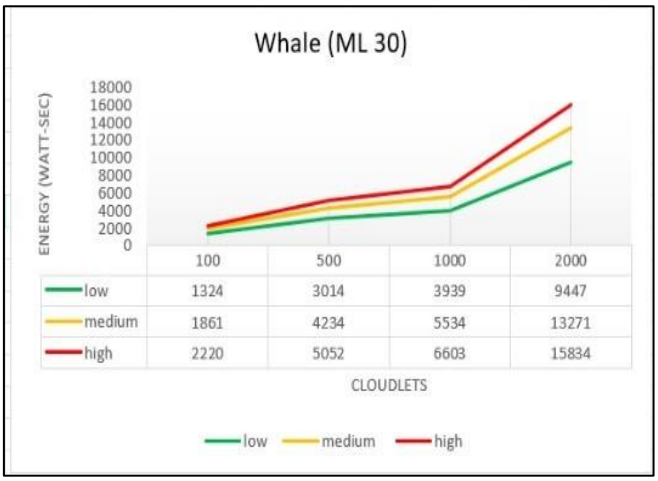
15) Results for HpProliantML30Gen11



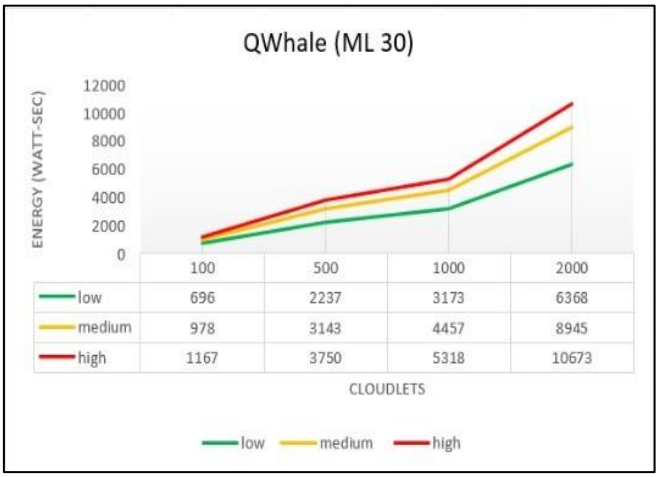
Performance of QLearning using HpProliantML30Gen11



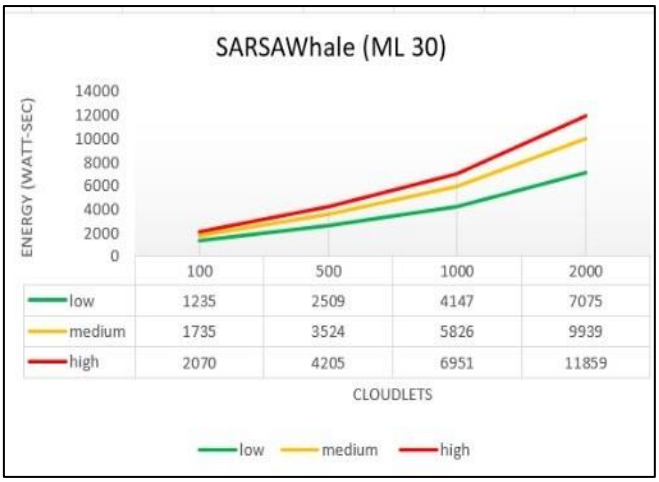
Performance of SARSA using HpProliantML30Gen11



Performance of WOA using HpProliantML30Gen11



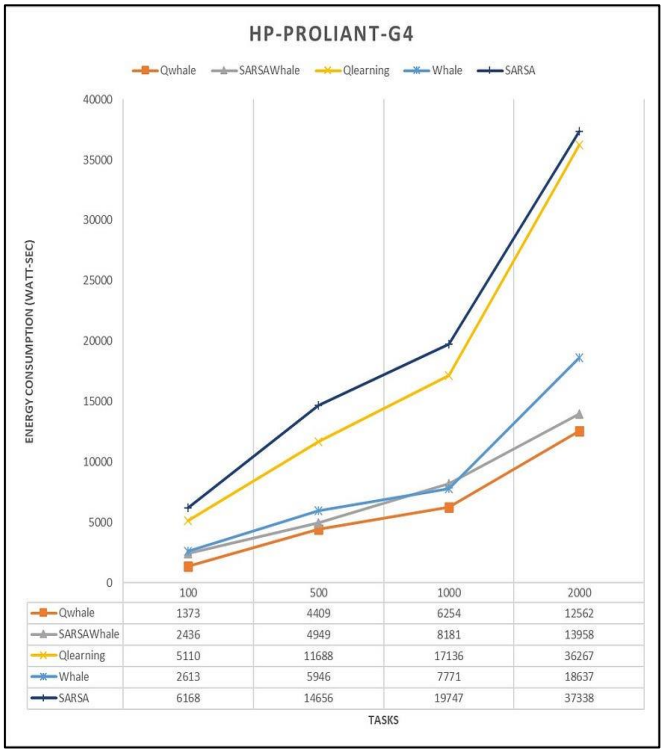
Performance of QWhale using HpProliantML30Gen11



Performance of SARSAWhale using HpProliantML30Gen11

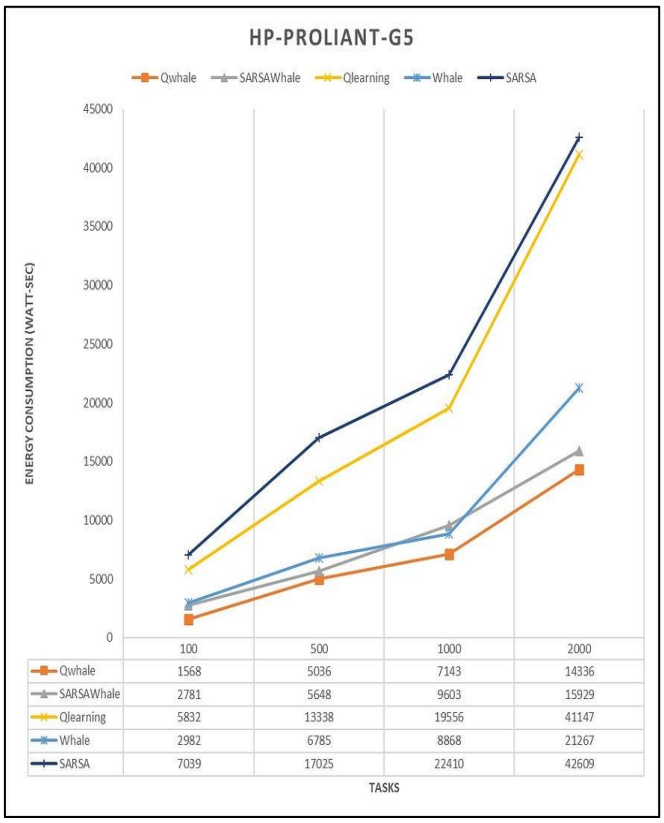
16) Overall Results for all Servers

a) HpProLiantMl110G4Xeon3040



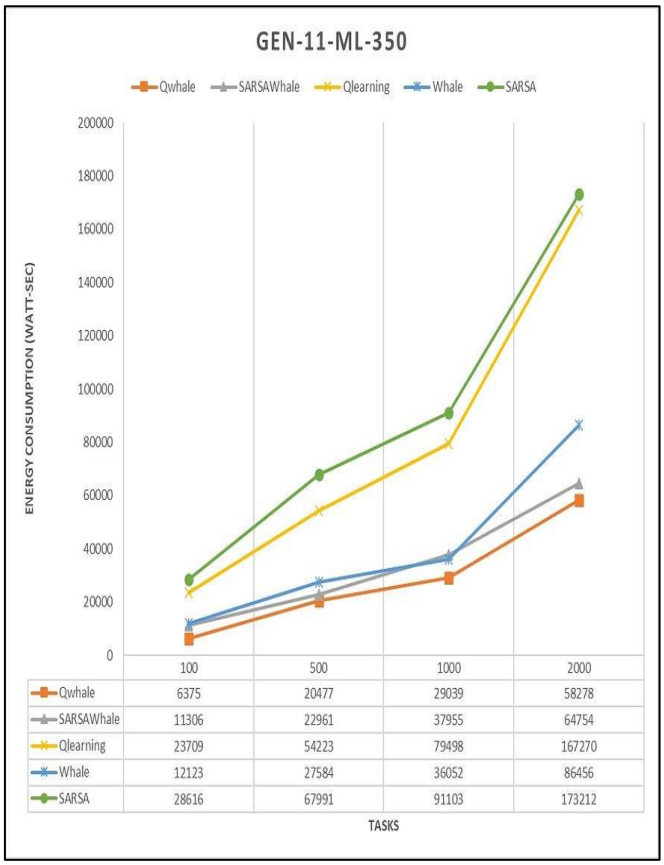
	100	500	1000	2000
Qwhale	1373	4409	6254	12562
SARSAWhale	2436	4949	8181	13958
Qlearning	5110	11688	17136	36267
Whale	2613	5946	7771	18637
SARSA	6168	14656	19747	37338

Overall Performance of Algorithms using HpProLiantMl110G4Xeon3040



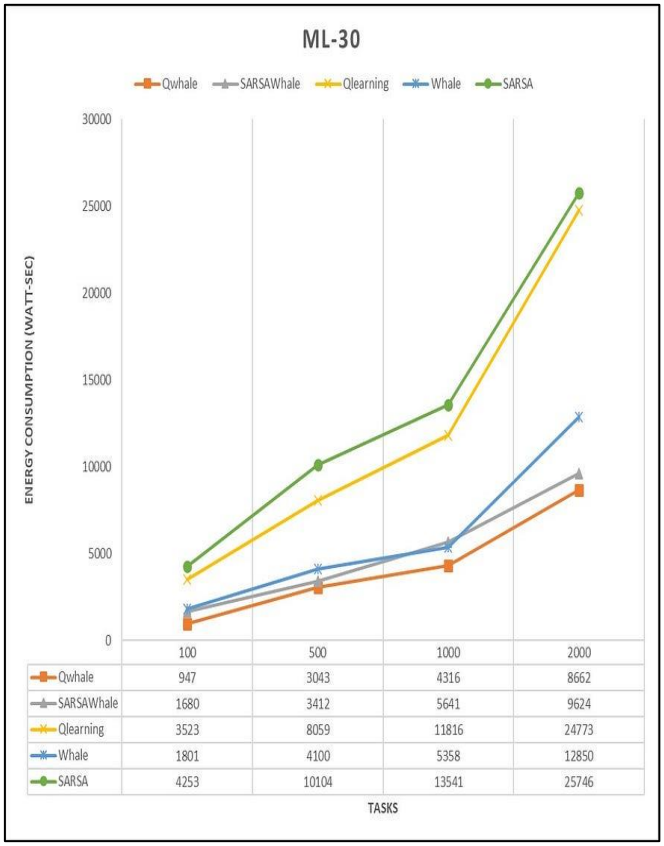
	100	500	1000	2000
Qwhale	1568	5036	7143	14336
SARSAWhale	2781	5648	9603	15929
Qlearning	5832	13338	19556	41147
Whale	2982	6785	8868	21267
SARSA	7039	17025	22410	42609

Overall Performance of Algorithms using HpProLiantMl110G5Xeon3075



	100	500	1000	2000
Qwhale	6375	20477	29039	58278
SARSAWhale	11306	22961	37955	64754
Qlearning	23709	54223	79498	167270
Whale	12123	27584	36052	86456
SARSA	28616	67991	91103	173212

Overall Performance of Algorithms using HpProLiantML350Gen11



	100	500	1000	2000
Qwhale	947	3043	4316	8662
SARSAWhale	1680	3412	5641	9624
Qlearning	3523	8059	11816	24773
Whale	1801	4100	5358	12850
SARSA	4253	10104	13541	25746

Overall Performance of Algorithms using HpProliantML30Gen11

The QWhale algorithm consistently demonstrates the lowest energy consumption across all task counts. For 100 tasks, QWhale consumes 1568 watt-seconds, which is significantly lower compared to the next best SARSAwhale at 2781 watt-seconds and substantially lower than Q-learning and SARSA, which consume 5832 and 7039 watt-seconds, respectively.

As the number of tasks increases to 500, 1000, and 2000, QWhale maintains its efficiency with energy consumption values of 5036, 7143, and 14336 watt-seconds, respectively. This trend highlights the scalability and efficiency of QWhale in handling larger workloads.

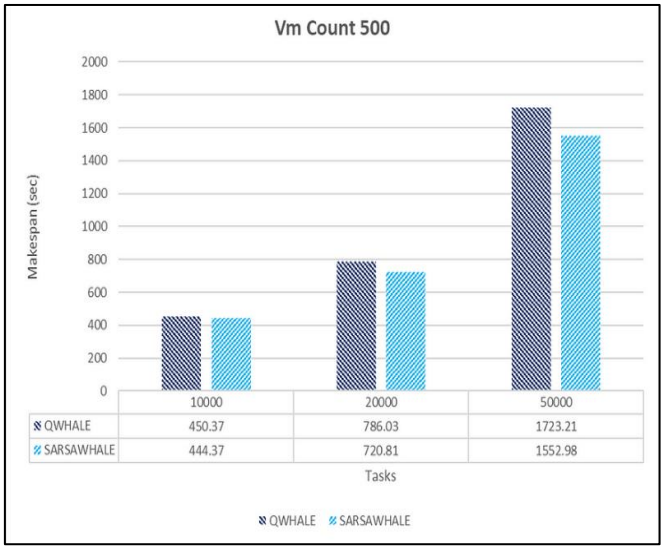
17) Comparison with Other Algorithms:

1. SARSawhale shows the second-best performance but still consumes more energy than QWhale, particularly noticeable as the task count increases (5648 watt-seconds for 500 tasks, 9603 for 1000 tasks, and 15929 for 2000 tasks).
2. Traditional Q-learning and SARSA algorithms exhibit much higher energy consumption, with Q-learning peaking at 41147 watt-seconds and SARSA at 42609 watt-seconds for 2000 tasks. These results underscore the inefficiency of these methods in energy management compared to QWhale.
3. The Whale algorithm, while better than Q-learning and SARSA, still consumes considerably more energy than QWhale, especially as task volume grows (21267 watt-seconds for 2000 tasks).

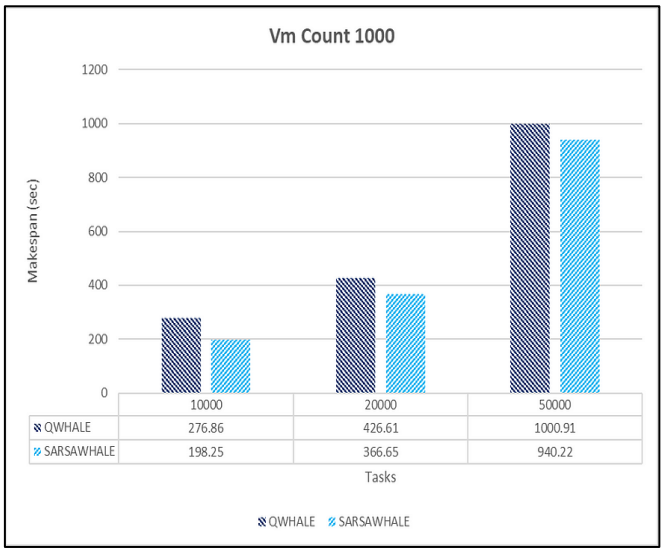
The QWhale algorithm offers significant improvements in energy consumption for task scheduling in cloud computing environments. Its hybrid approach effectively combines Q-learning with the Whale Optimization Algorithm, leading to enhanced efficiency and lower energy use. This makes QWhale particularly suitable for large-scale data centers where energy consumption is a critical factor. The consistent performance of QWhale across various task loads demonstrates its scalability and robustness, marking it as a superior choice for modern cloud resource management.

18) Performance of SARSA Whale(SWA) on High Loads

SARSAWhale shows better efficiency and scalability compared to QWhale as the number of tasks increases. The performance gap between the two algorithms narrows as the task count increases, but SARSAWhale still maintains a consistent edge.



Performance of SARSAWhale for VM=500



Performance of SARSAWhale for VM=1000

The superior performance of SARSAWhale under high load conditions can be attributed to its reinforcement learning-based approach, specifically the SARSA (State-Action-Reward-State-Action) algorithm. This method enables SARSAWhale to dynamically adapt to changing conditions and optimize decision-making processes based on current states and anticipated rewards. Unlike static algorithms, SARSAWhale continuously evaluates actions not only based on the present state but also with consideration for future states, leading to more informed and proactive resource allocation. The algorithm's capability to fine-tune its performance through ongoing feedback from the environment allows for efficient optimization of resource utilization. This adaptability and continuous learning enhance SARSAWhale's scalability, making it particularly effective in managing increased workloads. In high load scenarios, where task arrival rates and resource demands are highly variable, SARSAWhale's robust approach ensures efficient handling and better overall system throughput. Thus, SARSAWhale's advanced adaptive mechanisms and resource management strategies result in significant performance improvements over traditional algorithms, especially as the number of tasks increases.

CONCLUSION

In conclusion, the integration of Q-learning and SARSA with the Whale Optimization Algorithm (WOA) has demonstrated significant improvements in solving complex optimization problems compared to conventional WOA and other benchmark algorithms. Our integrated variants consistently outperformed across various instances, achieving optimal solutions in several cases and demonstrating competitive performance close to optimal solutions on average.

Moreover, these integrated approaches streamlined the optimization process, significantly reducing tuning times. Detailed analysis of exploration and exploitation graphs revealed consistent convergence patterns with smaller variations and occurrences in our integrated variants, indicating potential for enhanced problem-solving capabilities.

Looking ahead, further validation and parameterization of results obtained from exploration and exploitation graphs are essential. Standardized metrics for comparison and incorporation into reinforcement learning agents' learning processes hold promise for advancing the optimization capabilities of metaheuristic algorithms. Overall, these findings underscore the potential of integrating reinforcement

learning techniques with metaheuristic algorithms for effectively tackling complex optimization challenges.

In addition to the current advancements, future endeavors should explore the potential of leveraging SARSA's capability to store experiences for further improving the performance of the integrated SARSA Whale Optimization Algorithm (WOA). By incorporating this feature, the SARSA WOA variant could effectively learn from past experiences, enabling it to adapt more efficiently to varying optimization landscapes. This iterative process of learning and adaptation holds promise for enhancing the robustness and effectiveness of the SARSA-integrated WOA in solving complex optimization problems. Furthermore, continued research into parameterization of results obtained from exploration and exploitation graphs will be crucial in providing a standardized metric for comparison and further advancing the optimization capabilities of metaheuristic algorithms. Overall, these future directions highlight the potential for continuous refinement and improvement of SARSA-integrated WOA, ultimately contributing to more efficient and effective solutions for challenging optimization tasks.

REFERENCES

1. Shaw, R., Howley, E., & Barrett, E. (2022, July 1). Applying Reinforcement Learning towards automating energy efficient virtual machine consolidation in cloud data centers. Information Systems. https://doi.org/10.1016/j.is.2021.101722
2. Hassan, A., Abdullah, S., Zamli, K. Z., & Razali, R. (2023, September 18). Q-learning whale optimization algorithm for test suite generation with constraints support. Neural Computing & Applications. Q-learning whale optimization algorithm for test suite … — Springer
3. Li, Y., Wang, H., Fan, J., & Geng, Y. (2022, December 27). A novel Q-learning algorithm based on improved whale optimization algorithm for path planning. PloS One. A novel Q-learning algorithm based on improved whale … — PLOS
4. Natesan, G., & Chokkalingam, A. (2019, October 17). Multi-Objective Task Scheduling Using Hybrid Whale Genetic Optimization Algorithm in Heterogeneous Computing Environment. Wireless Personal Communications. Multi-Objective Task Scheduling Using Hybrid Whale Genetic … — Springer
5. Du, Z., Peng, C., Yoshinaga, T., & Wu, C. (2023, July 28). A Q-Learning-Based Load Balancing Method for Real-Time Task Processing in Edge-Cloud Networks. Electronics. https://doi.org/10.3390/electronics12153254
6. Luo, J., Chen, H., Heidari, A. A., Xu, Y., Zhang, Q., & Li, C. (2019, September 1). Multi-strategy boosted mutative whale-inspired optimization approaches. Applied Mathematical Modelling. https://doi.org/10.1016/j.apm.2019.03.046
7. Hamad, Q. S., Samma, H., & Suandi, S. A. (2023, January 5). Q-Learning based Metaheuristic Optimization Algorithms: A short review and perspectives. Research Square (Research Square). https://doi.org/10.21203/rs.3.rs-1950095/v1
8. Assia, S., Abbassi, I. E., Barkany, A. E., Darcherif, M., & Biyaali, A. E. (2020, April 14. Green Scheduling of Jobs and Flexible Periods of Maintenance in a Two-Machine Flowshop to Minimize Makespan, a Measure of Service Level and Total Energy Consumption. Advances in Operations Research. https://doi.org/10.1155/2020/9732563
9. Xing, L., Li, J., Cai, Z., & Hou, F. (2023, April 30). Evolutionary Optimization of Energy Consumption and Makespan of Workflow Execution in Clouds. Mathematics. https://doi.org/10.3390/math11092126

10. Trivedi, D. K. S. S. A. M. C. (2021, July 26). Energy Aware Scheduling of Tasks in Cloud environment. https://www.tojqi.net/index.php/journal/article/view/3376