Performance Enhancement of Cloud Resource Management by Using Optimized Reinforcement Learning Approaches

Gaurav Bajpai[1*], Pawan Singh[2], Abhay Kumar Agarwal[3]

[1]Department of Computer Science and Engineering, Amity University, Lucknow Campus, India, Email: gaurav.bajpai@s.amity.edu
[2]Department of Computer Science and Engineering, Amity University Uttar Pradesh, Lucknow Campus, India
[3]Department of Computer Science and Engineering Kamla Nehru Institute of Technology, Sultanpur, India
*Corresponding Author

Received: 17.04.2024 Revised : 18.05.2024 Accepted: 20.05.2024

ABSTRACT
Cloud computing provides on-demand access to a shared pool of specially configured computing resources. Scheduling tasks in these dynamic environments is challenging due to fluctuating workload demands, varying resource availability, and differing task priorities. Traditional optimization algorithms often fail to adapt effectively to these conditions. In this paper, we introduce the Q-Whale algorithm (QWA) and SARSA-Whale algorithm (SWA), a novel hybrid approach that integrates the Whale Optimization Algorithm (WOA) with Reinforcement learning techniques to address these challenges. The Q-Whale algorithm combines WOA's exploration capabilities with Q-learning's adaptive decision-making to optimize task scheduling in real time. Our experiments in dynamic computing environments show that the hybrid algorithm enhances resource utilization, reduces makespan, and meets task deadlines more effectively than traditional methods.

Keywords: Virtual Machines, Reource Management, Quality of Service, Genetic Algorithm, Hybrid Reinforcment Learning (HRL)

INTRODUCTION

Cloud computing offers various services on demand, allowing access to computing resources such as servers, applications, services, storage, and networks. According to the National Institute of Standards and Technology (NIST), these resources must be efficiently provisioned and released with minimal effort from brokers or service providers to meet the diverse demands of users. Companies are continually upgrading their infrastructure to keep pace with the rapid development and increasing service requirements of cloud computing. Every request in a cloud environment needs to be processed quickly and accurately, necessitating high-performance computing devices like data centers and virtual machines (VMs). Recent research in cloud computing has focused on optimizing resource utilization, improving task scheduling, enhancing cloud security, reducing costs, and boosting overall cloud performance[12,17].

Cloud service providers, especially those offering Infrastructure as a Service (IAAS), are responsible for distributing high-performing resources to data centers. To maximize resource utilization, jobs are assigned to multiple VMs that operate in parallel. In a virtual environment, task assignments to VMs must consider success rates, costs, processing times, and makespan. Consequently, extensive research has been conducted on quality of service (QoS) parameters to optimize cloud resource scheduling. Parallel scheduling of VMs aims to minimize makespan by optimizing processing times while using minimal resources. This approach ensures that tasks are completed efficiently and effectively, maximizing the overall performance of the cloud environment. Task scheduling is crucial for optimizing resource utilization and performance in dynamic computing environments, such as cloud computing systems. The dynamic nature of these environments[5]—with fluctuating workload demands, varying resource availability, and evolving task priorities—creates significant challenges for traditional task scheduling algorithms. These challenges often result in suboptimal solutions, inefficient resource utilization, and missed task deadlines.

To address these issues, researchers have been exploring hybrid approaches that integrate optimization algorithms with machine learning techniques. These hybrid methods aim to harness the strengths of both types of algorithms: optimization algorithms excel at exploring large solution spaces, while machine learning techniques provide adaptive decision-making capabilities in dynamic environments[5]. By

combining these approaches, researchers seek to develop more robust and efficient task scheduling solutions that can better handle the complexities of modern computing environments.

BACKGROUND

In this subsection, we highlight the known algorithms and terminologies available in the domain. They are as follows –

Task scheduling in dynamic computing environments

Optimizing resource utilization and performance in dynamic computing environments, such as cloud computing systems, hinges on effective task scheduling. Traditional task scheduling algorithms often struggle to adapt to the constantly changing conditions in these environments, which can result in suboptimal solutions and inefficient resource use. The dynamic nature of cloud computing involves fluctuating workload demands, varying resource availability, and shifting task priorities, all of which complicate the scheduling process. To address these challenges, researchers have investigated a range of approaches, including optimization algorithms, machine learning techniques, and hybrid methods that combine the strengths of both. Optimization algorithms are designed to explore a wide array of potential solutions to find the best possible scheduling outcomes. However, they can be limited by their inability to adapt swiftly to changes. On the other hand, machine learning techniques excel at making adaptive, data-driven decisions, but they may require extensive training and can be resource-intensive. Hybrid approaches seek to integrate the exploratory power of optimization algorithms with the adaptive capabilities of machine learning. These methods aim to provide more robust and efficient task scheduling solutions, capable of responding dynamically to the ever-changing demands of cloud computing environments. By leveraging the strengths of both optimization and machine learning, hybrid approaches hold promise for significantly improving resource utilization and overall performance in dynamic computing settings.

Genetic Algorithms (GA)

Genetic Algorithms (GA) are population-based optimization techniques inspired by the principles of natural selection and genetics. The evolutionary algorithms simulate the process of evolution[14,18,19], using mechanisms such as selection, crossover, and mutation to evolve a population of potential solutions over successive generations. GAs are particularly effective at exploring large solution spaces, making them well-suited for complex optimization problems where finding near-optimal solutions is critical. In task scheduling, GAs have been widely utilized due to their robustness and adaptability. They excel in handling the multifaceted challenges of scheduling, such as balancing workload distribution, minimizing processing times, and optimizing resource allocation. The iterative process of selection and refinement allows GAs to efficiently navigate through vast and complex search spaces, honing in on high-quality solutions that meet specific performance criteria. Furthermore, GAs' ability to adapt to changing conditions makes them highly effective in dynamic computing environments like cloud computing systems, where workload demands and resource availability are constantly shifting. By leveraging the evolutionary strategies of natural selection and genetics, GAs provide a powerful approach to solving a wide range of optimization problems, ensuring efficient and effective task scheduling and resource utilization in various computational settings.

Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA) is a nature-inspired optimization technique modeled after the social behavior of humpback whales. This algorithm mimics the whales' hunting strategy, particularly the bubble-net feeding method, to perform optimization tasks. Due to its ability to effectively explore large solution spaces and rapidly converge towards optimal solutions, WOA has been successfully applied to a wide range of optimization problems, including task scheduling. In task scheduling, WOA's efficiency lies in its capacity to balance exploration and exploitation during the search process. This balance ensures that the algorithm can both identify promising areas of the solution space and fine-tune solutions to achieve optimal performance. As a result, WOA has demonstrated considerable success in improving resource utilization, reducing processing times, and meeting deadlines in dynamic computing environments such as cloud computing systems. The robustness and adaptability of WOA make it a valuable tool in addressing the complex and evolving challenges of task scheduling, where traditional algorithms often fall short. By leveraging the natural strategies of humpback whales, WOA provides a powerful approach to optimizing performance and resource use in various computational settings.

Reinforcement learning techniques

Reinforcement learning techniques, including Q-learning and SARSA, have become increasingly popular for solving dynamic decision-making problems. These methods empower agents to learn optimal policies by interacting with their environment, allowing them to make informed decisions based on feedback from their actions. In task scheduling for dynamic computing environments, reinforcement learning techniques are particularly advantageous[5,17]. These environments are characterized by constantly changing workload demands, resource availability, and task priorities. Traditional scheduling algorithms often struggle to adapt to these fluctuations, but reinforcement learning offers a robust solution. Q-learning and SARSA, for example, enable agents to continuously improve their scheduling policies by learning from the outcomes of their decisions. Through trial and error, these agents can identify the most efficient ways to allocate resources and schedule tasks, even as conditions change. This adaptive learning process ensures that the scheduling strategy evolves to meet current demands, optimizing resource utilization and performance. By leveraging the strengths of reinforcement learning, task scheduling systems can become more responsive and efficient, leading to improved overall performance in dynamic computing environments such as cloud computing systems. The ability of these techniques to handle complexity and variability makes them an excellent choice for modern, dynamic task scheduling challenges.

Traditional task scheduling typically relies on First-Come, First-Served (FCFS) or Round Robin approaches. However, the increasing complexity and dynamic nature of modern computing environments require more sophisticated scheduling algorithms to optimize performance, reduce execution time, and minimize costs, including energy consumption and carbon emissions.

Genetic algorithms, which are based on evolutionary principles, offer robust solutions for complex task scheduling. Meanwhile, Whale Optimization Algorithm (WOA) is known for its fast convergence and ability to balance exploration and exploitation of the search space, making it effective even under highly variable task loads. In environments with fluctuating and uncertain load conditions, reinforcement learning techniques like Q-learning and SARSA are particularly valuable, as they leverage feedback mechanisms to adapt and optimize scheduling in real-time.The dynamic nature of task scheduling and the variability in demand necessitate solutions that approach optimal performance. While various algorithms excel in specific aspects, none fully integrates all necessary factors into a single framework. This paper explores hybrid approaches to address the challenges of uncertainty, complexity, and variability in task scheduling, aiming to enhance efficiency and achieve optimal solutions that reduce execution time, lower energy consumption, and promote green computing.

RELATED WORK

In this section, we explain the previous works in the same domain.

Hybrid optimization algorithms

Previous research has investigated hybrid optimization algorithms [4,19] that integrate various techniques, including the Whale Optimization Algorithm (WOA), Genetic Algorithms (GA), and reinforcement learning, to tackle task scheduling challenges. These hybrid approaches [7] aim to harness the unique strengths of each individual algorithm while mitigating their respective limitations, resulting in enhanced performance and efficiency. Hybrid optimization algorithms combine the exploratory capabilities of WOA, the adaptive and evolutionary strategies of GA, and the decision-making process of reinforcement learning techniques. WOA excels at searching large solution spaces effectively, GA offers robust mechanisms for evolving solutions through genetic operations, and reinforcement learning provides adaptive learning from environmental feedback. By integrating these diverse methodologies, hybrid algorithms can address the complexities and dynamic nature of task scheduling more comprehensively. For instance, while WOA can quickly identify promising areas within the solution space, GA can refine these solutions through its crossover and mutation processes, and reinforcement learning can continuously adapt and optimize the scheduling strategy based on real-time feedback. This synergy allows hybrid algorithms to achieve higher levels of resource utilization, minimize processing times, and meet task deadlines more reliably than single-method approaches. The combination of these powerful techniques ensures that hybrid optimization algorithms are well-equipped to handle the fluctuating demands and resource availability inherent in dynamic computing environments such as cloud computing systems. Consequently, these hybrid methods represent a significant advancement in the field of task scheduling, offering more efficient and effective solutions to modern computational challenges.

Multi-Objective Optimization

Research in multi-objective optimization [8,13] for task scheduling aims to simultaneously optimize several conflicting objectives, such as makespan, resource utilization, and energy consumption. This

approach recognizes that optimizing for a single criterion often leads to suboptimal outcomes in other areas, necessitating a balanced strategy that considers multiple factors. In dynamic computing environments, such as cloud computing systems, achieving this balance is particularly challenging due to the ever-changing workload demands and resource availability. Various algorithms and techniques have been developed to address these multi-objective task scheduling problems effectively.

One common approach involves evolutionary algorithms, which are well-suited for handling multiple objectives. For instance, the Non-dominated Sorting Genetic Algorithm (NSGA-II) and the Multi-Objective Particle Swarm Optimization (MOPSO) are popular methods that maintain a diverse set of solutions, allowing for a comprehensive exploration of the trade-offs between different objectives. These algorithms use techniques such as Pareto dominance to identify and evolve a set of optimal solutions that provide a balanced compromise among the various objectives.Research on other optimization techniques[9,21] include hybrid algorithms that combine elements of heuristic methods, metaheuristic strategies, and machine learning. These hybrid approaches leverage the strengths of individual methods to enhance performance and adaptability. For example, integrating reinforcement learning with evolutionary algorithms can improve the adaptability of the scheduling strategy in real-time, dynamically optimizing resource allocation as conditions change.

By focusing on multi-objective optimization, researchers aim to develop task scheduling solutions that not only minimize processing times and maximize resource utilization but also reduce energy consumption and other operational costs. This holistic approach ensures that the overall performance and efficiency of the computing environment are optimized, leading to more sustainable and cost-effective operations. The advancements in this field contribute significantly to the capability of dynamic computing systems to meet the diverse and evolving demands of modern applications.

Real-time task scheduling

Research in real-time task scheduling, as indicated by studies [6,9,11], centers on the optimization of task assignments and resource allocations in immediate response to stringent deadlines and performance demands. This area of study is crucial for applications where timely execution is critical, such as in industrial automation, real-time analytics, and mission-critical systems. Research in real-time task scheduling, as indicated by studies [9,11], centers on the optimization of task assignments and resource allocations in immediate response to stringent deadlines and performance demands. This area of study is crucial for applications where timely execution is critical, such as in industrial automation, real-time analytics, and mission-critical systems. Real-time task scheduling poses unique challenges due to the need for quick decision-making in dynamically changing environments. To address these challenges, researchers employ various techniques, including online learning, dynamic programming, and heuristic algorithms. Online learning algorithms continuously adapt and improve scheduling decisions based on real-time feedback and observations. By learning from past experiences, these algorithms can make informed decisions even in unpredictable environments. Dynamic programming techniques break down complex scheduling problems into smaller subproblems, allowing for efficient computation of optimal solutions. Heuristic algorithms, on the other hand, offer practical and fast solutions by employing rules of thumb or approximation methods to quickly generate schedules. Moreover, recent advancements in machine learning and artificial intelligence have further enriched the capabilities of real-time task scheduling systems. Techniques such as reinforcement learning enable agents to learn optimal scheduling policies through trial and error interactions with the environment. This adaptive learning process empowers systems to dynamically adjust scheduling decisions based on changing conditions, ensuring that deadlines are consistently met while optimizing resource utilization and performance. The significance of real-time task scheduling extends beyond individual applications to broader domains such as cloud computing, edge computing, and internet-of-things (IoT) systems, where responsiveness and efficiency are paramount. By continually refining scheduling strategies and leveraging innovative techniques, researchers strive to develop robust and adaptable solutions capable of meeting the evolving demands of real-time computing environments.

Power-aware task scheduling

Research in power-aware task scheduling, as evidenced by studies [13,15,16], seeks to reduce power or energy consumption in computing systems while still meeting performance requirements. This area of study is crucial for addressing the growing concerns about energy efficiency and sustainability in computing environments. Power-aware task scheduling involves optimizing the allocation of tasks to resources while considering power constraints. This optimization process aims to minimize energy consumption without compromising performance. To achieve this goal, researchers employ a variety of techniques, including optimization algorithms, machine learning methods, and dynamic voltage and

frequency scaling (DVFS). Optimization algorithms are used to find the most energy-efficient task assignments and resource allocations. These algorithms explore different scheduling strategies to identify solutions that strike a balance between power consumption and performance metrics such as makespan or response time. Machine learning techniques, on the other hand, leverage data-driven approaches to learn from past behavior and make intelligent decisions about task scheduling. By analyzing patterns and trends in workload and resource usage, machine learning models can predict optimal scheduling policies that minimize energy usage. Dynamic voltage and frequency scaling (DVFS) is another key technique used in power-aware task scheduling. DVFS adjusts the operating voltage and frequency of computing components, such as processors and memory, dynamically based on workload demands. By scaling down voltage and frequency during periods of low activity, DVFS reduces power consumption without sacrificing performance. Conversely, during peak workload periods, DVFS can increase voltage and frequency to ensure that tasks are completed efficiently. Power-aware task scheduling is essential for a wide range of computing systems, including data centers, cloud computing environments, and mobile devices. By implementing energy-efficient scheduling policies, organizations can reduce operational costs, minimize environmental impact, and extend the battery life of mobile devices. Continued research in this area is crucial for developing innovative strategies to address the growing demand for energy-efficient computing solutions.

Research on dynamic virtual machine consolidation has increasingly focused on minimizing execution time, makespan which reducing energy consumption, and enhancing environmental sustainability. Various individual and hybrid approaches have been explored, each addressing one or more of these factors depending on the algorithms employed. However, the complexity of fluctuating conditions requires comprehensive strategies that account for all relevant factors to achieve truly optimal results. The author identifies two critical elements for optimization: the speed at which exploration can be controlled, and the ability to maintain effective exploitation. Leveraging feedback mechanisms or prior experiences is essential in balancing these elements and guiding the system toward an optimal solution.

RESEARCH METHODOLOGY

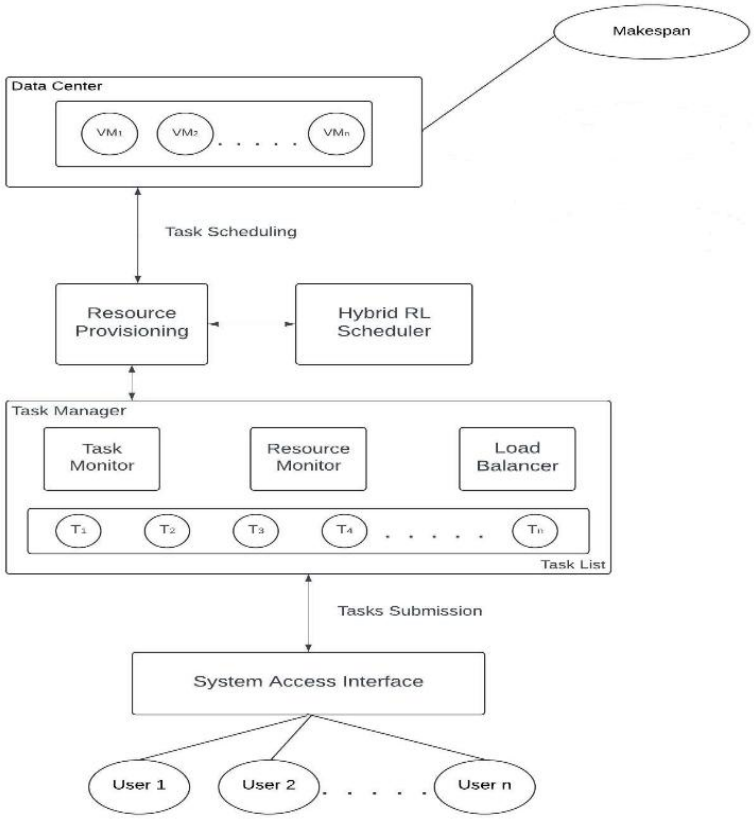


Figure 1. Overall flow of the proposed methodology

In this paper, the author presents a comprehensive approach to cloud-based task scheduling and resource management, illustrated through a high-level architectural design. This architecture, depicted in the

Figure, leverages reinforcement learning (RL) techniques, specifically the Q-Whale and SARSA-Whale algorithms, to optimize task scheduling within a cloud infrastructure. The primary objective of this system is to ensure efficient resource utilization while minimizing the Makespan, which is the total time required to complete all submitted tasks. At the heart of this architecture lies the Hybrid RL Scheduler (HRLS), which utilizes advanced RL algorithms, Q-Whale and SARSA-Whale (S-Whale), to manage the dynamic and complex nature of task scheduling in cloud environments. Q-Whale is a variant of the model-free Q-learning algorithm, designed to maximize cumulative rewards by selecting the best possible action based on the current state. In contrast, S-Whale is based on the SARSA algorithm, which stands for State-Action-Reward-State-Action and follows an on-policy learning approach, updating its policy based on the actions actually taken rather than the theoretically optimal actions. Tasks are submitted by users through the System Access Interface, which acts as the system's entry point. These tasks are then managed by the Task Manager, a critical component responsible for monitoring task statuses, tracking available resources, and balancing the workload across the system. The Task Manager includes a Task Monitor, which oversees task execution; a Resource Monitor, which manages computational resources; and a Load Balancer, which ensures tasks are evenly distributed across available resources, preventing any single resource from becoming overburdened.

The HRLS works in close coordination with the Resource Provisioning module, responsible for allocating appropriate resources—such as virtual machines (VMs) within the Data Center—based on the scheduler's decisions. This real-time interaction is vital for adjusting resource allocation dynamically, ensuring efficient execution of tasks across all available VMs (VM1, VM2, …, VMn). Once tasks are scheduled and resources are provisioned, they are executed within the Data Center. The system's performance is measured by the Makespan, with the overarching goal of minimizing this metric to enhance overall efficiency and throughput. This architecture, integrating the Q-Whale and S-Whale algorithms, represents a sophisticated and effective approach to cloud task scheduling. It underscores the critical role of reinforcement learning in optimizing resource management and task execution in cloud computing environments. By improving resource utilization, reducing operational costs, and minimizing task completion times, this system enhances the responsiveness and efficiency of cloud services. In the Figure, the requests are characterized as cloudlets or tasks, which users submit through an interface. These tasks are then allocated to selected virtual machines (VMs) across multiple data centers by the task manager, following the scheduling policy determined by the hybrid algorithm. The system's performance is evaluated using the makespan metric. As referenced in [4,10], the makespan refers to the time at which the last task in a set is completed, making it a key indicator of the effectiveness of scheduling methodologies. A lower makespan reflects superior and more optimal task scheduling for virtual machines (VMs) or servers. The program logic for calculating the makespan can be explained through the following steps:

1. Initialize Makespan=0, Finishtime;
2. Repeat it until all task get success status
3. If (Finishtime>Makespan)
4. Then Makespan=Finishtime

Here Finishtime refer to maximum time taken by any task assign to VMs.

In this section, we also explain various algorithms that were used in the proposed methodology. The Q-Whale algorithm, drawing inspiration from the hunting behaviors of killer whales, emerges as a metaheuristic approach used to tackle optimization challenges, particularly within the realm of cloud computing. Its primary objective is to minimize the makespan, representing the aggregate duration required to fulfill a specified set of tasks. In the context of cloud computing, the makespan signifies the duration taken to execute a batch of tasks across multiple VMs or servers. Efficient distribution of tasks among the available resources is crucial in cloud computing scenarios, aiming to minimize the makespan. By leveraging the Q-Whale algorithm, which combines exploration capabilities akin to killer whale hunting behaviors with optimization techniques, cloud computing systems can effectively allocate tasks across VMs or servers, thereby reducing the overall makespan. This optimization enhances system performance, resource utilization, and ultimately, user satisfaction.

Q-Whale algorithm

The Q-Whale algorithm, as outlined in [3], offers a strategic approach to minimize makespan in cloud computing environments. Here's a detailed breakdown of how this algorithm can be implemented:

a. Initialization: The process begins by establishing an initial population of solutions. In the context of cloud computing, this entails randomly allocating tasks to virtual machines (VMs) or servers within the infrastructure.
b. Evaluation: Each solution's makespan is computed by simulating the task execution on the assigned

resources and measuring the cumulative time required for completion.

c. Selection: Solutions exhibiting shorter makespan values are prioritized as promising candidates for further consideration. This selection process aims to identify the most efficient task-resource allocations.
d. Reproduction: New solutions are generated through the application of genetic operators, such as crossover and mutation. These operations facilitate the exploration of alternative task-resource assignment configurations by combining or modifying existing solutions.
e. Replacement: To maintain population diversity and prevent premature convergence, a subset of solutions in the population is replaced with the newly generated alternatives. This step fosters continual exploration of the solution space.
f. Termination: The iterative process of evaluation, selection, reproduction, and replacement continues until a predefined termination condition is met. This condition could be based on reaching a maximum number of iterations, achieving a specified level of improvement, or surpassing a predetermined time threshold.

Throughout this iterative optimization process, the Q-Whale algorithm dynamically adapts and evolves the population of solutions. By iteratively refining the assignment of tasks to resources, it endeavors to converge towards an optimal or near-optimal solution for the cloud computing workload. This iterative refinement, driven by the algorithm's exploration and exploitation capabilities, contributes to the continuous improvement of makespan and overall system performance in cloud computing environments.

Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA), referenced in [4,7,18], stands as a nature-inspired optimization technique emulating the social interactions of humpback whales. WOA operates by maintaining a population of candidate solutions, colloquially termed "whales," which are iteratively refined to approach optimal or near-optimal solutions. Its notable strength lies in effectively balancing exploration and exploitation throughout the optimization process. During the exploration phase, WOA navigates the solution space to uncover new promising regions, while in the exploitation phase, it leverages these discovered areas to further refine solutions. This dual strategy allows WOA to dynamically adapt its approach, promoting both thorough exploration and efficient exploitation of the solution space.

D = |C X* — X(t)|
(1)
X (t + 1) = X*(t)— A. D(2)

where t is the current iteration, X* is the best solution acquired so far, X is the current solution. A and C are coefficients computed as following:

A = 2a.r — a (3)
C = 2. r (4)

where a is linearly reduced from 2 to 0 over the trajectory of iterations as showing in Eq. (3) and r is a random number between 0 and 1.

The spiral updating position mechanism involves computing the distance between the current solution (whale) and the best solution (victim) by using the spiral equation as following:

X(t + 1)= D' .ebl. cos(2Pl)+ X*(t) (5)
X(t)=X* -D exp(b.l).Cos(2πl) (6)

where D is the distance between the whale and the victim, b is a constant for defining the shape of the logarithmic spiral, and l is a random number between - 1 and 1. Humpback whales use both mechanisms simultaneously. To model this behavior, a probability of 50% is introduced to select one of the mechanisms to update the whales' location during the search. The mathematical model is as follows:

$$X(t+1) = \begin{cases} Shrinkingencirclingequation\ (2), & if p < 0.5 \\ Spiralshapedpathequation(6), & if p \geq 0.5 \end{cases} \quad (7)$$

where p is a random number in [0,1]

While in the exploration phase, WOA involves a global search. The whales search randomly based on the location of each other. Therefore, a search agent's location is updated randomly instead of depending on the best search agent identified so far. This technique is used when the random values of A are greater than one to cause the search agent to move away from a reference whale.The mathematical model for the exploration phase is as follows:

D = |C.Xrand — X| (8)
X(t + 1)= Xrand — A.D (9)

WOA's effectiveness extends to a wide array of optimization problems, particularly those featuring nonlinear and multimodal objective functions. Its adaptive nature renders it well-suited for addressing dynamic optimization challenges, such as task scheduling in dynamic computing environments. In this

context, WOA continually updates the movement of whales around the target, akin to their hunting behavior, resulting in a mathematical model that guides the optimization process effectively. This adaptability enables WOA to respond to changing conditions and evolving objectives, ensuring robust performance in dynamic scenarios.

Q-learning algorithm

Q-learning [1,2] is a reinforcement learning technique used for decision-making in dynamic and uncertain environments. In Q-learning, an agent learns a policy for selecting actions based on its interactions with the environment. The agent maintains a Q-table (or Q-function), which stores Q-values representing the expected cumulative rewards for taking specific actions in given.

Q-learning [3] has been successfully applied to various decision-making problems, including robotic control, game playing, and resource allocation. In the context of task scheduling, Q-learning can learn an optimal policy for assigning tasks to computing resources based on the current state of the system, task characteristics, and environmental factors.

Q-table utilizes a state-action pair to index a Q value as a cumulative reward and is denoted as Q(s, a) where s is the state, and a is the action. The Q-table is dynamically updated depending on a given state-action pair's reward/ punishment.

$$Q(t+1)(st, at)= Q(st, at)+ \alpha(r + \gamma \max(Q(s', a')) — Q(st, at)) \quad (10)$$

where γ is the discount factor within [0,1], r is reward/ punishment and α is the learning rate within [0,1] and calculated as follows:

$$\alpha = 1 - 0.9 * (t/Epoch) \quad (11)$$

The value of the parameter a is an indication to perform exploration or exploitation. If the value is close to 1, the recently acquired data are given a greater priority, meaning exploration is performed for all defined states. Whereas, if the value is close to 0, the current data are given greater priority to be exploited. The value of the parameter c is an indication of whether to take the current reward/punishment or the previous one, and it was set to 0.

The value of parameter r is set as follows: r(t) = 1,
if the current action improves the solution r(t) = —1
otherwise (12)

SARSA algorithm SARSA [1,20] is an on-policy algorithm designed to teach a machine learning model a new Markov decision process policy in order to solve reinforcement learning challenges. It's an algorithm where, in the current state (S), an action (A) is taken and the agent gets a reward (R), and ends up in the next state (S1), and takes action (A1) in S1. Therefore, the tuple (S, A, R, S1, A1) stands for the acronym SARSA.

In the SARSA update equation depends on current value and target value. Target value based on behaviour policy, the Q-value is chosen using S' and A', the next state and the action chosen in the next state, respectively. This stands in contrast to Q-learning, which updates the equation where the max of Q(S', a) is taken.

$$Q(t+1)(st, at)= Q(st, at)+ \alpha(r + \gamma(Q(s', a')) — Q(st, at)) \quad (13)$$

The remaining process of the code is similar to the Q-learning code.

Q-Whale algorithm

The Q-Whale algorithm, as referenced in [2,9], merges the exploratory prowess of the Whale Optimization Algorithm (WOA) with the adaptive decision-making capabilities of Q-learning techniques to refine task scheduling in dynamic computing environments. In this innovative approach, WOA takes charge of traversing the solution space and crafting potential scheduling solutions. These solutions undergo thorough evaluation, assessing metrics such as makespan, resource utilization, and meeting deadlines to gauge their quality. Q-learning, a reinforcement learning technique detailed in [11], contributes to the exploration process by furnishing feedback on the efficacy of the generated solutions. Drawing from past experiences, Q-learning influences the selection of actions, or task scheduling decisions, generated by WOA. By favoring actions associated with higher rewards, Q-learning steers the exploration towards regions of the solution space that promise superior outcomes. This fusion of WOA and Q-learning aims to elevate the efficiency, resource utilization, and overall system performance in task scheduling endeavors within dynamic computing environments. Unlike conventional task scheduling algorithms, this hybrid methodology embraces adaptability and learning, enabling it to dynamically adjust to evolving conditions and achieve superior outcomes. Through iterative refinement and informed decision-making, the Q-Whale algorithm stands poised to revolutionize task scheduling practices in the realm of dynamic computing. The pseudo code of Q-Whale algorithm is as follows:

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.

The Q-Whalealgorithmspresents several advantages over conventional task scheduling algorithms and alternative hybrid methodologies. These advantages encompass:

a. Adaptability to Dynamic Environments: The Q-Whale algorithm adeptly navigates the dynamic landscape of computing environments by melding the exploratory prowess of WOA with the adaptive decision-making of Q-learning. This adaptability empowers the algorithm to swiftly respond to fluctuations in workload demands, shifts in resource availability, and evolving task priorities in real-time.
b. Efficient Exploration and Exploitation: Through the fusion of WOA and Q-learning, these algorithms efficiently explores the solution space while striking a delicate balance between exploration and exploitation. WOA's exploration phase uncovers promising regions, while Q-learning's exploitation phase refines solutions based on accumulated experiences, facilitating enhanced convergence towards optimal solutions.
c. Optimization of Multiple Objectives: The Q-Whale algorithm excels in simultaneously optimizing multiple objectives, such as minimizing makespan, maximizing resource utilization, and meeting task deadlines. By holistically considering diverse objectives, it offers a comprehensive approach to task scheduling optimization, thereby enhancing overall system performance.
d. Learning from Past Experiences: Leveraging Q-learning, the Q-Whale algorithm assimilates insights from past encounters to refine its decision-making process. By iteratively updating Q-values in response to observed rewards, the algorithm iteratively improves its policy, resulting in more informed decision-making and superior solution quality over time.
e. Effective Resource Utilization: With a focus on maximizing resource utilization, the Q-Whale algorithm efficiently allocates tasks across available computing resources. By integrating WOA and Q-learning, it identifies optimal task-resource assignments that minimize idle time and optimize computing resource usage, thereby enhancing overall efficiency.
f. Scalability and Robustness: The scalability and robustness of the Q-Whale algorithm render it well-suited for diverse computing environments, including large-scale cloud computing systems. Capable of tackling complex scheduling challenges involving numerous tasks and resources, it maintains efficiency and effectiveness across varying scales of operation.
g. Experimental Validation: Empirical validations and experiments conducted within dynamic computing environments underscore the efficacy of the Q-Whale algorithm. Demonstrating tangible improvements in resource utilization, makespan reduction, and task deadline adherence compared to conventional task scheduling methods and alternative hybrid approaches, these experimental findings attest to the real-world superiority of the Q-Whale algorithm.

SARSA-Whale algorithm

The SARSA-Whale algorithm, an on policy learning algorithm which work almost similar to Q-whale and perform better specially in high load conditions claimed in this paper also merges the exploratory prowess of the Whale Optimization Algorithm (WOA) with the adaptive decision-making capabilities of Q-learning techniques to refine task scheduling in dynamic computing environments. In this innovative approach which specially presented in this paper with its psuedo code, WOA takes charge of traversing the solution space and crafting potential scheduling solutions. These solutions undergo thorough

evaluation, assessing metrics such as makespan, resource utilization, and meeting deadlines to gauge their quality. SARSA, a reinforcement learning technique detailed in [11], contributes to the exploration process by furnishing feedback on the efficacy of the generated solutions. Drawing from past experiences, Q-learning influences the selection of actions, or task scheduling decisions, generated by WOA. By favoring actions associated with higher rewards, Q-learning steers the exploration towards regions of the solution space that promise superior outcomes. This fusion of WOA and Q-learning aims to elevate the efficiency, resource utilization, and overall system performance in task scheduling endeavors within dynamic computing environments. Unlike conventional task scheduling algorithms, this hybrid methodology embraces adaptability and learning, enabling it to dynamically adjust to evolving conditions and achieve superior outcomes. Through iterative refinement and informed decision-making, the Q-Whale algorithm stands poised to revolutionize task scheduling practices in the realm of dynamic computing. The pseudo code of Q-Whale algorithm is as follows:

The psuedo code for SARSA-Whale Algortihm

1. Initialize Q(s, a) arbitrarily
2. Repeat for each episode:
   2.1 Initialize state s
   2.2 Choose action a from state s using policy derived from Q (e.g., ϵ-greedy)
   2.3 Repeat for each step of the episode:

i. Take action a, observe reward r and next state s_{t+1}

   ii. Choose action a' from state s_{t+1} using policy derived from Q (e.g., ϵ-greedy)
   iii. Update Q-value:

$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s_{t+1}, a') - Q(s, a)]$

   iv. Set state s to s_{t+1}
   v. Set action a to a'
   2.4 Until state is terminal
3. Schedule cloudlets using the Whale Optimization Algorithm (WOA):
   3.1 Create a WOA scheduler object.
   3.2 Pass cloudlets and VMs to the scheduler.
   3.3 Execute the scheduling algorithm using WOA:

i. Initialize the whale population and parameters.

   ii. Calculate the fitness of each whale.
   iii. Update the position of each whale:
      - If the whale is near the prey:
        Update the whale's position towards the prey.
      - If the whale is searching for prey:
        Perform a random search based on a spiral position update.
   iv. Evaluate the new positions and update the best solution found.
   v. Repeat until the stopping criterion is met (e.g., maximum iterations or convergence).
4. Output the scheduled cloudlets with optimized VM allocation.

The SARSA-Whale algorithm offers several distinct advantages over conventional task scheduling algorithms and alternative hybrid methodologies:

a. 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.

b. 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.

c. 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.

d. 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.

e. 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.

f. 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.

g. 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.

h. 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.

EXPERIMENTAL RESULTS

In this section, we present the experimental results. The results are calculated and compared based on makespan. Figure 2 compares the performance of various algorithms: SARSA, Q-learning, Whale, SARSA Whale, and Q-Whale. For the simulation, we deployed 10 VMs across 3 data centers. Each data centre contains 2 hosts, with each host having a single CPU. The VMs are configured with 100 MIPS (Millions of Instructions Per Second) in a homogeneous setup, while in a heterogeneous setup, the MIPS values vary: low-performing VMs have 1000 MIPS, medium-performing VMs have 2000 MIPS, and high-performing VMs have 4000 MIPS. Additionally, each VM is allocated 2 GB of RAM. These parameters define the computational resources and network characteristics of the simulated environment, which influence the performance and behavior of the VMs during the simulation. The variability in VM configurations, particularly in terms of RAM and MIPS, allows us to test a range of scenarios. This helps us understand the performance and scalability of the data centre under different conditions. Figure 1 shows that QWhale performs the best.

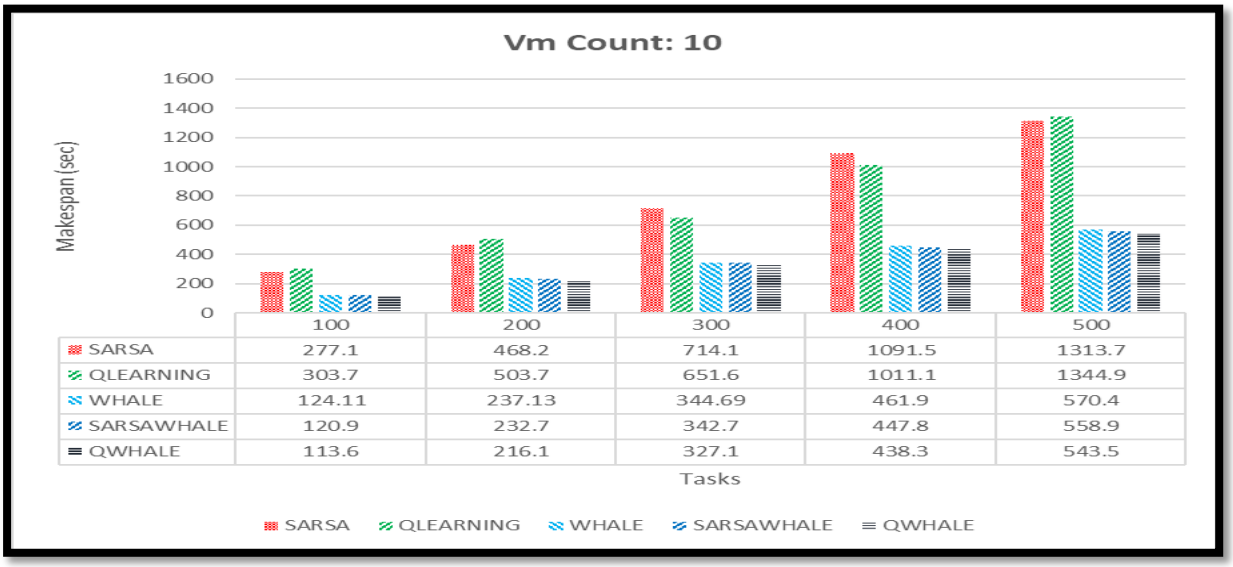
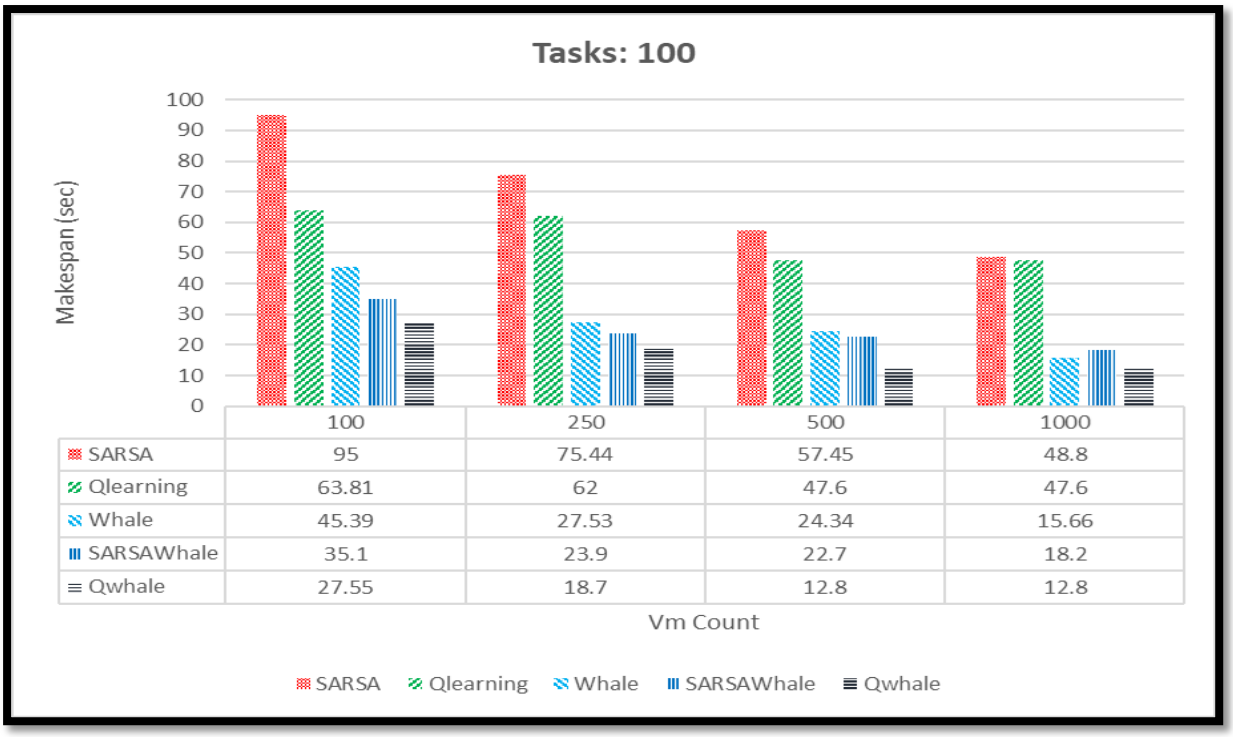


Figure 2. Performance of various algorithms for VM count =10

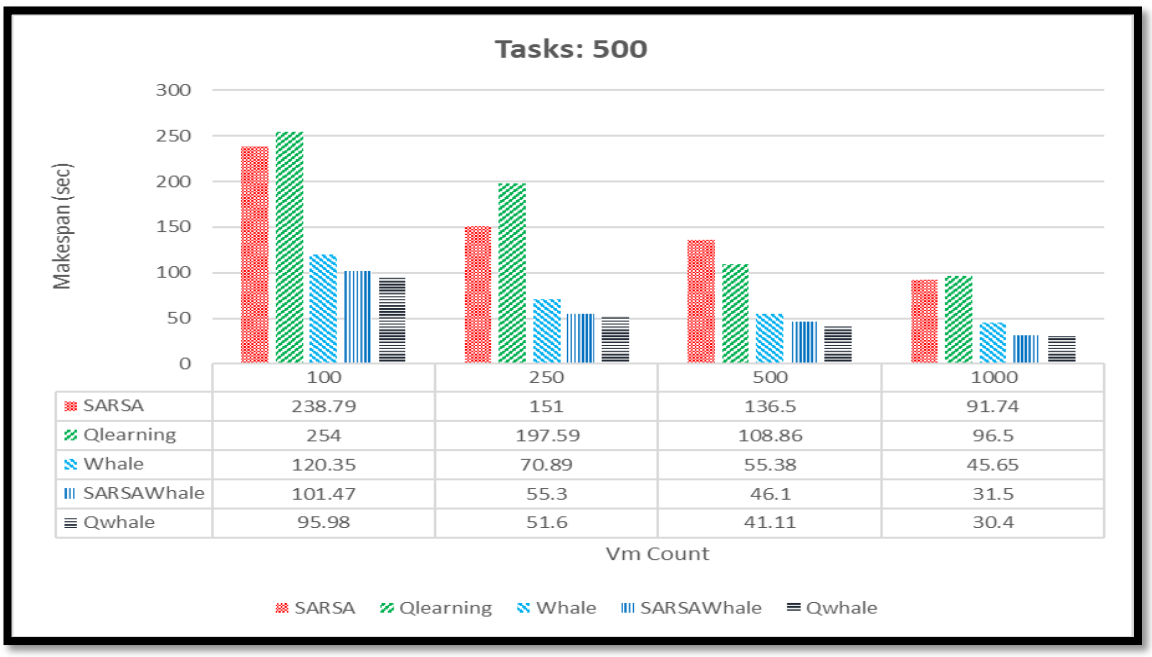
Figure 3 presents the results of our analysis for 100 tasks. Among the different approaches, Q-Whale demonstrates the best performance for 100 virtual machines (VMs), with SARSA-Whale following closely as the second-best. The Whale algorithm significantly outperforms both SARSA and Q-learning, achieving a makespan of 27.55 seconds. As we scale our deployment to 250 VMs, Q-Whale continues to lead, achieving an improved makespan of 18.7 seconds. This trend persists with further scaling; Q-Whale maintains its superior performance, reporting a makespan of 12.8 seconds when the deployment is expanded to 500 and then 1,000 VMs.



	100	250	500	1000
SARSA	95	75.44	57.45	48.8
Qlearning	63.81	62	47.6	47.6
Whale	45.39	27.53	24.34	15.66
SARSAWhale	35.1	23.9	22.7	18.2
Qwhale	27.55	18.7	12.8	12.8

Figure 3. Performance of various algorithms for tasks = 100 and varying VM counts.

Figure 4 illustrates a comparison of makespan across different approaches when 500 tasks are assigned. Among these, Q-Whale consistently exhibits the best performance, particularly in scenarios involving 100 virtual machines (VMs), where it outperforms other methods. SARSA-Whale ranks as the second-best approach in this setup, with the reported makespan for the 100 VM deployment being 95.98 seconds. As the deployment is scaled up to 250, 500, and ultimately 1,000 VMs, Q-Whale continues to maintain its lead, delivering superior performance compared to the other algorithms. This demonstrates Q-Whale's ability to effectively manage the increased computational load, maintaining an optimal balance between exploration and exploitation, which is crucial for achieving minimal makespan across varying scales of VM deployments.



	100	250	500	1000
SARSA	238.79	151	136.5	91.74
Qlearning	254	197.59	108.86	96.5
Whale	120.35	70.89	55.38	45.65
SARSAWhale	101.47	55.3	46.1	31.5
Qwhale	95.98	51.6	41.11	30.4

Figure 4. Performance of various algorithms for tasks = 500 and varying VM counts.

In Figure 5, we show the results for 1000 tasks analysis. For 100 VMs, Q-Whale outperforms all other algorithms, with SARSA-Whale being the second-best. Whale is significantly better than SARSA and Q-learning. We achieve a makespan of 157.8 seconds. When we scale the deployment to 250 VMs, Q-Whale and SARSA-Whale again show the best performance, with Q-Whale slightly ahead. The reported makespan is 82.42 seconds. Q-Whale and SARSA-Whale continue to outperform the other algorithms, with Q-Whale leading when we deploy 500 VMs for conducting same 1000 tasks. The makespan we achieve for Q-Whale us 58.3 seconds. Finally, we make 1000 VMs for performing 1000 tasks, Q-Whale achieves the best performance with 44.97 seconds makespan, closely followed by SARSA-Whale which has a makespan of 47.6 seconds. Whale also performs well compared to SARSA and Q-learning.

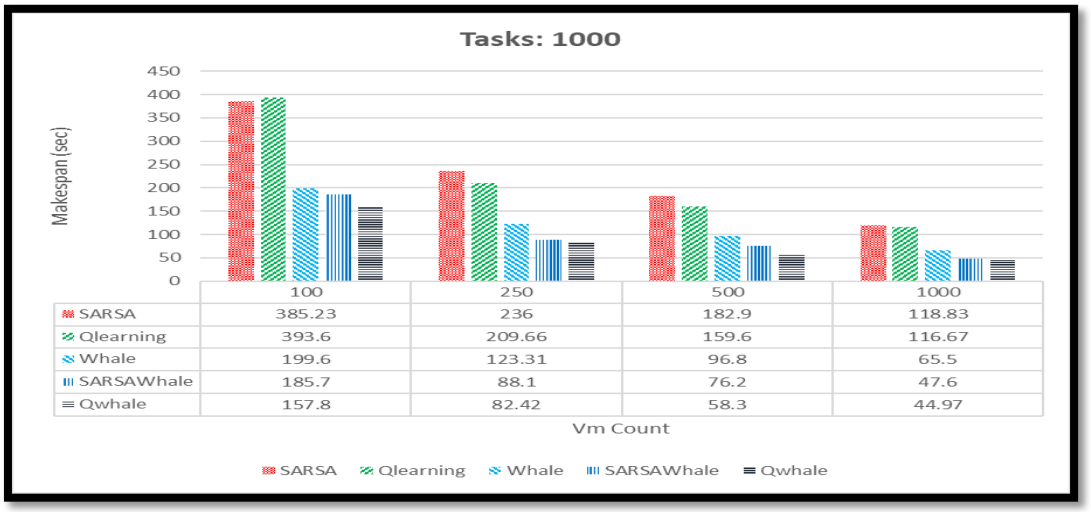


Figure 5. Performance of various algorithms for tasks = 1000 and varying VM counts.

In Figure 6, we had 5000 tasks analysis. When we have 100 VMs, Q-Whale performs the best with the shortest makespan of 729.1 seconds, followed closely by SARSA-Whale and Whale which have values 769.7 seconds and 795.62 seconds. SARSA and Q-learning have much higher makespan. When we have 250 VMs Q-Whale and SARSA-Whale again outperform the other algorithms, with Q-Whale being slightly better. The makespan for Q-Whale is 312.1 seconds. When we have 500 VMs, Q-Whale and SARSA-Whale maintain their superior performance, with the Whale algorithm still performing significantly better than SARSA and Q-learning. The reported makespan value for Q-Whale is 215.2 seconds. When we have 1000 VMs, SARSA-Whale achieves the best performance, closely followed by Q-Whale. Whale also shows good performance compared to SARSA and Q-learning.

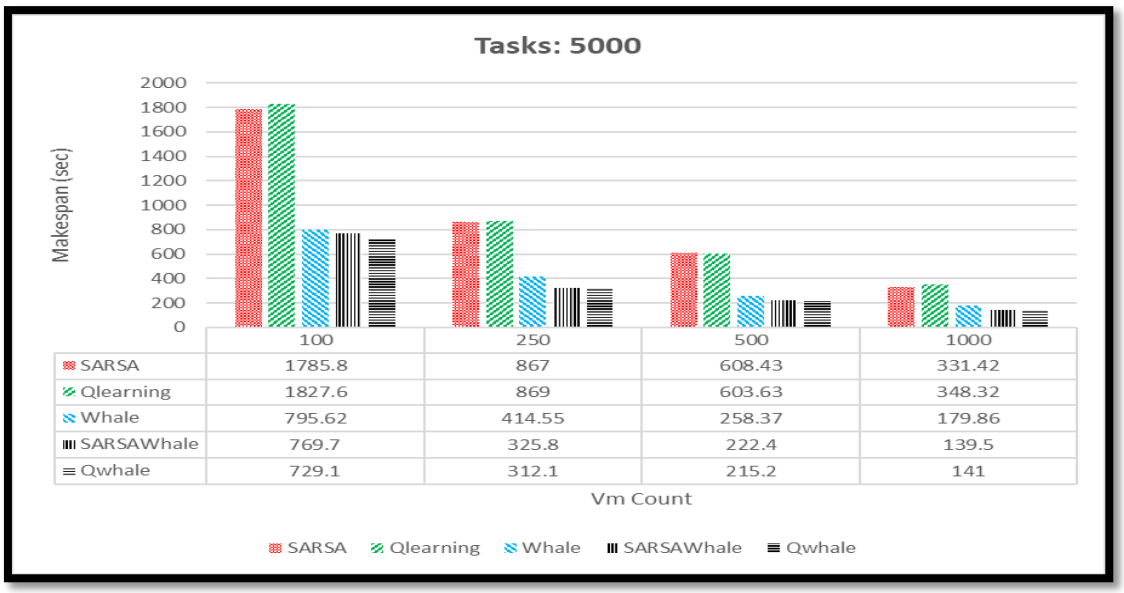


Figure 6. Performance of various algorithms for tasks = 5000 and varying VM counts.

Overall Observations and Insights

Our analysis reveals several key observations regarding the performance of the algorithms tested:

Superior Performance of Q-Whale: The Q-Whale algorithm consistently demonstrates the best performance across various VM counts and task numbers. It achieves the lowest makespan in all tested scenarios, indicating highly efficient task scheduling.

Strong Performance of SARSA-Whale: SARSA-Whale emerges as the second-best performer, consistently close to Q-Whale. Its performance across different configurations shows it is a robust and reliable algorithm for task scheduling specially when we are increasing the cloudlets against limited number of Vms.

Competent Performance of Whale Algorithm: The Whale algorithm also performs well, outperforming both SARSA and Q-learning. This indicates its strong capability in exploring and exploiting the solution space effectively.

Inefficiency of SARSA and Q-learning: SARSA and Q-learning generally exhibit higher makespans compared to the other algorithms, reflecting lower efficiency in task scheduling. Their performance highlights the limitations of using standalone reinforcement learning techniques for dynamic environments.

Advantages of Hybrid Approaches: The results clearly indicate that combining Whale optimization with reinforcement learning techniques like Q-learning and SARSA (resulting in the Q-Whale and SARSA-Whale algorithms) significantly enhances scheduling performance. These hybrid approaches leverage the strengths of both components, leading to better optimization and adaptability in dynamic computing environments.

CONCLUSION

Our experimental results provide compelling empirical evidence of the superiority of hybrid algorithms, particularly Q-Whale and SARSA-Whale, in the context of task scheduling. These algorithms combine the exploration strengths of Whale Optimization with the adaptive decision-making capabilities of reinforcement learning, resulting in significantly lower makespans, enhanced resource utilization, and overall improved system performance.

The integration of Whale Optimization's efficient search strategies with the feedback-driven learning processes of algorithms like Q-learning and SARSA enables these hybrids to adapt dynamically to the complexities of task scheduling in cloud computing environments. Their ability to balance exploration and exploitation makes them exceptionally well-suited for deployment in environments characterized by high variability and unpredictability, ensuring that resources are utilized optimally while maintaining high levels of system efficiency.

ETHICAL DECLARATION

Conflict of interest: No declaration required.
Financing: No reporting required.
Peer review: Double anonymous peer review.

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] Arvindhan, M., & Dhanaraj, R. K. (2023, November 6). Dynamic Q-Learning-Based Optimized Load Balancing Technique in Cloud. Journal of Mobile Information Systems. https://doi.org/10.1155/2023/7250267

[6] 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

[7] 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

[8] Sharma, S., & Pandey, N. K. (2024, February 1). Multi-Faceted Job Scheduling Optimization Using Q-learning With ABC In Cloud Environment. International Journal of Computing and Digital System/International Journal of Computing and Digital Systems. https://doi.org/10.12785/ijcds/150142

[9] 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

[10] Xiu, X., Li, J., Long, Y., & Wu, W. (2023, May 10). MRLCC: an adaptive cloud task scheduling method based on meta reinforcement learning. Journal of Cloud Computing. https://doi.org/10.1186/s13677-023-00440-8

[11] Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., & Zeng, J. (2020, July 1). Q-learning based dynamic task scheduling for energy-efficient cloud computing. Future Generation Computer Systems. https://doi.org/10.1016/j.future.2020.02.018

[12] Tran, C., Bui, T., & Pham, T. D. (2022, January 29). Virtual machine migration policy for multi-tier application in cloud computing based on Q-learning algorithm. Computing. https://doi.org/10.1007/s00607-021-01047-0

[13] 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

[14] 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

[15] 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

[16] Pawlish, M., Varde, A. S., & Robila, S. A. (2012, June 1). Analyzing utilization rates in data centers for optimizing energy management. https://doi.org/10.1109/igcc.2012.6322248

[17] Bajpai, G., Singh, P., Agarwal, A.K., A Comprehensive Review on Autonomous Consolidation of Virtual Machine for Energy and Resource Management. ACM International Conference Proceeding Series, 2023, https://doi.org/10.1145/3647444.3647901

[18] Naz, I., Naaz, S., Agarwal, P., Alankar, B., Siddiqui, F., & Ali, J. (2023, May 5). A Genetic Algorithm-Based Virtual Machine Allocation Policy for Load Balancing Using Actual Asymmetric Workload Traces. Symmetry. https://doi.org/10.3390/sym15051025

[19] Mangalampalli, S., Karri, G. R., & Köse, U. (2023, February 1). Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization. Journal of King Saud University. Computer and Information Sciences/MağalatĞam'aï Al-malīkSaud :Ùlm Al-ḥasibWa Al-ma'lumat. https://doi.org/10.1016/j.jksuci.2023.01.016

[20] Alfakih, T., Hassan, M. M., Gumaei, A., Savaglio, C., & Fortino, G. (2020). Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access, 8, 54074–54084. https://doi.org/10.1109/access.2020.2981434

[21] Jena, U., Das, P., & Kabat. (2022). Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. Journal of King Saud University - Computer and Information Sciences, 34(6), 2332–2342. https://doi.org/10.1016/j.jksuci.2020.01.012