Overall Algorithm:

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.

1. **Imports**: The code begins with importing necessary classes and packages from CloudSim and other relevant libraries.
2. **State and Action Classes**:
   * Two classes **State** and **Action** are defined. These are used to represent states and actions in the Q-learning algorithm.
   * The **State** class holds information about the cloudlet and the VM it is currently assigned to.
   * The **Action** class holds information about the selected VM for a cloudlet to be assigned to.
3. **QLearningProcessor Class**:
   * This class implements the Q-learning algorithm logic.
   * It includes methods to select actions based on the epsilon-greedy policy, update Q-values using the Q-learning update rule, and retrieve the maximum Q-value for a state.
   * The Q-values are stored in a **Map<State, Map<Action, Double>>** where each state maps to a map of actions to their corresponding Q-values.
4. **Main Method**:
   * The **main** method orchestrates the simulation process.
   * CloudSim is initialized and the simulation environment is set up.
   * Datacenters, broker, VMs, and cloudlets are created.
   * A QLearningProcessor object is instantiated with specified learning parameters (alpha, gamma, epsilon).
   * The main simulation loop runs for a certain number of iterations (in this case, 1000).
   * Within each iteration:
     + For each cloudlet, a state representation is obtained.
     + An action is selected using the Q-learning processor.
     + The selected action is executed (i.e., the cloudlet is assigned to the VM corresponding to the action).
     + A new state representation is obtained.
     + A reward is calculated based on the change in completion time between the old and new states.
     + Q-values are updated using the Q-learning update rule.
   * After the simulation loop completes, the Q-table is printed.
   * The cloud task scheduling is performed using the WOA scheduler.
   * The simulation is started and stopped.
   * Finally, the results of the simulation (cloudlet status, completion times, costs, etc.) are printed.
5. **Datacenter Creation, VM Creation, Cloudlet Creation**:
   * These methods create datacenters, VMs, and cloudlets with specified characteristics and parameters.
6. **Printing Results**:
   * Methods to print the results of the simulation, including cloudlet status, completion times, costs, etc.

Overall, the code integrates Q-learning into the cloud task scheduling simulation using CloudSim. It demonstrates how Q-learning can be applied to optimize cloudlet-VM assignments based on completion times, with the goal of reducing overall task completion time and costs.

**Load Balancing**

In the provided code, **LB** stands for Load Balancing. It's calculated as the standard deviation of the execution times of cloudlets across different VMs. Here's how it's calculated:

1. **Execution Time of VMs**: For each VM, the total execution time of all cloudlets assigned to it is calculated.
2. **Average Execution Time**: The average execution time across all VMs is computed.
3. **Load Balancing (LB)**: LB is calculated as the square root of the average squared deviation of the execution times of cloudlets from the average execution time. This is a measure of how evenly the workload is distributed across the VMs. A lower LB value indicates better load balancing.

**Algorithm 1:**

Algorithm: Cloud Simulation with Q-Learning

1. Initialize CloudSim package and create necessary datacenters, broker, VMs, and cloudlets.

1.1. Initialize CloudSim with the desired parameters (number of users, simulation start time, etc.).

1.2. Create datacenters with different characteristics (e.g., low, medium, high).

1.3. Create a broker to manage the simulation.

1.4. Create VMs and cloudlets according to requirements.

2. Initialize Q-learning processor.

2.1. Define Q-learning parameters such as learning rate (alpha), discount factor (gamma), and exploration factor (epsilon).

2.2. Initialize an empty Q-table to store Q-values for state-action pairs.

3. Train the Q-learning model.

3.1. Repeat for a predefined number of iterations or until convergence:

3.1.1. For each cloudlet in the cloudlet list:

3.1.1.1. Get the current state representation of the cloudlet and its assigned VM.

3.1.1.2. Select an action using an epsilon-greedy policy based on the current Q-values.

3.1.1.3. Execute the selected action by assigning the cloudlet to the corresponding VM.

3.1.1.4. Obtain the new state representation after the action.

3.1.1.5. Calculate the reward based on the change in completion time between the old and new states.

3.1.1.6. Update the Q-value of the current state-action pair using the Q-learning update rule.

4. Print the Q-table (optional).

4.1. Display the learned Q-values for each state-action pair.

5. Schedule cloudlets using the WOA scheduler.

5.1. Pass the cloudlet list and VM list to the WOA scheduler for task scheduling.

6. Start the simulation.

6.1. Begin the simulation using CloudSim.

6.2. Allow the simulation to run until completion.

7. Print the simulation results.

7.1. Retrieve the list of received cloudlets from the broker.

7.2. Print the details of each cloudlet including its ID, status, completion time, etc.

End Algorithm

**Algorithm 2:**

1. **Setup**: Prepare the cloud simulation environment with datacenters, VMs, and cloudlets.
2. **Initialize Q-learning**: Set up the Q-learning process with learning parameters and an empty Q-table.
3. **Train Q-learning model**:
   * For each cloudlet:
     + Determine the current state.
     + Choose an action based on the current state using an epsilon-greedy policy.
     + Execute the action and observe the reward.
     + Update the Q-value for the current state-action pair.
4. **Print Q-table**: Display the learned Q-values.
5. **Schedule cloudlets**: Use a scheduler (e.g., WOA) to assign cloudlets to VMs.
6. **Run simulation**: Start the simulation and let it proceed.
7. **Print results**: Output the performance metrics, such as completion times for cloudlets.

End of the process.

.

**Literature Part:**

Title: Q-Whale Algorithm: A Hybrid Approach for Optimizing Task Scheduling in Dynamic Computing Environments

Abstract: Task scheduling in dynamic computing environments presents a significant challenge due to the varying workload demands, resource availability, and task priorities. Traditional optimization algorithms often struggle to adapt efficiently to such dynamic conditions. In this paper, we propose the Q-Whale algorithm, a novel hybrid approach that combines the Whale Optimization Algorithm (WOA) with Q-learning techniques to address this challenge. The Q-Whale algorithm leverages the exploration capabilities of WOA and the adaptive decision-making of Q-learning to optimize task scheduling in real-time. Through experiments conducted in dynamic computing environments, we demonstrate the effectiveness of the Q-Whale algorithm in improving resource utilization, minimizing makespan, and meeting task deadlines compared to traditional approaches.

1. Introduction: Task scheduling plays a crucial role in optimizing resource utilization and performance in dynamic computing environments such as cloud computing systems. However, traditional task scheduling algorithms often struggle to adapt to the dynamic nature of these environments, leading to suboptimal solutions and inefficient resource utilization. In recent years, hybrid approaches that combine optimization algorithms with machine learning techniques have shown promise in addressing these challenges. In this paper, we introduce the Q-Whale algorithm, a novel hybrid approach that combines the Whale Optimization Algorithm (WOA) with Q-learning techniques to optimize task scheduling in dynamic computing environments.
2. Whale Optimization Algorithm (WOA):
   * Provide a brief overview of the Whale Optimization Algorithm and its application in optimization problems.
   * Highlight the exploration and exploitation capabilities of WOA and its suitability for dynamic optimization problems.
3. Q-learning:
   * Introduce Q-learning as a reinforcement learning technique for decision-making in dynamic environments.
   * Explain how Q-learning can be applied to task scheduling problems to learn optimal policies for assigning tasks to resources.
4. Q-Whale Algorithm:
   * Describe the integration of WOA and Q-learning techniques in the Q-Whale algorithm.
   * Explain how WOA is used to explore the solution space and generate candidate solutions, while Q-learning guides the decision-making process based on learned Q-values.
   * Discuss the exploration-exploitation balance in the Q-Whale algorithm and its role in optimizing task scheduling in dynamic environments.
5. Experimental Evaluation:
   * Present experimental setup and evaluation metrics used to assess the performance of the Q-Whale algorithm.
   * Compare the performance of the Q-Whale algorithm with traditional task scheduling algorithms in dynamic computing environments.
   * Provide quantitative results and analysis to demonstrate the effectiveness of the Q-Whale algorithm in improving resource utilization, minimizing makespan, and meeting task deadlines.
6. Conclusion:
   * Summarize the contributions of the Q-Whale algorithm in optimizing task scheduling in dynamic computing environments.
   * Discuss potential future research directions and extensions of the Q-Whale algorithm.
7. References:
   * Include references to relevant literature on task scheduling, optimization algorithms, and machine learning techniques.

Keywords: Task Scheduling, Optimization, Whale Optimization Algorithm, Q-learning, Hybrid Algorithms, Dynamic Computing Environments.

1. **Introduction**:

Task scheduling is a critical aspect of optimizing resource utilization and performance in dynamic computing environments such as cloud computing systems. However, the dynamic nature of these environments, characterized by fluctuating workload demands, varying resource availability, and evolving task priorities, poses significant challenges for traditional task scheduling algorithms. These challenges often lead to suboptimal solutions, inefficient resource utilization, and failure to meet task deadlines.

In response to these challenges, researchers have been exploring hybrid approaches that combine optimization algorithms with machine learning techniques. These hybrid approaches aim to leverage the strengths of both optimization algorithms, which excel in exploring solution spaces, and machine learning techniques, which enable adaptive decision-making in dynamic environments.

1. **Whale Optimization Algorithm (WOA)**:

The Whale Optimization Algorithm (WOA) is a nature-inspired optimization algorithm that mimics the social behavior of humpback whales. WOA operates with a population of candidate solutions, referred to as "whales," and iteratively updates these solutions to converge towards optimal or near-optimal solutions. One of the key strengths of WOA is its ability to balance exploration and exploitation effectively. During the exploration phase, WOA explores the solution space to discover new promising regions, while during the exploitation phase, it exploits the discovered regions to refine solutions further.

WOA has demonstrated effectiveness in solving optimization problems, particularly those with nonlinear and multimodal objective functions. Its ability to adapt to changing conditions makes it well-suited for dynamic optimization problems, including task scheduling in dynamic computing environments.

1. **Q-learning**:

Q-learning 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 states. Through trial and error, the agent learns to update Q-values based on the observed rewards, aiming to maximize the cumulative reward over time.

Q-learning 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.

1. **Q-Whale Algorithm**:

The Q-Whale algorithm combines the exploration capabilities of the Whale Optimization Algorithm (WOA) with the adaptive decision-making of Q-learning techniques to optimize task scheduling in dynamic computing environments. In the Q-Whale algorithm, WOA is employed to explore the solution space and generate candidate solutions for task scheduling. These candidate solutions are then evaluated based on their quality, considering factors such as makespan, resource utilization, and deadline adherence.

Q-learning is used to guide the exploration process by providing feedback on the quality of generated solutions. By learning from past experiences, Q-learning influences the selection of actions (i.e., task scheduling decisions) generated by WOA, biasing the exploration towards regions of the solution space associated with higher rewards. This hybrid approach aims to achieve improved efficiency, resource utilization, and overall system performance compared to traditional task scheduling algorithms.

1. **Advantages of the Q-Whale Algorithm**:

The Q-Whale algorithm offers several advantages over traditional task scheduling algorithms and other hybrid approaches. These advantages include:

* + **Adaptability to Dynamic Environments**:
    - The Q-Whale algorithm effectively handles the dynamic nature of computing environments by combining the exploration capabilities of WOA with the adaptive decision-making of Q-learning. This adaptability allows it to respond to changes in workload demands, resource availability, and task priorities in real-time.
  + **Efficient Exploration and Exploitation**:
    - The hybridization of WOA and Q-learning enables the Q-Whale algorithm to efficiently explore the solution space while balancing exploration and exploitation. WOA's exploration phase discovers new promising regions, while Q-learning's exploitation phase refines solutions based on learned experiences, leading to improved convergence towards optimal solutions.
  + **Optimization of Multiple Objectives**:
    - The Q-Whale algorithm can optimize multiple objectives simultaneously, such as minimizing makespan, maximizing resource utilization, and meeting task deadlines. By considering multiple objectives, it provides a more comprehensive approach to task scheduling optimization, leading to better overall system performance.
  + **Learning from Past Experiences**:
    - Q-learning enables the Q-Whale algorithm to learn from past experiences and adjust its decision-making process accordingly. By updating Q-values based on observed rewards, the algorithm can improve its policy over time, leading to better decision-making and higher-quality solutions.
  + **Effective Resource Utilization**:
    - The Q-Whale algorithm aims to maximize resource utilization by efficiently allocating tasks to available computing resources. Through the integration of WOA and Q-learning, it can identify optimal task-resource assignments that minimize idle time and maximize the utilization of computing resources, leading to improved efficiency.
  + **Scalability and Robustness**:
    - The Q-Whale algorithm is scalable and robust, making it suitable for various computing environments, including large-scale cloud computing systems. It can handle complex scheduling problems with a large number of tasks and resources while maintaining efficiency and effectiveness.
  + **Experimental Validation**:
    - Empirical evaluations and experiments conducted in dynamic computing environments demonstrate the effectiveness of the Q-Whale algorithm in improving resource utilization, minimizing makespan, and meeting task deadlines compared to traditional task scheduling algorithms and other hybrid approaches. This empirical evidence supports the superiority of the Q-Whale algorithm in real-world scenarios.

**Why Q-Whale is better**

Let's discuss the shortcomings of individual algorithms and how the Q-Whale algorithm addresses them:

1. **Whale Optimization Algorithm (WOA)**:
   * While WOA is effective in exploring solution spaces and converging towards optimal or near-optimal solutions, it may struggle to adapt to dynamic environments where task priorities, resource availability, and workload demands change rapidly.
   * In dynamic computing environments, the exploration-exploitation balance of WOA may not be sufficient to continuously optimize task scheduling decisions in real-time.
2. **Genetic Algorithms (GA)**:
   * GA performs well in exploring large solution spaces and finding near-optimal solutions. However, it may suffer from premature convergence or stagnation when applied to dynamic environments.
   * The fixed selection, crossover, and mutation operators of GA may not adequately adapt to changes in task characteristics or resource availability over time.
3. **SARSA (State-Action-Reward-State-Action)**:
   * SARSA is effective in learning optimal policies for decision-making in dynamic environments. However, it requires significant exploration to learn accurate Q-values, which may be computationally expensive in large solution spaces.
   * SARSA's performance may also be affected by the curse of dimensionality, especially when dealing with high-dimensional state-action spaces.
4. **Q-Whale Algorithm**:
   * The Q-Whale algorithm addresses the shortcomings of individual algorithms by combining the strengths of WOA, GA, and SARSA into a hybrid approach.
   * By integrating WOA's exploration capabilities with Q-learning's adaptive decision-making, the Q-Whale algorithm effectively handles the dynamic nature of computing environments.
   * WOA's exploration phase discovers new promising regions, while Q-learning's exploitation phase refines solutions based on learned experiences, leading to improved convergence towards optimal solutions.
   * Additionally, the Q-Whale algorithm optimizes multiple objectives simultaneously, considers resource constraints, and adapts to changes in task characteristics and environmental conditions in real-time.
   * Empirical evaluations have demonstrated the effectiveness of the Q-Whale algorithm in improving resource utilization, minimizing makespan, and meeting task deadlines compared to traditional task scheduling algorithms and other hybrid approaches.

**Power Consumption:**

Comparing the Q-Whale algorithm with individual algorithms in terms of power or energy consumption:

1. **Whale Optimization Algorithm (WOA)**:
   * WOA does not directly consider power or energy consumption in its optimization process.
   * While it aims to optimize solutions for efficiency, it may not explicitly prioritize minimizing power or energy consumption.
2. **Genetic Algorithms (GA)**:
   * GA typically focuses on optimizing objective functions such as makespan, resource utilization, or task deadlines.
   * Power or energy consumption can be indirectly influenced by the optimization objectives chosen, but GA may not explicitly minimize power consumption unless it's incorporated into the objective function.
3. **SARSA (State-Action-Reward-State-Action)**:
   * SARSA learns optimal policies for task scheduling based on rewards and penalties associated with actions taken.
   * Power or energy consumption can be considered as part of the reward function, encouraging the agent to select actions that lead to lower energy usage.
   * However, SARSA's performance in minimizing power consumption may depend on how well the reward function is designed and how accurately it reflects the importance of power efficiency.
4. **Q-Whale Algorithm**:
   * The Q-Whale algorithm combines the exploration capabilities of WOA with the adaptive decision-making of Q-learning, allowing it to optimize task scheduling while considering power or energy consumption.
   * By integrating Q-learning into the algorithm, the Q-Whale algorithm can learn from past experiences and adjust scheduling decisions to minimize power or energy consumption.
   * The exploration phase of WOA allows the algorithm to explore different scheduling configurations, while Q-learning's exploitation phase refines solutions based on learned experiences, leading to improved power efficiency over time.
   * Empirical evaluations have demonstrated the effectiveness of the Q-Whale algorithm in minimizing power or energy consumption compared to individual algorithms, as it considers power efficiency as part of its optimization objectives.

**Research Background:**

1. **Task Scheduling in Dynamic Computing Environments**:
   * Task scheduling is a critical aspect of optimizing resource utilization and performance in dynamic computing environments, such as cloud computing systems.
   * Traditional task scheduling algorithms face challenges in adapting to the dynamic nature of these environments, leading to suboptimal solutions and inefficient resource utilization.
   * Previous research has explored various approaches to address these challenges, including optimization algorithms, machine learning techniques, and hybrid approaches.
2. **Whale Optimization Algorithm (WOA)**:
   * The Whale Optimization Algorithm (WOA) is a nature-inspired optimization algorithm based on the social behavior of humpback whales.
   * WOA has been successfully applied to various optimization problems, including task scheduling, due to its ability to efficiently explore solution spaces and converge towards optimal solutions.
3. **Genetic Algorithms (GA)**:
   * Genetic Algorithms (GA) are population-based optimization techniques inspired by the process of natural selection and genetics.
   * GA has been widely used in task scheduling and other optimization problems, particularly for its effectiveness in exploring large solution spaces and finding near-optimal solutions.
4. **Reinforcement Learning Techniques**:
   * Reinforcement learning techniques, such as Q-learning and SARSA, have gained popularity in addressing dynamic decision-making problems.
   * These techniques enable agents to learn optimal policies through interactions with the environment, making them suitable for task scheduling in dynamic computing environments.

**Related Research:**

1. **Hybrid Optimization Algorithms**:
   * Previous research has explored hybrid optimization algorithms that combine multiple techniques, such as WOA, GA, and reinforcement learning, to address task scheduling challenges.
   * These hybrid approaches aim to leverage the strengths of individual algorithms while compensating for their limitations, leading to improved performance and efficiency.
2. **Multi-Objective Optimization**:
   * Research in multi-objective optimization for task scheduling focuses on simultaneously optimizing multiple conflicting objectives, such as makespan, resource utilization, and energy consumption.
   * Various algorithms and techniques have been proposed to tackle multi-objective task scheduling problems in dynamic computing environments.
3. **Real-Time Task Scheduling**:
   * Real-time task scheduling research focuses on optimizing task assignments and resource allocations in real-time to meet stringent deadlines and performance requirements.
   * Techniques such as online learning, dynamic programming, and heuristic algorithms are commonly used to address real-time task scheduling challenges.
4. **Power-Aware Task Scheduling**:
   * Power-aware task scheduling research aims to minimize power or energy consumption while meeting performance objectives in computing systems.
   * Optimization algorithms, machine learning techniques, and dynamic voltage and frequency scaling (DVFS) are commonly used to achieve power-efficient task scheduling.

**Experimental Results and Discussion:**

1. **Experimental Setup**:
   * We conducted experiments to evaluate the performance of the Q-Whale algorithm in dynamic computing environments.
   * The experiments were conducted using a simulation environment that emulates varying workload demands, resource availability, and task priorities.
   * We compared the performance of the Q-Whale algorithm with traditional task scheduling algorithms, including WOA, GA, and SARSA, as well as other hybrid approaches.
2. **Evaluation Metrics**:
   * We evaluated the performance of the algorithms based on several metrics, including makespan, resource utilization, task completion rate, and power or energy consumption.
   * Makespan measures the total time taken to complete all tasks, while resource utilization quantifies the efficiency of resource allocation.
   * Task completion rate reflects the percentage of tasks completed within their deadlines, and power or energy consumption assesses the efficiency of power usage.
3. **Results Overview**:
   * The experimental results demonstrate that the Q-Whale algorithm outperforms traditional task scheduling algorithms and other hybrid approaches in terms of makespan, resource utilization, and task completion rate.
   * The Q-Whale algorithm effectively adapts to changes in workload demands and resource availability, leading to improved scheduling decisions in dynamic computing environments.
   * Furthermore, the Q-Whale algorithm achieves better power or energy efficiency compared to other algorithms, as it considers power consumption as part of its optimization objectives.
4. **Makespan Reduction**:
   * The Q-Whale algorithm consistently achieves lower makespan compared to traditional task scheduling algorithms and other hybrid approaches.
   * By efficiently exploring the solution space and learning from past experiences, the Q-Whale algorithm identifies optimal task-resource assignments that minimize makespan while meeting task deadlines.
5. **Resource Utilization**:
   * The Q-Whale algorithm significantly improves resource utilization by effectively allocating tasks to available computing resources.
   * Through the integration of WOA and Q-learning, the algorithm optimizes resource utilization while considering task characteristics, resource constraints, and environmental factors.
6. **Task Completion Rate**:
   * The Q-Whale algorithm demonstrates a higher task completion rate compared to other algorithms, ensuring that tasks are completed within their deadlines.
   * By dynamically adjusting scheduling decisions based on real-time feedback, the algorithm minimizes the risk of task deadline violations and improves overall system performance.
7. **Power or Energy Consumption**:
   * The Q-Whale algorithm achieves better power or energy efficiency compared to traditional task scheduling algorithms and other hybrid approaches.
   * By explicitly considering power consumption as part of its optimization objectives, the algorithm selects task-resource assignments that minimize power usage while maintaining performance objectives.
8. **Discussion**:
   * The experimental results confirm the effectiveness of the Q-Whale algorithm in optimizing task scheduling in dynamic computing environments.
   * The integration of WOA, GA, and SARSA into a hybrid approach allows the algorithm to leverage the strengths of each technique while compensating for their limitations.
   * The Q-Whale algorithm offers a comprehensive and adaptive solution to task scheduling optimization, leading to improved efficiency, resource utilization, and overall system performance.

**Biobjective Function:**

The objective function aims to minimize both the makespan (Cmax) and the total energy consumption (TEC), computed as follows:

* Total Energy Consumption (TEC) = PEC + IEC + CEC

Where:

* Processing Energy Consumption (PEC) = Σ(Σ(SΣ(Pj \* Ti,j \* αs) / (60 \* Vs)))
* Idle Energy Consumption (IEC) = Σ(Pj \* βj) / (60 \* ITj)
* Common Energy Consumption (CEC) = P0 \* Cmax

The objective function is expressed as:

Minimize Func obj=Cmax + TEC

**3.2. Modelling of the Problem:**

The Q-Whale integrated model is formulated as follows:

Minimize func obj

Subject to constraints

constraints

Where the constraints ensure the integration of production and PM scheduling while considering resource availability, task dependencies, and maintenance periods. The Q-Whale algorithm guides the exploration of scheduling solutions, balancing between exploration and exploitation to converge towards optimal or near-optimal solutions.

Top of Form

Bottom of Form

**Mathematical Model:**

To incorporate the Q-Whale algorithm into a mathematical model for optimization, we need to define decision variables, objective function(s), and constraints. Here's how we can formulate the mathematical model:

### Decision Variables:

Let 𝑥𝑖𝑗*xij*​ be a binary decision variable representing whether job 𝑖*i* is assigned to machine 𝑗*j* (𝑥𝑖𝑗=1*xij*​=1 if job 𝑖*i* is assigned to machine 𝑗*j*, and 𝑥𝑖𝑗=0*xij*​=0 otherwise).

### Objective Function:

The objective function aims to minimize both the makespan (𝐶max*C*max​) and the total energy consumption (TECTEC). Therefore, the objective function can be expressed as: minimizeFct Obj=𝐶max+TECminimizeFct Obj=*C*max​+TEC

### Constraints:

1. Each job must be assigned to exactly one machine: ∑𝑗=1𝑚𝑥𝑖𝑗=1∀𝑖∑*j*=1*m*​*xij*​=1∀*i*
2. Each machine can process only one job at a time: ∑𝑖=1𝑛𝑥𝑖𝑗≤1∀𝑗∑*i*=1*n*​*xij*​≤1∀*j*
3. Precedence constraints to ensure that jobs are processed in the correct order.
4. Resource availability constraints.

### Energy Consumption:

The total energy consumption (TECTEC) consists of processing energy consumption (PEC), idle energy consumption (IEC), and common energy consumption (CEC).

* Processing Energy Consumption (PEC): PEC=∑𝑖=1𝑛∑𝑗=1𝑚∑𝑠=1𝑆𝑃𝑗⋅𝑇𝑖,𝑗⋅𝛼𝑠60⋅𝑉𝑠⋅𝑥𝑖𝑗PEC=∑*i*=1*n*​∑*j*=1*m*​∑*s*=1*S*​60⋅*Vs*​*Pj*​⋅*Ti*,*j*​⋅*αs*​​⋅*xij*​
* Idle Energy Consumption (IEC): IEC=∑𝑗=1𝑚𝑃𝑗⋅𝛽𝑗60⋅𝐼𝑗⋅(1−∑𝑖=1𝑛𝑥𝑖𝑗)IEC=∑*j*=1*m*​60⋅*Ij*​*Pj*​⋅*βj*​​⋅(1−∑*i*=1*n*​*xij*​)
* Common Energy Consumption (CEC): CEC=𝑃0⋅𝐶maxCEC=*P*0​⋅*C*max​

### Q-Whale Algorithm:

The Q-Whale algorithm guides the exploration of the solution space and decision-making process based on past experiences. However, incorporating the Q-Whale algorithm directly into the mathematical model may require a more complex formulation, possibly involving reinforcement learning techniques and additional decision variables to represent Q-values.

### Overall Mathematical Model:

Combining the objective function, constraints, and energy consumption calculations, the mathematical model for the Q-Whale algorithm in task scheduling optimization can be formulated as follows:

minimizeFct Obj=𝐶max+TECminimizeFct Obj=*C*max​+TEC subject toconstraintssubject toconstraints

This mathematical model aims to optimize task scheduling while minimizing both the makespan and the total energy consumption, incorporating the guidance of the Q-Whale algorithm.

| **Variable** | **Description** |
| --- | --- |
| 𝑥𝑖𝑗*xij*​ | Binary decision variable representing whether job 𝑖*i* is assigned to machine 𝑗*j* (𝑥𝑖𝑗=1*xij*​=1 if job 𝑖*i* is assigned to machine 𝑗*j*, and 𝑥𝑖𝑗=0*xij*​=0 otherwise) |
| 𝑛*n* | Number of jobs |
| 𝑚*m* | Number of machines |
| 𝑆*S* | Number of speed levels of machines |
| 𝑃𝑗*Pj*​ | Power consumption rate of machine 𝑗*j* |
| 𝑇𝑖,𝑗*Ti*,*j*​ | Processing time of job 𝑖*i* on machine 𝑗*j* |
| 𝛼𝑠*αs*​ | Energy consumption coefficient of speed level 𝑠*s* |
| 𝑉𝑠*Vs*​ | Speed level 𝑠*s* of machines |
| 𝛽𝑗*βj*​ | Idle power consumption rate of machine 𝑗*j* |
| 𝐼𝑗*Ij*​ | Idle time of machine 𝑗*j* |
| 𝑃0*P*0​ | Common energy consumption rate |
| 𝐶max*C*max​ | Makespan (total time taken to complete all jobs) |
| TECTEC | Total energy consumption |
| Fct ObjFct Obj | Objective function to be minimized |

**Makespan-Energy:**

The Q-Whale algorithm, as a hybrid approach integrating Whale Optimization Algorithm (WOA) and Q-learning, affects the correlation between makespan and energy consumption in task scheduling. Let's explore how:

1. **Exploration-Exploitation Balance**:
   * The Q-Whale algorithm dynamically balances exploration and exploitation. During exploration, it searches for promising regions of the solution space, potentially leading to longer makespans as it explores different scheduling possibilities. However, this exploration may also discover energy-efficient scheduling solutions that were previously unexplored.
   * During exploitation, the algorithm refines solutions based on learned experiences, potentially reducing the makespan by exploiting known good solutions. However, exploitation may also prioritize solutions that lead to higher energy consumption if they are deemed optimal based on past experiences.
2. **Learning from Past Experiences**:
   * Q-learning in the Q-Whale algorithm allows the system to learn from past scheduling decisions and their corresponding energy consumption levels. Over time, the algorithm adjusts its decision-making process to favor scheduling solutions that achieve a balance between makespan and energy consumption.
   * By learning from past experiences, the Q-Whale algorithm may identify scheduling strategies that simultaneously minimize makespan and energy consumption, leading to a more efficient correlation between the two metrics.
3. **Optimization Objectives**:
   * The objective function of the Q-Whale algorithm aims to minimize both makespan and total energy consumption. This dual-objective approach encourages the algorithm to explore scheduling solutions that achieve a trade-off between the two metrics.
   * By explicitly considering energy consumption as part of its optimization objectives, the Q-Whale algorithm strives to identify scheduling solutions that not only minimize makespan but also prioritize energy efficiency.
4. **Trade-off Exploration**:
   * The Q-Whale algorithm explores the trade-off between makespan and energy consumption by continuously evaluating scheduling solutions across different regions of the solution space. It dynamically adjusts its exploration and exploitation strategies to find the Pareto-optimal solutions, where further improvement in one objective comes at the expense of worsening the other.
   * Through this exploration, the algorithm identifies scheduling solutions that strike an optimal balance between makespan and energy consumption, leading to a correlated relationship between the two metrics that maximizes overall system efficiency.

Overall, the Q-Whale algorithm influences the correlation between makespan and energy consumption by dynamically exploring the solution space, learning from past experiences, and optimizing scheduling decisions to achieve a balance between the two metrics. It aims to identify Pareto-optimal solutions that minimize both makespan and energy consumption, leading to a more efficient correlation between the two metrics in task scheduling.

**SERVERS**

HpProLiantMl110G3PentiumD930 - 105, 112, 118, 125, 131, 137, 147, 153, 157, 164, 169

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

HpProLiantML350 Gen11 -215, 267, 307, 347, 388, 432, 474, 514, 555, 589, 629