Task Scheduling and Resource Allocation Based on Ant-Colony Optimization and Deep Reinforcement Learning

Ulysse Rugwiro[1], Chunhua Gu[2], Weichao Ding[3]

School of Information Science and Engineering East China University of Science and Technology, China
ulysserugwiro@live.com, chgu@ecust.edu.cn, weichaoding@126.com

Abstract

Cloud computing has become a significant aspect of today's rapidly growing technology, accessing as it does a large number of servers, given users' constant need to access their data efficiently and quickly. Cloud computing providers can flexibly place a user's task into an appropriate virtual machine and allocate the resource to the tasks for proper execution. However, user tasks can take a long time to complete the execution when the required resources are not available on the server. To overcome this problem, we propose a task scheduling and resource allocation model based on Hybrid Ant Colony Optimization and Deep Reinforcement Learning. In this article, our goal is to minimize the overall task completion time and improve the utilization of idle resources. The task scheduling was performed by constructing a Binary In-order Traversal Tree using weighted values. We then introduced a Deep Reinforcement Learning (DRL) algorithm to reduce space complexity by splitting resources into state space and action space. A state space will contain idle resources, which are used in task allocation. Then the scheduled task will search the resources based on Ant Colony Optimization. When it finds an optimal resource, it will allocate it to the task, and the server will put the allocated resources into action space. If the VM is overloaded, migration is performed. We simulated the proposed algorithm using CloudSim and evaluated the performance in terms of task completion time and resource utilization. Our proposed work evaluation shows mitigation of the above-described problems and illustrates the reduction of waiting time and improvement in idle resource utilization.

Keywords: Task scheduling, Resource allocation, Ant Colony Optimization, Deep Reinforcement Algorithm

1 Introduction

In cloud computing, there are hundreds of thousands of users who invoke their tasks into the cloud for resource allocation. Task scheduling plays a significant role in improving the performance of cloud computing. Scheduling is responsible for optimal resource selection for a given task, considering its parameters. Users' tasks are buffered in a queue manager, then the tasks are split into queues (urgent queue and waiting queue) based on their deadlines [1], by segregating tasks according to their deadline; the waiting time will be reduced, and the urgent queue will get the highest priority for execution.

A Cloud workflow scheduling in a multi-tenant environment [2] based on the schedule gap (i.e., a time of an idle CPU) experiments with the idea of selecting a better schedule position according to the schedule gap, when a new task arrives. The conventional workflow scheduling algorithms proposed are Easy Backfilling, First Come First Served (FCFS) and Minimum Completion Time (MCT). One of the other hot topics in cloud computing is parallel computing. Scheduling of parallel jobs presents a Shortest Job First (SJF) Model [3]. It starts by checking the minimal resource available in the data center, and then jobs are scheduled based on available resources. It is performed based on Task Placing and Resource Allocation Algorithm (TAPRA), which uses three phases; Stochastic workloads scheduled considering multiple QoS constraints [4] which build around Cross Entropy based Scheduling Scheme (CESS) which optimizes the accumulative QoS and sojourn time of all the tasks. CESS will assign the Probability Density Function (PDF) array to each task and generate samples. These samples are compared with QoS and the sojourn time (waiting time), updating the elite sample in each iteration if necessary. There are many more approaches that were proposed over the last few years regarding task scheduling and the allocation of the resources. Figure 1 gives a high overview concept of task scheduling and resource allocation.

Power consumption has been the focus in recent years; it has become critical to analyze how the power consumed by Data centers can be reduced. A Dynamic Power Resource Allocation (DPRA) built around Cloud Resource Requirement Module (CRRM), Cloud Task Requirement Module (CTRS), Cloud Resource Creation Module (CRCM), Cloud Resource Power

*Corresponding Author: Ulysse Rugwiro; E-mail: ulysserugwiro@live.com
DOI: 10.3966/160792642019092005013

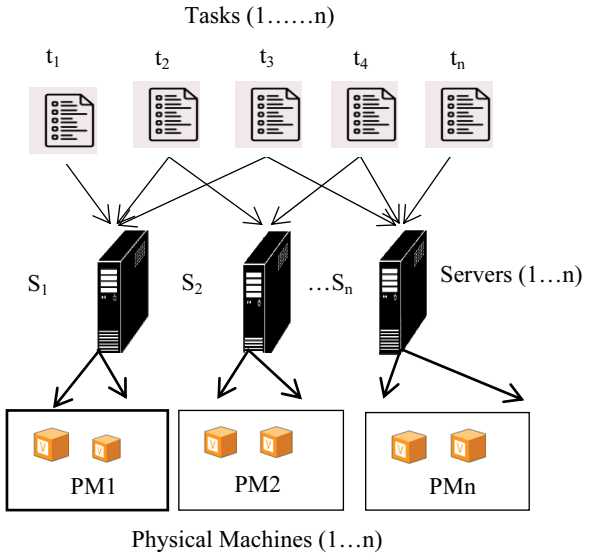

Figure 1. Task scheduling and Resource allocation model

Module (CRPM), and Cloud Resource Allocation Module (CRAM) is addressed in [5]. It uses Particle swarm optimization (PSO) algorithm to deploy VMs into PMs with dynamic resource allocation, and migration is performed using Cloud Resource Migration Module (CRMM). A MapReduce Constraint Programming based Resource Management algorithm (MRCP-RM) technique for task scheduling and resource allocation [6] describes how the jobs are allocated to an optimal resource by map, shuffle and reduce methods. It is Implemented on the strength of Hadoop constraint programming-based resource allocation HCP-RM, which performs match marking and scheduling using CPLEX java API to create and solve an Optimization Programming Language (OPL) model. Delay aware flow scheduling (DAFS) [7] proposes two types of scheduling disciplines; Work Conserving (WC) and Non-Work Conserving (NWC). WC is the process where a packet is maintained in the queue after deadline expiration, and eventually gets served. NWC is the process where a packet is immediately dropped from the queue upon deadline expiry. DAFS is used to decrease the deadline mismatch, blocking probabilities and improving the average throughput.

Energy efficient, VM prediction and migration framework [8] reduces an unpredicted overload, minimizes migration overhead and improves resource utilization by dedicating separate resource predictor module to each virtual machine. A resource predictor module consists of two predictors that monitor and collect the VM's memory and usage traces to predict the VM's future CPU and memory demands. Energy efficient hierarchical resource management [9] consists of three phases; Global cloud server, Local ISP server and Gateway server connected through the internet, and the cluster head formed based on battery level. The fuzzy rule-based scheme is applied to minimize the delay, and a foglet selection phase is developed based on the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS).

1.1 Contributions

The work contribution is summarized as follows:

- Managing resources by scheduling the user's tasks with Binary In-order Traversal Tree using weighted values of each task.
- In cloud environment, the resource scarcity happens more often. To reduce the complexity, we proposed a hybrid resource allocation technique; a combination of Deep Reinforcement Learning (DRL) and Ant-Colony Optimization (ACO). The DRL is introduced to improve the utilization of idle resources by splitting resources into Action space and State Space in cloud environment. DRL based resource management minimize resource usage for large-scale cloud environment with very huge number of servers that obtain numerous requests from users per day. ACO is implemented to map tasks to appropriate virtual machines once the condition is satisfied
- Resource predictor and Overload predictor will calculate the CPU, memory and I/O usage of Virtual Machine to minimize the resource overload on a Physical Machine.

The rest of the paper is organized as follow: **Section II** talks about previous work; **Section III:** describes the problem formulation; **Section IV:** presents the proposed work; **Section V**: Experimental analysis; **Section VI:** concludes the paper.

2 Related Work

Resource scheduling in cloud has been studied in the past few years. Previous work, Zhang and Zhou [10] presented a Dynamic task scheduling with VM being pre-created to maximize the task scheduling performance and ensures that the tasks dynamically match the concrete VMs. Taking the results of task classifier into consideration, the proposed process involves a matcher, which matches tasks to VMs. if the resources are busy, the matched tasks are kept in the waiting queue until the resource becomes idle; the process need more time for task execution as a result the completion delay. In [11], Tram et al. proposed dynamic flow scheduling with uncertain flow duration in optical data centers by introducing the Markov decision scheduling process to estimate the flows' termination and expected revenue. They in addition developed a scheduling algorithm for the known and unknown flow service time. They then introduced a congestion factor for paired ToRs which determines the traffic flow scheduling order but calculating congestion factor for each pair leads to high time consumption in the scheduling process. In [12], Zuo et

al. developed a Multi-Objective Scheduling Ant Colony Optimization (MOSACO) which main objective was to optimize the limited resources available in cloud environment. They considered the task completion time, the number of deadline violations, and the degree of resource utilization by using Ant Colony Optimization algorithm. The process focuses on time-first and cost-first strategies to be able to provide the highest optimality. In [13], Zhao et al. presented the task scheduling approach in a heterogeneous cloud which consists of edge cloud and remote cloud (single user and multiple users' cases). The investigated issue is more on the limited computation resources available in Edge cloud which results in QoS degradation. The solution coordinated the heterogeneous cloud which includes the edge cloud and the remote cloud; The traffic load needs to be optimized due to data transmission delay in this process. When traffic load is heavy, more resources map to a user with shorter data transmission delay. When traffic load is light, more resources map to a user with longer data transmission delay; with the assistance of remote cloud the probability of bounded task delay can be improved by 40%.

Resource allocation in cloud environment is one of the keys for large-scale for cloud application; it quick because one of the hot research areas in recent years. In [14], Bey et al. introduced a new task scheduling strategy for resource allocation to minimize the task execution time and maximize the resource exploitation in a cloud environment. With this approach, an initial solution is applied to find the clusters with minimum completion time and the jobs are assigned sequentially to the cluster using Min-Min algorithm, which is suitable only for a minimum number of tasks and resources. In [15], Liu et al., discussed an issue of the uncertainty of the task runtime because the tasks real-time execution might not be the same as their given or evaluated time. To overcome this problem, they introduced a Path Cut (PC) algorithm to map all scientific workflow tasks to a VM instance by determining the critical path and the task duplication used to replicate tasks in-order to eliminate the deviation caused by uncertain task runtime. Additionally, the least resource appending algorithm is used to remap the tasks and add a new virtual machine in the process. The critical path leads to a greater time consumption due to the high completion time of tasks in Path Cut algorithm.

In order to lower the power consumption and in the process improve the coefficient of resource utilization of current cloud computing systems. In [16], Xu et al. presented a resource pre-allocation strategy to minimize the power consumption by using Resource Allocation based on Probabilistic Matching (RA-PM) and Resource Allocation based on Improved Simulated Annealing (RA-ISA). Cubic exponential smoothing method predicts the future workloads and pre-allocates the appropriate resources. However, the gap between the prediction and actual workloads leads to resource wastage in the allocation process. They also presented a Task scheduling and resource allocation strategy based on the idea of Pareto based Fruit Fly Optimization Algorithm (PFOA). The PFOA have two search procedures; first is the smell-based search stage which consists of a critical path-based search operator, a re-assigning operator and a random search operator which are adopted to explore the promising region. Second is the Vision-based search stage, a non-dominated sorting technique is employed to sort the temporary population, and the top N individuals survive. It has a computational complexity to generate operators to task scheduling and resource allocation process based on PFOA.

3 Problem Description

Resource allocation and task scheduling are the key challenges in cloud computing that constrain the energy of underlying distributed hardware in cloud data centers. Several researchers have concentrated on scheduling and resource allocation by developing various approaches/techniques in cloud environment. In [3], the SJF scheduling policy was proposed, which processed with several iterations. In this process, the Minimum Available Resources was estimated by using the total amount of available computing energy in a cloud data center. In the SJF scheduling, the shortest jobs have a high level of priority so that jobs are assigned to resources in advance. Therefore, the longest jobs would be waiting a long time for the completion of task. In [2], the CWSA was proposed for multi-tenant task. The tasks were submitted to service queue and sorted according to the deadline priority. The scheduler was involved to schedule the workflows based on three policies.

- Scheduling policy 1: Scheduling was done by the FCFS algorithm with respect to the arrival of tasks.
- Scheduling policy 2: Scheduling was performed by EASY backfilling by considering the deadline of tasks.
- Scheduling policy 3: Scheduling was done by the MCT method, which was based on minimum completion time of tasks.

With the FCFS, incoming tasks were sorted by the scheduler in the arriving order of tasks. Based on this order, if the required resources are available to execute the first task, that is immediately processed. Otherwise, the tasks wait until the resource become available for tasks that leads to increased waiting time for allocating the tasks based on resources. In [19], the utility prediction aware VM consolidation model was applied periodically to adapt and optimise the VM placement according to the workload. This work aims to migrate the overloaded VMs to other PMs. If at least one

resource i.e., CPU or memory exceeds total capacity, PMs will be considered as overloaded. Therefore, the number of VM migration is high in this process.

To overcome the above-mentioned problems, we have proposed task scheduling and resource allocation in cloud environment. The user submits a set of independent tasks to the cloud server, $T = \{T_1, T_2, T_3, ..., T_n\}$. T_n is the 'nth' independent task of the user. The data centre is used to execute the tasks, which maintain the different servers. Each server in the data centre has different efficiencies for executing the task. Then, the resource allocation strategy indicates the amount of computing resource allocated to each task.

We denote the set of VMs by $VM = \{VM_1, VM_2, VM_3, ..., VM_n\}$. We formulate the addressed task scheduling and resource allocation process in our work as follows

$$\begin{aligned} &\text{Minimize } T_C \\ &\text{If } D_{T_i} < D_m \text{ then put into short } Q \\ &\text{If } D_{T_i} < D_m \text{ then put into log } Q \\ &\text{Maximize } VM_U \end{aligned} \tag{1}$$

The objective of Eq (1) is to minimise the task completion time (T_C). The tasks are assigned into short queues when the deadline of the task (D_{T_i}) is less than the deadline median value (D_m); additionally, the tasks are assigned to a long queue when the deadline of tasks is greater than the deadline median value. The idle VM resource utilisation (VM_U) is maximised by splitting the resources into action space and state space.

4 Proposed Work

To address some of the challenges in the existing works, we proposed task scheduling and resource allocation based on the hybrid ACO and DRL algorithm. This hybrid algorithm is designed to maximize the idle VM resource utilization and minimize the task completion time. Figure 2 implies the overall architecture of the proposed work, which includes various phases such as task queuing, task scheduling and resource allocation, overloaded prediction, and VM migration. Initially, the user tasks are submitted in the global queue. By considering the deadline, tasks are divided into long queue and short queue. Further, queued tasks are scheduled by using the Binary In-order Traversal Tree based on the weighted values of task.

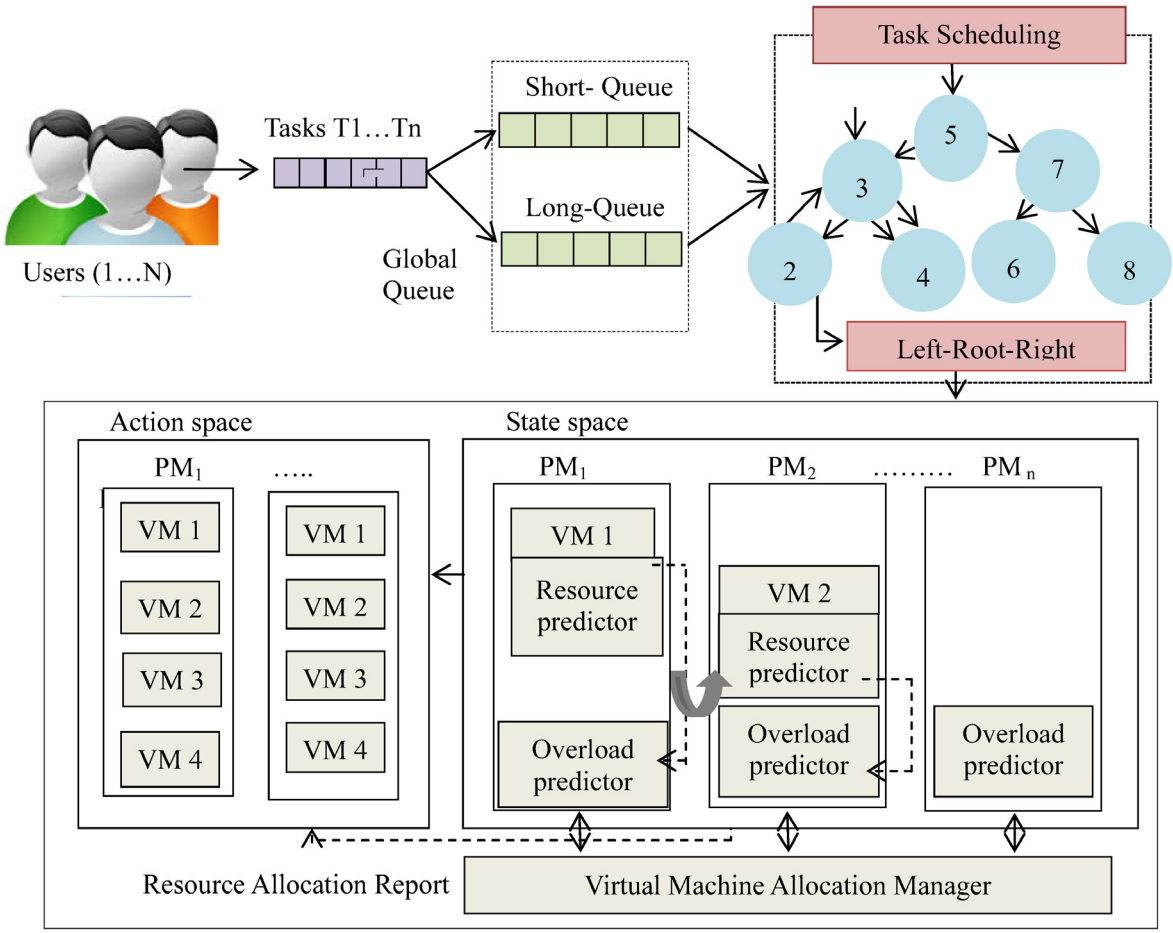

Figure 2. Detailed task scheduling and resource allocation

Resources are categorized into action space and state space to improve the utilization of resource. Action space is comprised of allocated physical machines and state space is comprised of idle physical machines in which tasks are to be allocated. Each VM has a resource predictor and overload predictor that simply shows the intensives (CPU, memory, and I/O) of resources. Further, the scheduled tasks are requested for the required resources, it searches the resource based on ACO, and when it finds optimal VM, it allocates it. The overloaded predictors of all PMs in the state space are connected with the VMA manager, which monitors and manages the resource allocation process. By using the overloaded predictor, overloaded resources are easily predicted so the number of VM migration is reduced. Then, the overloaded VM in one PM migrates to another PM and then the idle PMs are switched to sleep mode to save power. The entire proposed work is elaborated in the following subsections.

4.1 Task Queuing using Median Deadline

All the incoming tasks at time are submitted to global queue (G_Q), and then they are divided into short queue and long queue with respect to its deadline. Deadline of tasks are represented as $D_T = d_{T_1}, d_{T_2}, d_{T_3}, ..., d_{T_n}$ with certain time limit period by which tasks are divided into two queues. Further, the deadline median value (D_m) of overall task is determined for task queuing. A short queue (S_Q) is comprised of tasks whose deadline is lower than the deadline median value whereas long queue (L_Q) is comprised of tasks whose deadline is higher than the deadline median value. With these queues, the tasks are scheduled by using the Binary In-order Traversal tree.

Pseudocode for Task Queuing

Input: Tasks (1…..n)
Output: S_Q and L_Q
1. Begin
2. $G_Q \leftarrow$ Tasks $T = \{T_1, T_2, T_3, ..., T_n\}$..
3. Task deadline $\rightarrow D_T = \{d_{T_2}, d_{T_3}, ..., d_{T_n}\}$
4. Calculate deadline median value D_m
5. If ($D_{Ti} < D_{m)}$
   {Put the task in short Q}
   Else
   {Put the task in long Q}
6. End if
7. End

The Pseudocode defines the steps of task queuing. The deadline of each task is compared with the median value. As the goal is to minimize the waiting time, smaller deadline tasks are placed in the short queue, which gets high priority. Tasks with longer deadlines are placed in the long queue.

4.2 Task Scheduling Using Binary In-order Traversal Tree

We schedule tasks with Binary In-order Traversal Tree by calculating weighting values for each task. To calculate the weighted value, we consider the following parameters of each task; task length (l), tardiness (t), makespan (m) and slack time (s).

Task scheduling by tree provide a perfect schedule with a guaranteed quality by constructing binary search tree. First, we look into the requested user tasks, then process it from the tree for execution.

Definition 1. Each node (A) in a tree (T) has a key (Key (A)). The set of nodes rather than the root node have a parent P (A). The elements for tree T may have a left child node (Left (A)) and/or a right child node (Right (A)).

The pattern is that for any node A, for all set of nodes B in the left subtree of A, Key(B) ≤ Key(A). For all nodes B in the right subtree of A Key(B) ≥ Key (A)

Tasks from the short queue $S_Q = \{T_1, T_2, \ldots T_n\}$ are moved towards the scheduler for scheduling. Each task consists of its corresponding task length; represented as, $L = L_1, L_2, L_3, ..., L_n$.

Makespan value (i.e. completion time of task) represented as, $M = m_1, m_2, m_3, ..., m_n$. Tardiness value, defined as arrival time between tasks (i.e., delay). Specifically, it can be defined as the difference between a late job's due data and its completion time.

It can be represented as, $T = t_1, t_2, t_3, ..., t_n$. The slack time is the amount of time that the task can be delayed without missing its deadline and the order of tasks according to non-decreasing slack-time. It can be expressed as, $S = s_1, s_2, s_3, ..., s_n$; The weighted value calculated using task length, makespan, tardiness and slack time represented as,

$$\text{Weight of task, } w = \frac{\Sigma l_i t_i m_i s_i}{\sqrt{\Sigma l_i m_i^2 \Sigma l_i t_i^2 \Sigma l_i s_i^2}} \quad \textbf{(2)}$$

Where **w** - Weight of task; $\boldsymbol{l_i}$ – task length of i^{th} task; m_i - makespan of i^{th} task, t_i - tardiness of i^{th} task; s_i – slack time of i^{th} task. However, the task scheduling algorithm places restrictions on the execution order of the operations before scheduling. We alleviate this problem by representing all the weighted values in a tree. Weight of each task is placed in the nodes of a Binary In-Order Traversal Tree. The process of Binary in-order Trees has the first incoming weighted value placed in the Root node. Afterwards, the upcoming tasks have their weighted values compared to the root node. If the task's weighted value is greater than the root value, it is placed on the right node of the root. If the task's weighted value is less than the root value, it is placed on the left node of the root. Then the tasks are

scheduled by In-order traversal, where a task in the left node is first scheduled, then the root node, and finally the right node is scheduled, i.e., Left Node-Root Node-Right node. Assume weighted values of tasks in the short queue are TW = {9,2,7,3,5,4,1,8,6}. These weighted values of the tasks are scheduled by using Binary In-order Traversal Tree as Figure 3.

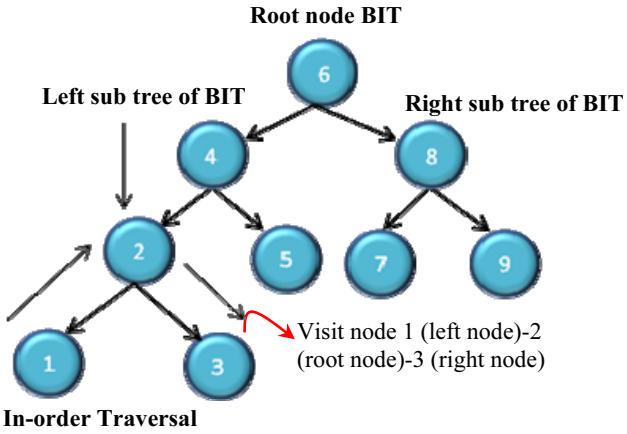

Figure 3. Task scheduling using Binary In-order Traversal Tree (BIT)

Figure.3 Shows the scheduling process of the incoming weighted value w=6 with the task placed in the root node. Then w=4 is compared with the root value and set to the left of the root node, because the value is less than the root weighted value. Next w=8 is compared with the root value and set to the right of the root node because the value is greater than the root value.

The process is repeated, until all the weighted values of tasks are placed in the binary tree. In Binary In-order traversal, nodes are scheduled in the order of Left-Node; Root-Node; Right-Node. Finally, the scheduled tasks are in the order of TW = {1, 2, 3, 4, 5, 6, 7, 8, 9}. The process can be represented by the following pseudocode.

Pseudocode for Task Scheduling

Input: SQ and LQ
Output: Scheduled task
1. Begin
2. SQ = {T1, T2,…Tn}
3. Get L, M, T, S
4. calculate Task's weighted values eqn. (2)
5. $T_W = T_{W_1}, T_{W_2}, ..., T_{W_n}$
6. Assign
   T_{W_1} = root node
7. If ($T_{W_i} > T_{W_j}$)
   {Place in left node}
   Else
   {Place in right node}
8. Schedule Left node- Root node – Right node
9. End

4.3 Hybrid Resource Allocation

Resource allocation plays a significant role in assigning the tasks to virtual machines. This is also a challenging task, because of the assignment of each task to a VM with the available resources on the physical machine. Our proposed Hybrid Ant Colony Optimization with DRL Algorithm method improves the utilization of idle resources for efficient resource allocation. Among the various search algorithms, DRL and ACO are very similar. The ACO algorithm employs a colony of agents (ants) to the optimization problem and each ant lays some quantity of pheromone on the paths they have taken. Similar to ACO, DRL algorithms employ agents (RL agents) to learn about the optimal policy that will increase the overall reward in the long run. The optimal policy is improved according to the state/action values. Moreover, both ACO and DRL algorithms have exploitation and exploration trade-off due to the searching nature.

A DRL is primarily designed to generate best long-term decisions by learning from the varying environment; like a pattern of user's request. It is suitable for training enormous user requests with fast convergence speed. The major reasons for DRL in resource management are; the efficiency of task execution; Secondly, learning from a particular user tasks instead of FIFO order step gives higher efficiency and lastly, it eliminates divergence and remarkably low runtime and energy cost.

The Automatic decision-making approaches like reinforcement learning (RL) were introduced to solve the resource allocation issue actively in the cloud computing environment. Adopting an ACO, loss of each action during the training procedure and in addition to obtain the optimal policy, it is essential to identify the actions, reward functions and states. However, a complete cloud resource allocation framework displays high dimensions in state and action spaces, which prevents the use of traditional RL techniques.

Firstly, the resources are divided into action space and state space. The action space consists of the virtual machines already allocated to the tasks. State space comprises of unallocated VMs, and these idle resources will be allocated to the scheduled tasks later. Each virtual machine's utilization is calculated by using their corresponding CPU intensive, Memory intensive and I/O intensive utilizations.

$$VM_U = U_{VM_i}^{CPU} + U_{VM_i}^{M} + U_{VM_i}^{I/O} \quad \textbf{(3)}$$

Where,
$VM_{U\_}$ Virtual machine utilization
$U_{VM_i}^{CPU}$ - CPU utilization of ith virtual machine
$U_{VM_i}^{M}$ - Memory utilization of ith virtual machine
$U_{VM_i}^{I/O}$ - I/O utilization of ith virtual machine

If the virtual machines in the state space are fully allocated, it transferred to the action space which is shown in Figure 4.

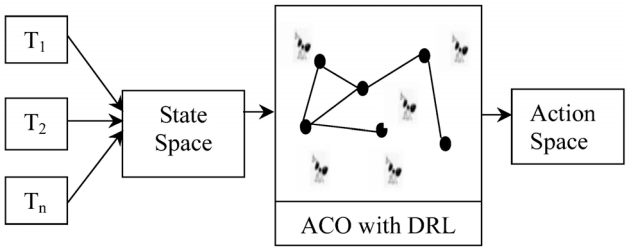

Figure 4. Resource allocation

4.3.1 The Use of Deep Reinforcement Learning for Resource Management

State space. In DRL, the CPU, Memory and I/O intensives of each virtual machine are taken into consideration to split the resources into state and action space.

We denote task T's arrival time T_t, S^{T_t} as the union of the server cluster state at task T's arrival time $S_C^{t_T}$ and the task T's state S_T, i.e., $S^{t_T} = S_C^{t_T} \cup S_T$. The entire server can be equally divided into K groups, G1…. Gk. We define the state of servers in group G_K at time t as g_k^t. We also define the utilization requirement of resource type R of task T by U_{TR}, and the utilization level of server m at time t as U_{mR}^t. Therefore, the system state S^{T_t} of the DRL based cloud service allocation tier can be represented using U_{TR} and $U_{mR}^{t_T}$ as follows,

$$S^{t_T} = [S_C^{t_T}, S_T] = [g_1^{t_T}, ..., g_k^{t_T}, S_T] \quad (4)$$

$$= [U_{11}^{t_T}, ..., U_{1|D|}^{t_T}, ..., U_{|M||D|}^{t_T}, U_{T_1}, ..., U_{T|D|}, d_T] \quad (5)$$

Where d_T is the (estimated) task duration. The state space consists of all possible states of VM and it has a high dimension.

Action space. The action space of the DRL for cloud resource allocation is defined as the allocated virtual machines. The action space for a cluster with N servers is defined as follows,

$$A = \{a \mid a \in \{1, 2, ... \mid N \mid\}\} \quad (6)$$

Figure 4. shows the resource allocation process in state space using ACO with DRL.

As we will see in Figure 5, It can be observed that the action space is significantly reduced (to the same size as the total number of servers by using continuous DRL based decision system.

Reward. In our process, the reward specifies the range of the idle resources and allocated resources. There are two assumptions involved in the reward function.

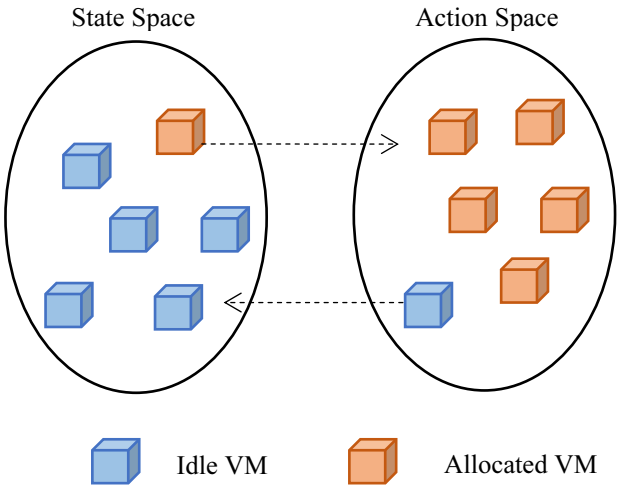

Figure 5. Transition of VM using DRL

- If the resources of the virtual machine such as CPU, memory, I/O are partially or fully allocated to the task for performing their process, that VM is placed in the action space. Here, the virtual machines are allocated to the task based on their requirement.
- If the resources of a virtual machine are unused by the tasks, that virtual machine is placed in the state space. In our process, the virtual machines in the state space are mainly preferred for allocating the tasks.

Transition. In resource allocation, the VM transfer from the state space to action space and vice versa is referred to as transition. If a virtual machine in the state space turns out to be busy, it is transferred to the action space. When a virtual machine in the action space becomes idle, it is transferred to the state space.

$$VM_{Busy} \rightarrow AS \quad (7)$$

$$VM_{Idle} \rightarrow SS \quad (8)$$

Here, AS refers to action space and SS refers to state space. Figure 5 illustrates the transition of the VM in cloud.

4.3.2 The Use of ACO for Resource Allocation

The Deep Reinforcement Learning algorithm is used to mitigate the resource complexity. As scheduled tasks are placed into state space, resources are allocated to the required tasks based on ACO with DRL. Ant Colony Optimization algorithm is a random probability model that searches for an optimal path. It is inspired by how ants discover an appropriate path to locate food. Ants will release a certain quantity of pheromones, and gradually the pheromone concentration on the shortest path becomes higher and attracts more ants, leading to the discovery of the best route. In our system, we have proposed an improved ACO algorithm which is based on scout characteristics, and it solves the problem of resource complexity. The

main idea of the improved ACO algorithm is partitioning the artificial ants into two groups: scout ants and common ants. The common ant performs according to the searching process of the basic ACO algorithm. The scout ants have some difference from common ants; they calculate each resource capacity of the current optimal solution and search around the optimal solution according to the resource capacity. In basic ACO, tasks randomly choose the resources at each step of the solution construction and select the following resource to be visited according to the probabilistic decision rule. However, the scout ants' distinct characteristic, whereby they differ from common ants, is that they search around the optimal solution. At the same time, they will directly visit the following resource in optimal solution. When ant k states in resource i, the probability of moving to the neighbor j of resource i is given by,

$$P_{ij}^{k}(t) = \{\frac{\tau_{ij}^{a}(t)}{\Sigma_{j\in N_t^k}\tau_{ij}^{a}(t)}, j \in N_t^k\} \quad \textbf{(9)}$$

Where N_k^t are the adjacent nodes of node i. **α** is a factor used to amplify the impact of pheromone concentration. If **α** is too large, it will overpower the impact of the pheromone, leading the algorithm to converge onto a sub-optimal path. When the ant is in the $(t+1)^{th}$ iteration, pheromone concentration of each path becomes,

$$\tau_{ij}(t+1) = (1-\beta)\times\tau_{ij}(t) + \sum_{k=1}^{n_k}\Delta\tau_{ij}^{k} \quad \textbf{(10)}$$

Where n_k is the number of ants and β is the pheromone evaporation parameter. Based on ACO, the resources are allocated to tasks in three steps:

- Understanding user demands
- Initialization of parameters
- Allocation of resources

The allocation by ACO can be illustrated by Figure 6 To perform the user's request, the task is initialized with CPU, Memory and I/O intensive requirements. Tasks must be allocated to resources which satisfy their needs; i.e. T (CPU, memory, I/O) → R. Tasks search for the optimal resource for execution. Each virtual machine has a resource predictor, which is used to search for resources for the tasks.

The initialized tasks search for the best solution S_{best} and update the pheromone. If the tasks reach the optimal solution, they again search the resources around the optimal solution based on resource capacity. Finally, the tasks are allocated to suitable resources based on improved ACO. Otherwise, it again computes the S_{best}. Here, we consider Alpha=1 which is the pheromone constant, and Beta = 1, which is the heuristic constant. Finally, the allocated tasks shift into the action space.

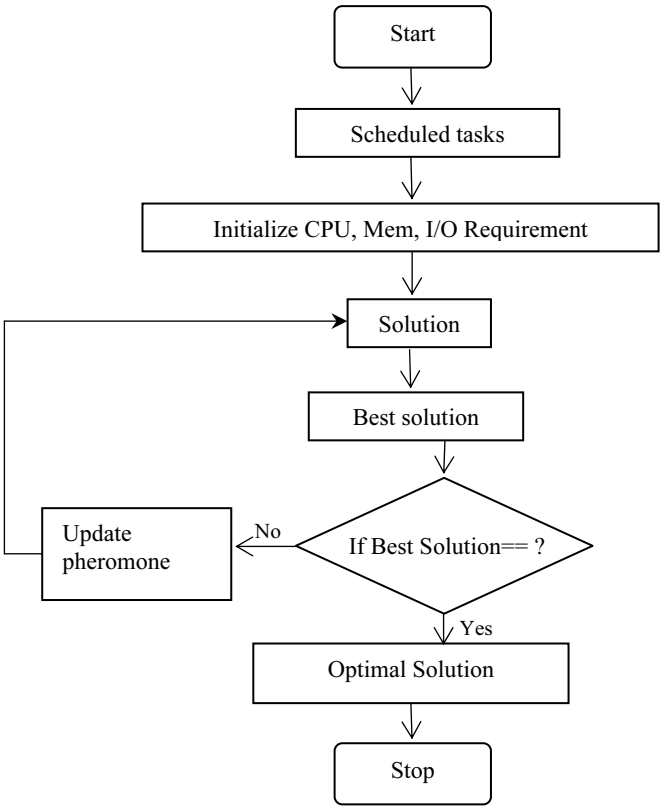

Figure 6. Flow chart for resource allocation

Pseudocode for ACO

Input: Scheduled Tasks T = (T_1…T_n)
Output: $T_1 \rightarrow R_1$

1. Begin
2. Compute State space (Equation 4)
2. Get T={T_1,T_2,…T_n}
3. Initialize T→ (CPU, Mem, I/O)
4. Compute S_{best}
5. Update pheromone
6. If
   {T = T_{max}}
   Get optimal solution
7. Search the solution around optimal solution
   Else
   Go to step 4
8. End if
9. End

To fulfil our objectives such as high-power consumption and less execution time, we propose a DRL framework which solves the overall resource allocation and power management problems in cloud computing systems. As earlier discussed, the proposed DRL framework comprises two spaces: state space and action space. State space is for VM resource allocation to the servers and action space for VM management of all servers. Besides reducing the search time, we propose an ACO that enhances scalability and also reduces the dimensions of state and action space. To reduce the action space, we adopt a continuous-time and DRL technique, in which each decision is made

with the arrival time of a new VM request.

The above Pseudocodes give the details about resource allocation using ACO with DRL. The overall algorithm for resource allocation is summarized as follows,

Algorithm for Resource Allocation

Input: scheduled tasks
Output: Resource allocated tasks
1. Begin
2. By using DRL, resources are divided into state space and action space
3. Define state space from (Equation 4)
4. Compute reward function
5. Scheduled task entered into state space .// Ant Colony Optimization for resource allocation
7. Initialize the requirement of tasks (CPU, Mem, I/O)
8. Compute the best solution S_{best}
9. Update pheromone
10. Get optimal resource if maximum iteration reached
11. Put allocated virtual machines into Action space
12. Update the Action space
13. End

4.4 VM Migration

When a virtual machine is overloaded, it is migrated to another physical machine in the host. In the proposed method, each VM and PM is designed with predictors.

Resource predictor. A separate resource predictor module is dedicated to each VM in the physical machine. The resource predictor module for VM x consists of three primary jobs which monitor and collect the VM's CPU, Memory and I/O usage, to predict the VM's future CPU, Memory and I/O demands. The resource predictor also provides information about available resources in the VM for future resource allocation.

Overload predictor. Each physical machine in the host has an overload predictor. Let us assume that it is located on PM y. It fetches the predicted CPU, Memory and I/O demands for all VMs which are hosted on that PM, and calculates the PM's predicted aggregate CPU, Memory and I/O demands, U^y_{CPU}, U^y_{men} and $U^y_{I/O}$ respectively, as follows,

$$U^y_{CPU} = \Sigma_{x \in V;\, \theta(x)=y} P^x_{men} \quad \textbf{(11)}$$

$$U^y_{men} = \Sigma_{x \in V;\, \theta(x)=y} P^x_{CPU} \quad \textbf{(12)}$$

$$U^y_{I/O} = \Sigma_{x \in V;\, \theta(x)=y} P^x_{I/O} \quad \textbf{(13)}$$

Where $\theta(x) = y$ denotes that VM x is hosted on PM y. The module compares the predicted value to the PM's supported value. The supported value of PMs for CPU, Memory and I/O demands is calculated as follows,

$$S^y_{CPU} = \sum_{x=1}^{n} S^x_{CPU} \quad \textbf{(14)}$$

$$S^y_{men} = \sum_{x=1}^{n} S^x_{men} \quad \textbf{(15)}$$

$$S^y_{I/O} = \sum_{x=1}^{n} S^x_{I/O} \quad \textbf{(16)}$$

If $U^y_{CPU} > S^y_{CPU}$ or $U^y_{men} > S^y_{men}$ or, then the overload predictor notifies the virtual machine allocation manager that PM y is expected to have an overload in the coming periods. The overload predictor also forwards the predicted CPU, Memory and I/O demands, P^x_{CPU}, P^x_{men} and $P^x_{I/O}$ for VM x hosted on PM j. This prediction is used to decide which VM(s) to keep, which VM(s) to migrate as well as where the migrated VM(s) should be moved.

VM allocation manager. The physical machines' overload predictors are managed by the VM allocation manager. Thus, the VM allocation manager knows the information about the available resources on the PMs. The VM allocation Manager considers a PM as overloaded if the prediction resource level of the PM is greater than its actual support level. Then the overloaded VMs are migrated to the underutilized physical machine.

5 Experimental Evaluation

In this section, a set of experiments are provided to evaluate our proposed work. We will compare our proposed work with existing methods. We have proposed ACO with DRL for resource allocation to improve the idle resource utilization and reduce the completion time. This section is divided into three sub-sections: simulation setup, performance metrics, and comparative analysis.

5.1 Simulation Setup

To implement the proposed work, we used a Pentium IV processor, 2GB RAM and 60 GB hard disk. The performance of the proposed work was evaluated using a Cloudsim 3 toolkit.

Simulation requirements are listed out in Table 1.

Table 1. Simulation set up requirement

Requirements	Specification	Value / Ranges
Data centers	No. of data centers	10
	No. of hosts	2-6
Cloudlet (Task)	Task length	1000-20000
	No. of tasks	50-500
Virtual Machines (VMs)	Memory (RAM)	128-2048
	MIPS	500-2000
	Bandwidth	500-1000
	No. of VM	100-200

Table 1 shows the required parameters for performing our proposed work. The requirements of the system setup can be changed based on the users' need. In our process, we have considered the above-mentioned requirements for the system implementation. The quantity of the requirements does not degrade the performance of the system.

5.2 Performance Metrics

This section defines each performance metric. The performance of the proposed work is evaluated with the following metrics.

Execution time. Execution time is defined as the time taken for scheduling the tasks and allocating resources to the task. Execution time is a significant parameter; it should be low to achieve better performance of the model.

$$T_E = \sum_{t=0}^{n} T_s + T_R \tag{17}$$

Resource utilization. Resource utilization is defined as the total usage of the resources such as CPU, Memory and I/O to execute the task. Here, utilization of each virtual machine is calculated by using their corresponding CPU intensive, Memory intensive and I/O intensive utilization.

$$VM_U = \sum_{i=1}^{n} U_{VM_i}^{CPU} + U_{VM_i}^{Mem} + U_{VM_i}^{I/O} \tag{18}$$

Power consumption. Power consumption is defined as power consumed by the usage of CPU, Memory and I/O. To achieve efficient performance, we must reduce the power consumption of the system.

$$P_{P_M} = \sum_{vm=1}^{n} P_{vm}^{CPU} + P_{vm}^{men} + P_{vm}^{I/O} \tag{19}$$

5.3 Comparative Analysis

This section briefly describes the performance of the proposed approach with reference to existing approaches. To check the performance of our proposed approach, we first compared our proposed task scheduling and resource allocation (hybrid ACO and DRL) with Cloud Workflow Scheduling Algorithm (CWSA) [2], Dynamic Power Saving Resource Allocation (DPRA) [5] , and Heuristic approach [17]. Table 2 provide the keys focused when we are evaluating these existing frameworks in a cloud computing environment. Generally, execution time, power consumption and resource utilization can be improved for better performance.

Table 2. Demerits of previous algorithms

Previous Algorithms	Limitations
Cloud Workflow Scheduling Algorithm (CWSA) [2]	It takes more time to allocate the resources Lengthier process
Dynamic Power Saving Resource Allocation (DPRA) [5]	Power utilization rate is high Execution time is high
Heuristic Approach [17]	High complexity due to the integration of two optimization algorithms Smaller utilization tasks have to wait for longer time

Execution time. Execution time is one of the key parameters to evaluate performance of the system. The execution time not only depends on scheduling and resource allocation method, but also on the number of incoming tasks at the time. In our work, tasks were scheduled and processed within the deadline. Execution time of our proposed work is compared with CWSA [2] and the Heuristic approach [17] processes in Table 3.

Table 3. Execution time comparison of the proposed approach with previous approaches

Average execution time			
Num of task	CWSA	Heuristic Approach	Proposed method
100	100	90	80
200	150	100	90
300	160	110	100
400	170	130	120
500	190	150	135
600	200	170	150
700	220	190	170
800	230	220	190
900	240	240	220
1000	250	250	230

The plot below represents the execution time against the number of tasks, because the execution time of a task increases linearly when the number of tasks is increased.

Figure 7 shows the changes in tasks' execution time corresponding to the number of tasks. In the proposed method, tasks are scheduled based on task length, tardiness and makespan, which reduces the waiting time of short deadline tasks that leads to reduction in execution time, which is significantly better than in previous work. The task execution time varies based on the deadline of the tasks. The tasks in the long queue consume more time for execution when compared to tasks in the short queue. When the execution time is reduced, it linearly reduces the time taken for completing the task. In this way, our system shows lesser completion time compared to other previous work.

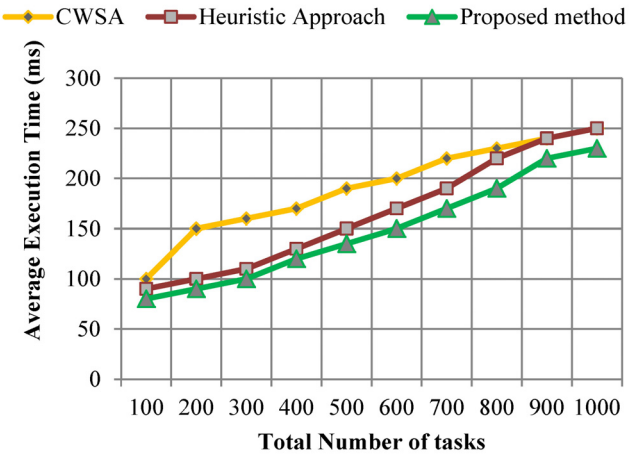

Figure 7. Comparison on average execution time

Resource utilization. Each task has different resource utilization levels based on their requirements to complete the task. The use of resources increases when the tasks take up and make use of the idle resources. We compare the proposed model to CWSA [2] and ACOPSO [18]. The resource Utilization of the proposed approach, the CWSA [2] and ACOPSO [18] approaches are provided in Table 4.

Table 4. Resource utilization comparison with the proposed approach with previous approaches

Average Arrival time	Resource Utilization		
	ACOPSO	CWSA	Proposed method
1	85%	85%	85%
2	80%	80%	80%
3	78%	78%	78%
4	74%	74%	74%
5	71%	71%	71%
6	67%	67%	67%
7	64%	64%	64%
8	60%	60%	60%

The graphical representation of the resource utilization can be represented below.

Figure 8 Depicts the percentage of resource utilization of CWSA and the proposed work. The graph clearly shows the rate of utilized resources by considering the time taken from the arrival of task to the next task. The rate of resource utilization changes during the time between the arrival of each task.

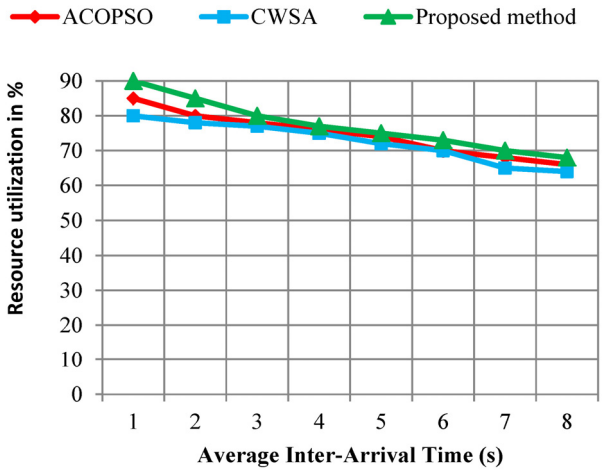

Figure 8. Comparison on resource utilization

As displayed above, the proposed method exhibits better resource utilization than CWSA and ACOPSO methods, since CWRA uses three scheduling policies for task scheduling, namely Easy Backing, FCFS and MCT. These policies take high waiting time to allocate resources for scheduled tasks. In the case of the proposed method, resource use is up to 10% better than the existing methods, since we used a larger number of ants (agents) to search for the best solution. This is due to the DRL process, which divides the resources into state space and action space to improve idle resource utilization. Based on the results from ACO and DRL, we have obtained this result and agents are used according to the available resources. In our system, the tasks initially utilize the resources in the state space where the resources are idle, which improves the utilization of idle resources.

Power consumption. Energy consumption is one of the significant parameters in the evaluation, which increases the performance of a physical machine.

Figure 9 shows the power consumption of physical machines investigated when the number of the virtual machines varies from 0-500. Each physical machine in the host consumes power by the usage of CPU, Memory and I/O. We compared the power consumption of the proposed method to DPRA. The results show that the proposed method has lower power consumption, more specifically a 10% reduction in consumption of power compared with the existing methods.

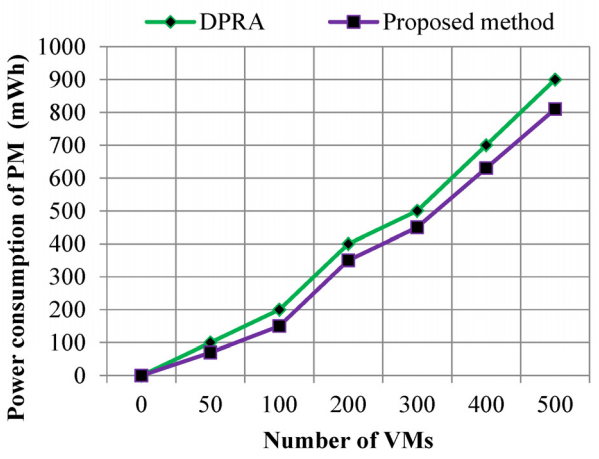

Figure 9. Comparison on power utilization

6 Conclusion

Cloud computing has come to be widely recognized as an essential computing model, with the involvement of vast numbers of users. Due to the large number of user requests, scheduling and resource allocation are performed mainly for task organization and execution in an efficient manner. Our proposed work is designed in such a way as to improve resource utilization and reduce execution time. In this paper, we initiated task queuing based on the task deadline, and tasks are scheduled using Binary In - order Traversal Tree based on a weighted value for each task. We then perform the resource allocation by hybrid Ant Colony Optimization and Deep Reinforcement Learning algorithm. The resource utilization is improved by splitting them into state space and action space by DRL, and the resources are allocated based on ACO. The overall allocation process is managed by VM allocation manager which also performs VM migration to reduce power consumption. Finally, performance evaluation indicates that our proposed work is efficient for task scheduling and resource allocation, with improved resource utilization and reduced execution time.

In the future work, we plan further investigation on the following area:

- VM migration can be further enhanced by using VM consolidation process.
- Introducing real time data to implement the proposed task scheduling and resource allocation process.
- Implement the resource provisioning in future for efficient resource allocation

References

[1] R. Karthikeyan, P. Chitra, Novel Power Reduction Framework for Enhancing Cloud Computing by Integrated GSNN Scheduling Method, *Cluster Computing*, Vol. 21, No. 1, pp. 755-766, March, 2018.

[2] B. P. Rimal, M. Maier, Workflow Scheduling in Multi-Tenant Cloud Computing Environments, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 1, pp. 290-304, January, 2017.

[3] L. Shi, Z. Zhang, T. Robertazzi, Energy-aware Scheduling of Embarrassingly Parallel Jobs and Resource Allocation in Cloud, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 6, pp. 1607-1620, June, 2017.

[4] Y. Chen, L. Wang, X. Chen, R. Ranjan, A. Y. Zomaya, Y. Zhou, S. Hu, Stochastic Workload Scheduling for Uncoordinated Datacenter Clouds with Multiple QoS Constraints, *IEEE Transactions on Cloud Computing*, June, 2016. DOI: 10.1109/TCC.2016.2586048.

[5] L.-D. Chou, H.-F. Chen, F.-H. Tseng, H.-C. Chao, Y.-J. Chang, DPRA: Dynamic Power-Saving Resource Allocation for Cloud Data Center Using Particle Swarm Optimization, *IEEE Systems Journal*, Vol. 12, No. 2, pp. 1554-1565, June, 2018.

[6] N. Lim, S. Majumdar, P. Ashwood-Smith, MRCP-RM: A Technique for Resource Allocation and Scheduling of MapReduce Jobs with Deadlines, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 5, pp. 1375-1389, May, 2017.

[7] M. Khabbaz, K. Shaban, C. Assi, Delay-Aware Flow Scheduling in Low Latency Enterprise Datacenter Networks: Modelling and Performance Analysis, *IEEE Transactions on Communications*, Vol. 65, No. 5, pp. 2078-2090, May, 2017.

[8] M. Dabbagh, B. Hamdaoui, M. Guizaniy, A. Rayesz, An Energy-Efficient VM Prediction and Migration Framework for Overcommitted Clouds, *IEEE Transactions on Cloud Computing*, Vol. 6, No. 4, pp. 955-966, October-December, 2018.

[9] A. Ahmad, A. Paul, M. Khan, S. Jabbar, M. M. U. Rathore, N. Chilamkurti, N. Min-Allah, Energy Efficient Hierarchical Resource Management for Mobile Cloud Computing, *IEEE Transactions on Sustainable Computing* , Vol. 2, No. 2, pp. 100-112, April-June, 2017.

[10] P. Zhang, M. Zhou, Dynamic Cloud Task Scheduling Based on a Two-Stage Strategy, *IEEE Transactions on Automation Science and Engineering*, Vol. 15, No. 2, pp. 772-783, April, 2018.

[11] T. Truong-Huu, M. Gurusamy, S. T. Girisankar, Dynamic Flow Scheduling with Uncertain Flow Duration in Optical Data Centers, *IEEE access*, Vol. 5, pp. 11200-11214, June, 2017.

[12] L. Zuo, L. Shu, S. Dong, Y. Chen, L. Yan, A Multi-objective Hybrid Cloud Resource Scheduling Method Based on Deadline and Cost Constraints, *IEEE access*, Vol. 5, pp. 22067-22080, December, 2016.

[13] T. Zhao, S. Zhou, X. Guo, Z. Niu, Tasks Scheduling and Resource Allocation in Heterogeneous Cloud for Delay-bounded Mobile Edge Computing, *2017 IEEE International Conference on Communications (ICC)*, Paris, France, 2017, pp. 1-7.

[14] K. B. Bey, F. Benhammadi, F. Sebbak, M. Mataoui, New Tasks Scheduling Strategy for Resources Allocation in Cloud

Computing Environment, *2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, Istanbul, Turkey, 2015, pp. 1-5.

[15] S. Liu, K. Ren, K. Deng, J. Song, A Dynamic Resource Allocation and Task Scheduling Strategy with Uncertain Task Runtime on IaaS clouds, *2016 Sixth International Conference on Information Science and Technology (ICIST)*, Dalian, China, 2016, pp. 174-180.

[16] X. Xu, L. Cao, X. Wang, Resource Pre-allocation Algorithms for Low-energy Task Scheduling of Cloud computing, *Journal of Systems Engineering and Electronics*, Vol. 27, No. 2, pp. 457- 469, July, 2016.

[17] M. B. Gawali, S. K. Shinde, Task Scheduling and Resource Allocation in Cloud Computing Using a Heuristic Approach, *Springer Journal of Cloud Computing: Advances, Systems and Applications*, Vol. 7, No. 4, pp. 1-16, February, 2018.

[18] D. Kumar, B. Kavitha, M. Padmavathy, B. Harshini, E. PReethi, P. Varalakshmi, Optimized Particle Swarm Optimization Based Deadline Constrained Task Scheduling in Hybrid Cloud, *ICTACT Journal on Soft Computing*, Vol. 6, No. 2, pp. 1117-1122, January, 2016.

[19] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, H. Tenhunen, Energy Aware VM Consolidation in Cloud Data Centers Using Utilization Prediction Model, *IEEE Transactions on Cloud Computing*, Vol. 7, No. 2, pp. 381-388, October, 2016.

Biographies

Ulysse Rugwiro is a Ph.D. candidate in Computer Science and Application at East China University of Science and Technology. He received his MSc degree from ECUST, China. He is working as a researcher in the field of Green Cloud Computing.

Chunhua Gu born in 1970, he is Professor and Ph.D. supervisor in the School of Information Science and Engineering, East China University of Science and Technology. Senior member of China Computer Federation. His main research interests include cloud computing and internet of things.

Weichao Ding born in 1989, he is Ph.D. candidate in the School of Information Science and Engineering, East China University of Science and Technology. His main research interests include cloud computing and high-performance computing.