International Journal of Computer Networks and Applications (IJCNA)
DOI: 10.22247/ijcna/2023/221887 Volume 10, Issue 3, May – June (2023)

RESEARCH ARTICLE

Amended Hybrid Scheduling for Cloud Computing with Real-Time Reliability Forecasting

Ramya Boopathi
Department of Computer Science, Vellalar College for Women, Erode, Tamil Nadu, India.
ramyaboopathiphd1988@gmail.com

E.S. Samundeeswari
Department of Computer Science, Vellalar College for Women, Erode, Tamil Nadu, India.
samundeeswari@vcw.ac.in

Received: 04 April 2023 / Revised: 11 May 2023 / Accepted: 16 May 2023 / Published: 30 June 2023

Abstract – Cloud computing has emerged as the feasible paradigm to satisfy the computing requirements of high-performance applications by an ideal distribution of tasks to resources. But, it is problematic when attaining multiple scheduling objectives such as throughput, makespan, and resource use. To resolve this problem, many Task Scheduling Algorithms (TSAs) are recently developed using single or multi-objective metaheuristic strategies. Amongst, the TS based on a Multi-objective Grey Wolf Optimizer (TSMGWO) handles multiple objectives to discover ideal tasks and assign resources to the tasks. However, it only maximizes the resource use and throughput when reducing the makespan, whereas it is also crucial to optimize other parameters like the utilization of the memory, and bandwidth. Hence, this article proposes a hybrid TSA depending on the linear matching method and backfilling, which uses the memory and bandwidth requirements for effective TS. Initially, a Long Short-Term Memory (LSTM) network is adopted as a meta-learner to predict the task runtime reliability. Then, the tasks are divided into predictable and unpredictable queues. The tasks with higher expected runtime are scheduled by a plan-based scheduling approach based on the Tuna Swarm Optimization (TSO). The remaining tasks are backfilled by the VIKOR technique. To reduce resource use, a particular fraction of CPU cores is kept for backfilling, which is modified dynamically depending on the Resource Use Ratio (RUR) of predictable tasks among freshly submitted tasks. Finally, a general simulation reveals that the proposed algorithm outperforms the earlier metaheuristic, plan-based, and backfilling TSAs.

Index Terms – Cloud Computing, Task Scheduling, TSMGWO, Meta-Learning, LSTM, Plan-Based Scheduling, Tuna Swarm Optimization, Backfilling, VIKOR.

1. INTRODUCTION

In recent years, cloud computing has become increasingly significant in several industries. As a principle of on-demand cloud services over web-based systems, it is operated to meet user demands for resource access or to allow members to acquire cloud facilities when needed [1]. Based on the application and design strategies required by the members, it may also handle a wide range of services [2]. It is a mix of parallel and distributed computing that makes use of shared resources like software and hardware, which are paid for as they are used [3].

One of the operational paradigms used to operate servers, data centers, and VMs is called Infrastructure-as-a-Service (IaaS). IaaS is a type of cloud facility, which offers a host or CPU in the cloud to process and store data. It gives users access to servers locally because the cloud resources are hosted on VMs [4]. It relies on the demands of the member, named Service Level Agreements (SLAs), and Quality-of-Service (QoS). The expense is based on the contract between the member and the Cloud Service Providers (CSPs) [5]. Additionally, it facilitates CSPs to deliver data centers with high-efficiency computing resources, which helps users to access cloud applications.

The VM utilizes the cloud resources such as the memory, or CPU. The VM resources are needed by the demanded resource for functioning [6]. So, the cloud system has an imbalanced resource allocation, and a few VMs cannot get the resources they need since several VMs have preemptive and non-preemptive relations to resources [7]. The VM must be fast to react while the task is delivered to be executed in the cloud to minimize waiting time. But, tasks must be allocated among each VM simultaneously to maintain equilibrium and guarantee effective utilization of the available resources. So, TS is essential to allocate tasks across available resources [8]. If several tasks are allocated to a single VM or several VMs, the allocated tasks can execute simultaneously to fulfill the requirements. So, it is important to verify the allocation to guarantee that not each task is placed onto one VM, since this might make other VMs unavailable or imbalanced in the system. To prevent this, a schedule should consider additional factors like makespan, cost, and resource use [9]. It ensures maximum resource use by delivering satisfactory performance under various task constraints such as execution deadlines. Several researchers have modeled TS problems as bin packing problems or metaheuristic problems, which are solved using a First Come First Serve (FCFS), Max-Min, and so on [10-11]. On the other hand, such methods have the drawbacks of deceiving in local minima because of the multimodal behavior of TS. Currently, metaheuristic algorithms have attained significant attention to find the close-ideal result to the TS challenge within a minimum duration [12]. Numerous metaheuristic algorithms have solved TS problems using single or multiple objectives to discover close-ideal results and offer trade-offs to CSPs. Amongst, the TSMGWO [13] can reduce the cost and enhance the resource utilization of VMs by finding near-optimal solutions when handling conflicting objectives. Nonetheless, this algorithm only tries to optimize system throughput, resource usage, and balancing workloads across VMs, whereas additional parameters like the use of the memory, and bandwidth are needed to improve the TS performance.

Therefore, in this paper, a hybrid TSA is proposed by using two main scheduling strategies called linear matching method and backfilling. The key goal is to consider the utilization of the memory and bandwidth with less computation time for effective TS. The major contributions of this paper include:

1. First, the LSTM is applied as a meta-learner to predict the network-produced task runtime reliability during task submission, which helps to categorize the tasks into predictable and unpredictable tasks.
2. Then, the plan-based scheduling approach based on Tuna Swarm optimization is adopted for tasks with a maximum projected runtime and backfills the residual tasks on top of the deliberate tasks.
3. To achieve backfilling, a VIKOR (VIseKriterijumska Optimizacija I Kompromisno Resenje) technique is adopted. When resource distribution is applied to reduce resource use, a particular fraction of CPU cores is set aside for backfilling of fewer predictable tasks. This fraction is modified dynamically depending on the RUR of predictable tasks among freshly submitted tasks. Thus, this algorithm can discover the optimal available resources during TS for improving makespan, system throughput, resource use, and minimizing execution time.

The remaining article is prepared as follows: Section II discusses different TSAs that emerged in previous years. Section III explains the presented TSA and Section IV portrays its efficiency. Section V abridges the whole work.

2. LITERATURE SURVEY

A Hybrid Moth Search Algorithm and Differential Evolution (HMSADE) [14] has been presented to plan the jobs in cloud systems. But, it considers only the execution time as the objective function, whereas it needs other parameters like memory usage, bandwidth, etc., to increase the efficiency of TS with less time complexity. A Laxity-based Priority (LBP) [15] has been suggested to allocate tasks using a suitable significance that enhances the sensitivity of job delay. Also, Ant Colony Optimization (ACO) has been employed to lessen aggregate energy utilization. Nonetheless, it didn't schedule the independent tasks.

Chen et al. [16] developed a multi-objective Improved Whale Optimization Algorithm (IWOA) to distribute the cloud jobs. Nevertheless, the tradeoff between exploration and exploitation was less efficient due to the poor convergence and it did not handle independent tasks. Jia et al. [17] presented an Improved Whale Optimization (IWC) scheme to enhance TS efficiency by finding the optimal whale individuals using the inertial weight mechanism. Also, add and delete functions were used after all iterations to monitor and choose the best individuals. But, the impact on the memory load value was not clear, which needs to enhance memory usage.

Wang & Zuo [18] designed a cloud workflow scheduling method called HPSO by hybridizing Particle Swarm Optimization (PSO) and idle timeslot-aware rules for decreasing the execution cost of a workflow request under a deadline limit. A novel particle encoding was used to define the VM category needed by all tasks and the scheduling sequence of tasks. To decode a particle into a scheduling solution, an idle timeslot-aware decoding process was applied. Also, a repair scheme was adopted to handle the task's invalid priorities. But, the network bandwidth among VM instances was assumed to be equal.

Dubey & Sharma [19] developed a new hybrid TSA called Chemical Reaction-PSO (CR-PSO) to allocate multiple independent tasks on the available VMs. The CR optimization and PSO were hybridized by uniting the features for the optimal schedule sequence where tasks can be processed according to the demand and deadline concurrently. But, it did not consider the dependent tasks and it needs additional parameters like bandwidth for improving efficiency.

Ajmal et al. [20] developed Hybrid Ant Genetic Algorithm (HAGA) for TS effectively. A new strategy was used to add and evaporate VM pheromones to identify the load on the VM. The GA mutation process was redesigned to identify loaded VMs and schedule tasks to balance loads among VMs. But, it needs other parameters like resource usage, energy consumption, etc., to increase further efficiency.

Calzarossa et al. [21] analyzed a multi-objective restricted optimization issue to recognize the best scheduling strategies for systematic tasks to be employed in unreliable cloud scenarios. The main aim was to reduce the estimated task execution period and monetary expense under probabilistic restraints on deadline and budget. This issue was resolved by the combined Monte Carlo method and Genetic Algorithm (MCGA), the cloud clients were permitted to select the strategy of the Pareto optimum group ensuring their demands and interests. But, the optimum results were not achieved under severe deadlines and costs, because the variability raises.

Kumar et al. [22] modeled the TS problem as a non-linear restricted optimization dilemma and solved using the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) to reduce makespan and power usage. But, it did not consider the cost and deadline restraints for TS. Oudaa et al. [23] presented a novel framework depending on the Quantile Regression Deep Q-Network (QR-DQN) to create a suitable strategy and the best long-term solutions to assign resources and schedule tasks to corresponding VMs. But, if the number of tasks was increased, then more tasks were discarded because of running out of resources or time.

Sharma & Garg [24] presented the Load-Balancing Ant Colony Optimization (LBACO), which combines the Genetic Algorithm (GA) and ACO for QoS-based TS in cloud computing. In contrast, advanced optimization schemes must be deployed to further boost competence while considering multiple objectives. Mahmoud et al. [25] suggested a multi-objective TS using a Decision Tree (TS-DT) algorithm to schedule and execute the applications' tasks. But, it did not consider memory, bandwidth, etc., to determine the objective function for effective TS.

Kruekaew & Kimpan [26] designed an independent Multi-objective TS Optimization depending on the Artificial Bee Colony (ABC) scheme with a Q-learning (MOABCQ). The objective of this algorithm was to enhance TS and resource usage, throughput, and achieve load-balancing among VMs. On the other hand, it cannot ensure that this algorithm was optimal so that the network performance was not enhanced in each test corpus.

Table 1 summarizes earlier researchers in terms of algorithms used, merits, and demerits compared to the proposed algorithm.

Table 1 Comparison of Earlier TSA in Cloud Computing

Ref. No.	Algorithms used	Merits	Demerits
[13]	TSMGWO	Reduce the cost and improve resource utilization.	It did not consider other crucial factors like memory and bandwidth for TS.
[14]	HMSADE	It can schedule the tasks to the VM when consuming a minimum makespan.	It considered only one objective function such as the execution time.
[15]	LBP-ACO	It can ensure reasonable scheduling length and reduce the failure rate of TS with varied deadlines.	It didn't schedule the independent tasks.
[16]	Multi-objective IWOA	It can enhance efficiency based on system load and resource usage.	The tradeoff between exploration and exploitation was less efficient due to the poor convergence and it did not handle independent tasks.
[17]	IWC	It achieved better TS period, and scheduling cost.	The impact on the memory load value was not clear, which needs to enhance memory usage.
[18]	HPSO	It achieved better efficiency in terms of execution cost and the success rate in reaching the deadline.	The network bandwidth among VM instances was assumed to be equal.
[19]	CR-PSO	It was more economical. It achieved less execution period, and cost.	It did not consider the dependent tasks and it needs additional parameters like bandwidth for improving efficiency.
[20]	HAGA	It can discover a feasible schedule to reduce the execution time of tasks.	The optimum results were not achieved under severe deadlines and costs, because the variability raises.

[21]	MCGA	It was based on the deadline and budget constraints.	No feasible solutions to the constrained optimization problem were not achieved for the larger variability factor of VM performance.
[22]	TOPSIS	It was robust in terms of makespan and reliability metrics.	It did not consider the cost and deadline restraints for TS.
[23]	QR-DQN	It was cost-effective by guaranteeing QoS efficiency.	If the number of tasks was increased, then more tasks were discarded because of running out of resources or time.
[24]	LBACO	It can decrease the execution period and cost.	It did not perform well for multi-objective optimization.
[25]	TS-DT	It can decrease makespan and enhance resource usage.	It increases energy utilization. It needs to consider memory, bandwidth, energy usage, etc., to determine the objective function for effective TS.
[26]	MOABCQ	It can reduce makespan, cost, degree of imbalance, and increase throughput.	It cannot ensure that this algorithm was optimal so that the network performance was not enhanced in each test corpus.
Proposed	LSTM-TSO-VIKOR	It can schedule both predictable and unpredictable tasks efficiently by considering multiple objectives, including makespan, resource usage, execution time, throughput, memory, and bandwidth.	Still, it did not consider energy utilization as the objective function.

3. PROPOSED METHODOLOGY

In this section, the proposed hybrid TSA is explained briefly. An overview of the TSA in cloud computing is portrayed in Figure 1. Many independent task requests are forwarded through a graphical user interface to the cloud broker in a task list. The CSP verifies whether the appropriate number of cloud resources are available to complete the requirements for task execution or not. If the required number of cloud resources is available, then the requested tasks are scheduled to the resources based on the scheduling algorithm. Otherwise, the CSP denied the requested tasks to the cloud user and waits for the availability of the resources.

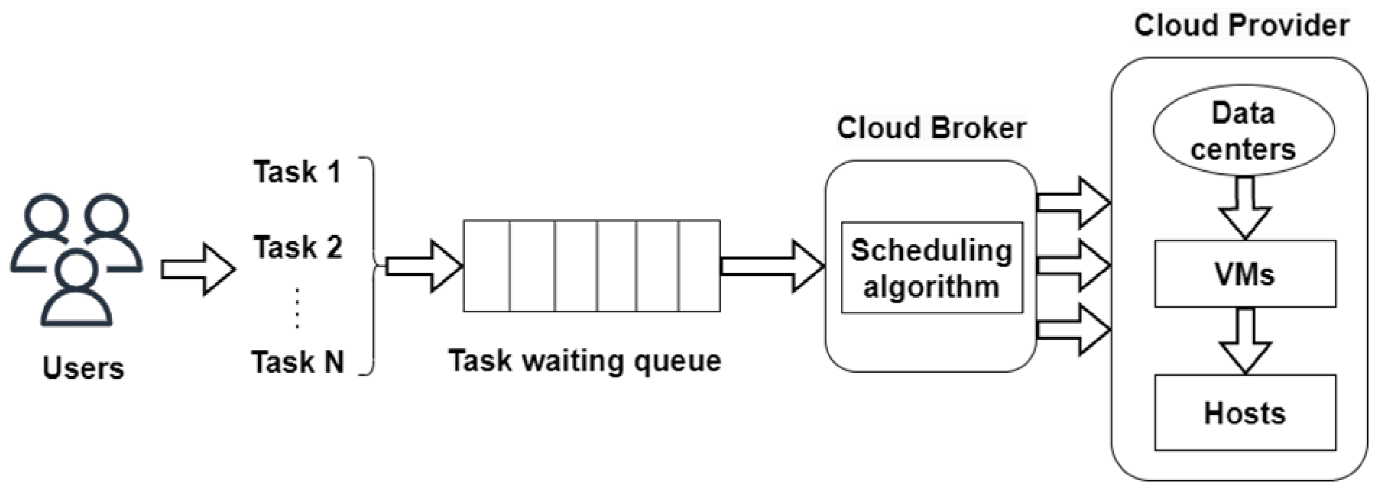

Figure 1 Overview of Task Scheduling in Cloud Computing

3.1. Problem Definition

Initially, the task j is assumed with a specified submission period and resource necessity. During the submission, these are submitted by the consumer, where n separate tasks, and j contains different characteristics: submission period r_j, resource requirement d_j, actual running period p_j, and requested running period $\hat{p}_j$. The problem considered is to implement a group of simultaneous tasks with inflexible resource necessities on the cloud with m resource units. The tasks are submitted over an interval. The resource necessity d_j of j is equal to the consumer-demanded resource acknowledged at the period of submission. The real runtime is named a posteriori while the task finishes. The challenge is how to schedule tasks to realize enhanced efficiency, with no proper runtime values during the submission period.

To combat this problem, a hybrid scheduling system is proposed, which utilizes a linear matching method and backfilling for tasks with random runtime. During the task submission, the LSTM network is employed to predict the reliability of task runtime. After that, tasks are categorized into predictable and unpredictable queues using the learned LSTM network. Moreover, the tasks with higher expected runtime are scheduled by the TSO as plan-based scheduling, and the tasks in the unpredictable queue are scheduled on the residual accessible resources by the VIKOR-based backfilling. The predictable and unpredictable tasks are described as follows:

- Predictable tasks: These are tasks with great expectation reliability and are represented by the predicted runtime reliability of 60% or higher.
- Unpredictable tasks: These are tasks with little expectation reliability and are represented by the predicted runtime reliability of less than 60%.

3.2. Formation of Objective (Fitness) Functions

The proposed model considers 6 major objectives such as resource utilization, makespan, memory usage, bandwidth use, execution time, and throughput for optimization.

3.2.1. Makespan

It is the total period needed from submitting a task to the end of the task by the consumer. It is calculated in equation (1):

$$Makespan = max\left(MS(VM_x)\right), 1 \le x \le n \quad (1)$$

In Eq. (1), n is the number of VMs, and $MS(VM_x)$ is calculated in equation (2).

$$MS(VM_x) = \sum CT_{xy} \times Assigned(x,y) \quad (2)$$

In Eq. (2), CT is the computation period of task j_x on VM_y, $Assigned(x,y) = 1$, when j_x is scheduled on VM_y; or else, $Assigned(x,y) = 0$. The computation period of VM_y to execute j_x is calculated in equation (3):

$$CT_{xy} = \sum \left(j_x.MI \middle/ VM_y.MIPS \right) \quad (3)$$

In Eq. (3), $j_x.MI$ denotes a million instructions of j_x, $VM_y.MIPS$ provides a Million Instructions Per Second of VM_y. So, the 1st objective is to reduce makespan as given in equation (4):

$$f_1 = \min(Makespan) \quad (4)$$

3.2.2. Resource Utilization

•

The mean RUR is calculated as the fraction of the mean makespan to the highest makespan of the cloud network as in equation (5) and equation (6):

$$Mean\_RUR = {Mean\ makespan} \big/ {Cloud\ makespan} \quad (5)$$

$$Mean\ makespan = \sum {MS(VM_x)} \big/ {n}, 1 \le x \le n \quad (6)$$

The cloud makespan is equal to the maximum period for the completion of each system workload. The lower value of $Mean\_RUR$ represents the insignificant use of computing resources and the higher value defines the complete use of computing resources in the cloud platform. So, the 2nd objective is to increase the mean RUR as in equation (7):

$$f_2 = \max(Mean\_RUR) \quad (7)$$

3.2.3. Execution Time

4. It is also known as the Degree of Imbalance (DoI), which measures the inequity of workload dissemination among VMs as per their abilities. It is calculated by equation (8).

$$IB\_Deg = \frac{Max\_CTime_j - Min\_CTime_j}{Mean\_CTime_j} \quad (8)$$

In Eq. (8), Max_CTime_i is the highest implementation period of j on each VM, Min_CTime_i is the lowest implementation period of j on each VM, $Mean\_CTime_j$ is the mean completion period of j on each VM. The lesser IB_Deg define that the cloud workload is balanced properly, while greater values define that load balancing is ineffective. So, the 3rd objective is to reduce the DoI (execution time) as in equation (9):

$$f_3 = \min(IB\_Deg) \quad (9)$$

3.2.4. Throughput

5. It determines the amount of tasks executed per interval. It is calculated based on the makespan by equation (10).

$$Throughput = {Number\ of\ tasks} \big/ {Makespan} \quad (10)$$

A greater throughput is needed for an efficient TSA. So, the fourth objective is to increase throughput as in equation (11):

$$f_4 = \max(Throughput) \quad (11)$$

3.2.5. Memory Utilization

It defines the maximum memory requirement of each VM for task execution and is calculated by equation (12).

$$Memory = AM_x + \frac{RM_j}{TM_x} \quad (12)$$

In Eq. (12), AM_x indicates the memory utilization before implementing j at the i^{th} VM, RM_j indicates the memory holding the request of j, and TM_x denotes the overall memory accessible at i^{th} VM. So, the fifth objective is to reduce memory utilization as in equation (13):

$$f_5 = \min(Memory) \quad (13)$$

3.2.6. Bandwidth Usage

It defines the maximum bandwidth requirement of each VM for task execution and is calculated by equation (14).

$$Bandwidth = AB_x + \frac{RB_j}{TB_x} \quad (14)$$

In Eq. (14), AB_x indicates the bandwidth utilization before implementing j at the i^{th} VM, RB_j defines the bandwidth holding the request of j, and TB_x denotes the overall bandwidth accessible at i^{th} VM. So, the sixth objective is to minimize the bandwidth utilization as in equation (15):

$$f_6 = \min(Bandwidth) \quad (15)$$

3.3. Hybrid Task Scheduling System

The proposed hybrid TSA comprises 4 major modules as shown in Figure 2 including the meta-learner module, hybridization parameter fine-tuning module, chief scheduler, and repository. If the task is submitted to the network, its runtime, and reliability are predicted in the meta-learner module. Then, the predicted values are utilized by the chief scheduler to calculate the positioning period of the fresh task in the cloud platform. Once the task execution is completed, data regarding real runtimes is stored in the repository to be utilized by the meta-learner module and hybridization parameter fine-tuning module. The repository has the essential information with the LSTM network hyperparameters. Because the task finishes execution on the cloud, the runtime value is added to the repository.

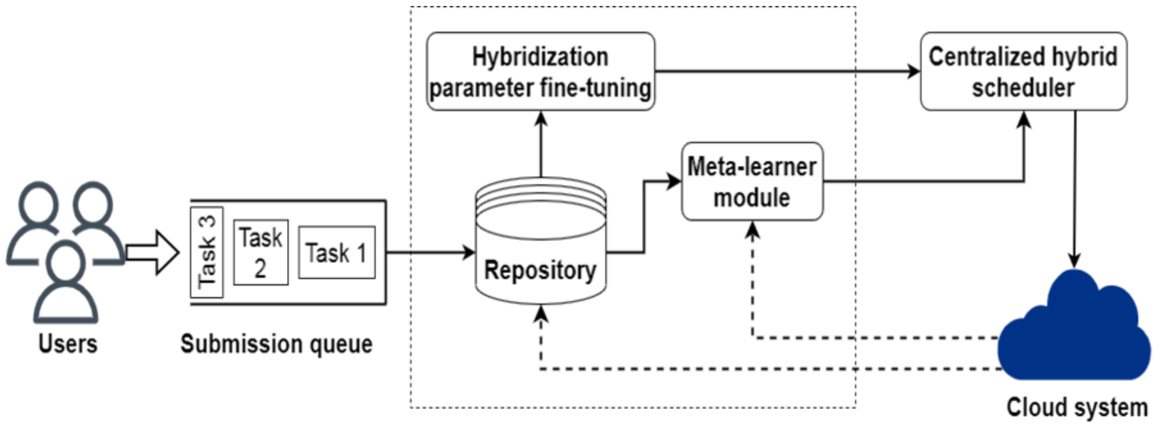

Figure 2 Schematic Representation of the Presented Hybrid TSA in Cloud Computing

3.3.1. Central Scheduler

Utilizing the input from the meta-learner module, tasks are split into 2 queues: predictable and unpredictable tasks. Tasks that are represented as predictable by the meta-learner module are input to a fresh queue, known as expected. The hybrid scheduler has 2 schedulers, namely plan-based and backfilling schedulers. Initially, the chief scheduler applies plan-based scheduling to determine the early period of all predictable tasks in the waiting queue. To perform this, a particular proportion of resources utilizing a certain ratio of CPU cores calculated by the hybridization parameter module is deliberated. Then, once the positioning time of the predictable tasks is calculated, VIKOR with backfilling scheduler calculates the early period of unpredictable tasks on the residual accessible resources. The entire plan for each task is utilized to position tasks on the cloud. This process is continued during task submission, task execution, or task termination.

3.3.2. Hybridization Parameter Fine-tuning Module

The fraction of resources utilized by the plan-based scheduling strategy to plan predictable tasks α is known hybridization parameter and can be modified by the hybridization parameter fine-tuning module. The value of α is modified according to the fraction of the aggregate resource utilization of predictable tasks to the cumulative resource request of each task. The modification of α is defined by equation (16).

$$\alpha_{t+1} = \alpha_t * \frac{\sum_{i\in predictable} d_i}{\sum_{j\in each\ task} d_j} \quad (16)$$

In Eq. (16), d_j denotes the resource requirement for the task j. The early value of α represented as α_0 is selected via a grid search.

3.3.3. Meta-Learner Module

The meta-learner module is used to predict runtime and the reliability of the expected runtime. The runtime forecast is done by the LSTM network model. The runtime reliability prediction defines how expected the runtime of a specified task is depending on the attributes regarding the task during submission. To decide the forecast reliability of tasks, the LSTM network model is applied. The framework hyperparameters are stored in the repository, and the framework is reconstructed each day. The main role of the meta-learner module is to decide whether the tasks belong to predictable or unpredictable categories.

To analyze estimation accuracy for each task, a proper metric called a point-wise measure is used, which determines the correctness of runtime estimation for all tasks by equating the estimated task runtime $(\hat{p}_j)$ and actual task runtime (p_j) as equation (17):

$$Acc_j = \begin{cases} 1, & \hat{p}_j = p_j \\ \frac{\hat{p}_j}{p_j}, & \hat{p}_j < p_j \\ \frac{p_j}{\hat{p}_j}, & \hat{p}_j > p_j \end{cases} \quad (17)$$

3.3.4. LSTM Network Model as Meta-Learning for Estimating Runtime Prediction Accuracy

The LSTM is employed to predict the forecast correctness task runtimes (expectation reliability). The LSTM model predicts the reliability of the freshly submitted tasks using the attributes listed in Table 2 and the correctness of implemented tasks as the target values. As depicted in Figure 3, attribute vectors of implemented tasks $f_i = \{z_{1i}, \dots, z_{ki}\}$, and their respective forecast correctness Acc_i are utilized to train the LSTM network. This network translates the attributes and the expected runtime to correctness. Then, the learned framework is utilized to measure the runtime forecast correctness of the freshly submitted tasks.

In Figure 4, the expectation reliability provides correctness for task i, $\widehat{Acc}_i$, by attribute vectors $f_i = \{z_{1i}, \dots, z_{ki}, \hat{p}_i\}$.

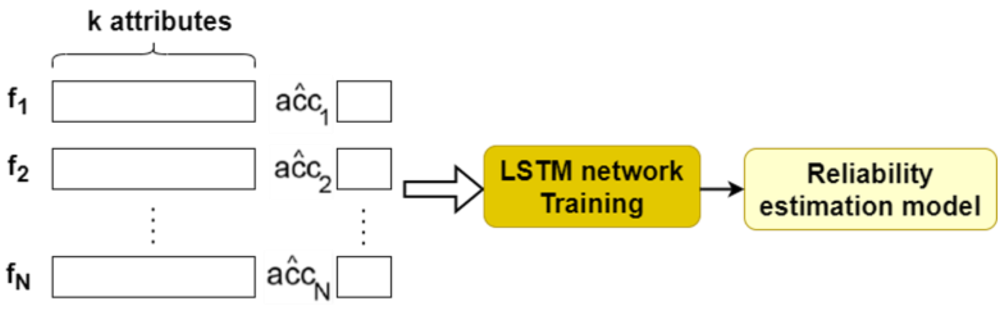

Figure 3 Training of the Prediction Reliability Estimation Model

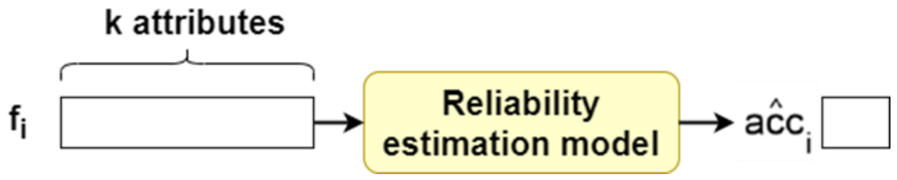

Figure 4 Predicting Accuracy for Newly Submitted Tasks using Trained Reliability Estimation Model

Table 2 Attributes Considered for Task Runtime Prediction Reliability Estimation

Attribute category	Attribute name
Consumer-demanded runtime	$\hat{p}_j$ (reqTime)
Real runtime and CPU for the earlier implemented tasks from the identical consumer	last, beflast, beflast2, lastcpu
Highest runtime and CPU for the earlier implemented tasks from the	maxrt, maxcpu

identical consumer	
Seasonality attributes	tod1, tod2, dow1, dow2
Mean and standard variance of runtime and CPU of the tasks from the identical consumer	meanrt, stdrt, meancpu, stdcpu
The quantity of implemented tasks from the identical consumer	prevuser

The LSTM network includes 3 gate control strategies as shown in Figure 5 such as forget, input, and output gate.

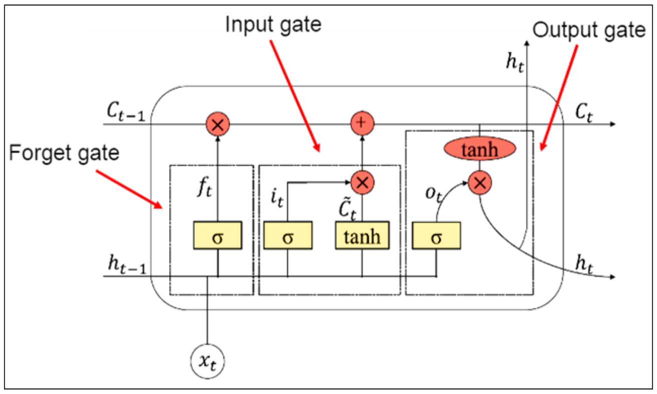

Figure 5 Architecture of LSTM Cell

The presence of the forget gate is to compute the level of disremembering the data course preceded by the ongoing LSTM unit. The determination is defined in equation (18):

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (18)$$

The role of the input gate is to estimate how much present data is included in the data course. The determination is defined in equations (19) and (20):

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (19)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (20)$$

Once the data traverse via the input and forget gates, the LSTM fine-tunes its units to determine the outcome of the ongoing LSTM unit and passes it to the consecutive LSTM unit. The determination is defined in equation (21):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (21)$$

The output gate merges the present input and LSTM unit to compute the result of the present LSTM unit. The computation is defined in equations (22) and (23):

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (22)$$

$$h_t = o_t * tanh(C_t) \quad (23)$$

In Eqns. (18)-(23), W_f, W_i, W_o and W_C are the weight coefficient vector associated with the hidden layers, input and output gates, and the neuron condition vector, correspondingly. Additionally, b_f, b_i, b_o and b_C denote their corresponding offset values (i.e., bias values, which prevent overfitting problems). σ is the sigmoid activation function, x_t is the input to the input gate, h_t is the output of a hidden state, h_{t-1} is the output of a previous hidden state, C_t is a cell state at time t, f_t is a forget gate, $\tilde{C}_t$ is a new memory, i.e. cell update, and o_t is the output of the output gate. The process to obtain an LSTM predictor $\widetilde{meta}$ for input F is presented in Algorithm 1. The algorithm contains a learning collection of k-dimensional input attributes of dimension N represented as $F = \{f_1, \dots, f_N\}$ and their respective target correctness values $ACC = \{Acc_1, \dots, Acc_N\}$. As the initial timestep, the LSTM $meta_0(F)$ is fitted to F and ACC. Utilizing the trained LSTM, a solution area of attribute space for fresh data is provided. The expected target value for fresh data is anticipated by averaging the runtime of the tasks in a similar sub-area. This process is continued for a varying number of epochs.

```
Initialize $meta_0 = \operatorname{argmin} \sum_{i=1}^{N} L(Acc_i, \gamma)$
$for(m = 1, \dots, M)$
    $for(i = 1, \dots, N)$
        Calculate $r_{im} = -\left[\frac{\partial L Acc_i meta(f_i)}{\partial meta(f(x_i))}\right]_{f=f_{m-1}}$;
    $end\ for$
    Fit the LSTM network to the target $r_{im}$;
    $for(j = 1, \dots, m)$
        Calculate $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_j \in R_{jm}} \left(Acc_i meta_{m-1}(f_{m-1}(x_i) + \gamma)\right)$;
    $end\ for$
    Update $meta_m(x) = meta_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right)$;
$end\ for$
Result: $\widetilde{meta}(x) = meta_M(x)$
```

Algorithm 1 LSTM Network as the Meta-Learner

3.4. Plan-based Scheduling for Predictable Tasks using Tuna Swarm Optimization

A species of marine predatory fish known as tuna or tunnini. The sizes of the different tuna species vary seriously. Tuna are the main oceanic hunters that consume a range of surface and midwater species. The "fishtail form" is a unique and efficient swimming technique used by continuous swimmers and tunas, wherein the body is inflexible and the long, thin tail swings swiftly. The lone tuna moves incredibly swiftly, but it cannot keep up with the small fish's quick response [27]. The tuna will consequently participate in "group migratory" predation. They exploit their cunning to seek and catch their prey. They performed two hunting policies: (i) spiral hunting: while eating, tuna swim in a spiral pattern to entice fish into shallow water where they may be effortlessly attacked, and (ii) parabolic hunting: all tunas swim in a line after the one before it, developing a parabolic curve around its victim.

3.4.1. Initialization

The process of scheduling predictable tasks on the available resources is started by TSO by randomly generating primary populations in the search region by equation (24).

$$\mathrm{S_i^{ini} = rand \cdot (u_l - l_l) + l_l, i = 1, \dots, NP} \quad (24)$$

In Eq. (24), $\mathrm{S_i^{ini}}$ is to the $\mathrm{i^{th}}$ tuna, $\mathrm{u_l}$ and $\mathrm{l_l}$ indicate the maximum and minimum limits of the search area, N is the number of tuna populations, Dim is the population size, and rand is uniformly distributed arbitrary vector ranging between 0 and N. Every individual $\mathrm{S_i^{ini}}$ in the tuna swarm stands for a nominee result for TSO. Every tuna comprises a group of Dim-dimensional numbers.

In all iterations, each tuna in the search region calculates its fitness function as equation (25):

$$\mathrm{f = [f_1, f_2, f_3, f_4, f_5, f_6]} \quad (25)$$

In Eq. (25), $\mathrm{f_1}$ is the makespan, $\mathrm{f_2}$ is the resource utilization, $\mathrm{f_3}$ is the runtime (DoI), $\mathrm{f_4}$ is the throughput, $\mathrm{f_5}$ is the memory utilization, and $\mathrm{f_6}$ is the bandwidth utilization. The exploitation and exploration tradeoff is achieved by combining the genetic operators such as crossover and mutation operators in each iteration for new population generation. Also, the location of each tuna is updated based on the two different foraging strategies.

3.4.1.1. Spiral Hunting

The majority of tuna cannot decide which direction to swim in while pursuing their meal, however a tiny percentage of fish may direct the swarm. The adjacent tuna will pursue this tiny group of fish when they begin pursuing their prey. The complete tuna swarm can eventually create a spiral pattern to capture its target.

If the tuna swarm uses a spiral hunting approach, each can communicate to determine which individuals or nearby individuals in the swarm are ideal to follow. Even the most talented individual occasionally fails to successfully guide the swarm in prey acquisition. The tuna will then decide to follow a random swarm member. The spiral foraging strategy is defined by equations (26), (27), (28), (29), and (30).

$$S_i^{t+1} = \begin{cases} \alpha_1 \cdot (S_{rand}^t + \beta \cdot |S_{rand}^t - S_i^t|) + \alpha_2 \cdot S_i^t, & i = 1 \\ \alpha_1 \cdot (S_{rand}^t + \beta \cdot |S_{rand}^t - S_i^t|) + \alpha_2 \cdot S_{i-1}^t, & i = 2,3,\dots,N \end{cases} \; if\ rand < \frac{t}{t_{max}}$$
$$S_i^{t+1} = \begin{cases} \alpha_1 \cdot (S_{best}^t + \beta \cdot |S_{best}^t - S_i^t|) + \alpha_2 \cdot S_i^t, & i = 1 \\ \alpha_1 \cdot (S_{best}^t + \beta \cdot |S_{best}^t - S_i^t|) + \alpha_2 \cdot S_{i-1}^t, & i = 2,3,\dots,N \end{cases} \; if\ rand \geq \frac{t}{t_{max}} \quad (26)$$

$$\alpha_1 = \alpha + (1 - \alpha) \cdot \frac{t}{t_{max}} \quad (27)$$

$$\alpha_2 = (1 - \alpha) - (1 - \alpha) \cdot \frac{t}{t_{max}} \quad (28)$$

$$\beta = e^{bl} \cdot \cos(2\pi b) \quad (29)$$

$$l = e^{3\cos\left(\left((t_{max} + 1/t) - 1\right)\pi\right)} \quad (30)$$

In Eqns. (26) – (30), S_i^{t+1} is the i^{th} tuna in the $t+1$ iteration, which is generated by the crossover and mutation operators, S_{best}^t is the present best individual, S_{rand}^t denotes the reference point arbitrarily chosen in the tuna swarm, α_1 indicates the weight value to handle the tuna whirling to the ideal individual or arbitrarily chosen nearby individuals, α_2 indicates the weight value to handle the tuna whirling to the individual in front of it, β denotes the distance factor to handle the gap between the tuna and the best tuna or an arbitrarily chosen reference individual, α indicates a constant to determine the level of tuna following, t is the present iteration, t_{max} indicates the highest iterations and b defines the arbitrary value from 0 to 1.

3.4.1.2. Parabolic Hunting

Tunas collaborate to feed by producing both a spiral pattern and a parabolic pattern. Tuna develops a parabolic shape using prey as a point of reference. Tuna also search their surroundings for nourishment. These two techniques are used simultaneously, with a 50% selection probability for each. This strategy is described in equations (31) and (32):

$$\mathrm{S_i^{t+1}} = \begin{cases} \mathrm{S_{best}^t + rand \cdot (S_{best}^t - S_i^t) + \gamma \cdot p^2 \cdot (S_{best}^t - S_i^t),} & \mathrm{if\ rand < 0.5} \\ \mathrm{\gamma \cdot p^2 \cdot S_i^t,} & \mathrm{if\ rand \geq 0.5} \end{cases} \quad (31)$$

Where $p = \left(1 - \frac{t}{t_{max}}\right)^{\left(\frac{t}{t_{max}}\right)}$ (32)

In Eq. (31), γ is the random value of 1 or -1. During the iteration, every tuna can randomly select to execute either the spiral or the parabolic hunting policy. As well, tuna can produce fresh individuals using crossover and mutation operators in the search region according to the probability z. The crossover and mutation operators are applied between two tuna swarms (the best one and the worst one based on their fitness value) to generate new offspring (new population).

As a result, TSO can select various strategies based on those genetic operators while producing fresh individual locations. In the implementation of the TSO, each tuna in the inhabitants is regularly modified until the termination criteria are met. At last, the TSO provides the best individual in the inhabitants and its best solution (scheduling ideal predictable tasks to the available resources). Algorithm 2 presents the TSO algorithm to find optimal tasks scheduled to the available resources.

```
Input: Tuna population size $NP$, maximum iteration $itr_{max}$, the number of predictable jobs $PT_j$, and the number of VMs $VM_x$, where $j \in \{1, \dots, J\}$ and $x \in \{1, \dots, X\}$
Output: Set of optimal predictable task schedules
Begin
Generate the initial population of tunas $S_i^{ini}$ $(i = 1, \dots, NP)$ randomly;
Set free parameters $a$ and $z$;
$while(t < t_{max})$
    Compute the fitness value $f$ of all tunas as Eq. (25);
    Modify the location and value of the best tuna $S_{best}^t$;
    $for(all\ tunas)$
        Modify $\alpha_1, \alpha_2, p$ by Eqns. (27), (28), and (32);
        $if(rand < z)$
            Modify $S_i^{t+1}$ using Eq. (24);
        $else\ if(rand \geq z)$
            $if(rand < 0.5)$
                Modify the location $S_i^{t+1}$ using Eq. (26);
            $else\ if(rand \geq 0.5)$
                Modify the location $S_i^{t+1}$ using Eq. (31);
            $end\ if$
        $end\ if$
    $end\ for$
$end\ while$
Find the best tuna $S_{best}$ in the search space, and the optimal fitness value $(f(S_{best}))$;
End
```

Algorithm 2 Plan-Based Scheduling for Predictable Tasks Using TSO

3.5. Backfilling for Unpredictable Tasks Using VIKOR Technique

The VIKOR technique is utilized with backfilling for tasks with unpredictable runtime. This technique solves the conflicts among the unpredictable tasks by formulating the criteria matrix A as equation (33):

$$A = \begin{array}{ccc} UT_1 & f_{11} & f_{12} \\ UT_2 & f_{21} & f_{22} \\ \vdots & \vdots & \vdots \\ UT_j & f_{j1} & f_{j2} \end{array} \quad (33)$$

Consider the criteria function $c = \{1, \dots, k\}$ and defines the benefit, then calculate f_c^* and f_c^-, where f_c^* determines the best task among the unpredictable tasks and f_c^- determines the worst tasks as equation (34).

$$f_c^* = \max_i f_{ic}, \quad f_c^- = \min_i f_{ic} \quad (34)$$

The maximum group utility (S_i) and individual regret of opponents (R_i) are determined by equations (35) and (36).

$$S_i = \sum_{c=1}^{n} w_c\, (f_c^* - f_{ic})/(f_c^* - f_c^-) \quad (35)$$

$$R_i = \max_c w_c\, (f_c^* - f_{ic})/(f_c^* - f_c^-) \quad (36)$$

In Eqns. (34) – (36), $i = \{1, \dots, n\}$. Here, 2 criteria are considered such as the execution time (E) and deadline (DT) for backfilling. The weight matrix for criteria is determined by equations (37) and (38).

$$W = \{w_E, w_{DT}\} \quad (37)$$

$$w_E = \sum_{c=1}^{k} \frac{\left({E_c}/{10}\right)}{A*j} \quad (38)$$

In Eq. (38), A is the maximum weight, which is equal to 6 because each execution period is in minutes. w_{DT} is calculated in equation (39).

$$w_{DT} = \sum_{c=1}^{k} \frac{\left({(DT_c - E_c)}/{E_c}\right)}{j} \quad (39)$$

After that, for $Q_i, i = \{1, \dots, n\}$ in equation (40):

$$Q_i = \frac{v(S_i - S^*)}{(S^- - S^*)} + (1 - v)\frac{(R_i - R^*)}{(R^- - R^*)} \quad (40)$$

In Eq. (40), $S^* = \min_i S_i$, $S^- = \max_i S_i$, $R^* = \min_i R_i$, $R^- = \max_i R_i$, and $v = 0.5$, i.e. weight of the majority of criteria. Moreover, rank S, R and Q in descending manner. Sort the best measure, i.e. Q (minimum) using the below criteria: Consider T' is the alternative task at the initial position and T'' is at the second position in Q as equations (41) and (42):

$$Q(T'') - Q(T') \geq DQ \quad (41)$$

$$DQ = \frac{1}{j-1} \quad (42)$$

When T'' is the best alternative in S and R, it is stable and should satisfy: $v > 0.5$ (majority rule) or $v \approx 0.5$ (consensus) or $v < 0.5$ (with veto). Therefore, the best alternative task T' among unpredictable tasks is chosen as a backfill task UT_b for scheduling with UT_j concurrently.

```
Input: Unpredictable task set $S = \{UT_1, \dots, UT_j\}$
Output: Backfill task $UT_b$
Initialize $Q' \leftarrow S$;
Choose $E$ and $DT$ as criteria for backfilling;
$for(i = 1, \dots, j)$
    Create a criteria matrix $A$ using Eqns. (33) & (34);
$end\ for$
$for(c = 1, \dots, k)$
    Calculate weight vector $W$ using Eqns. (38) & (39);
$end\ for$
$for(i = 1, \dots, n)$
    Find $S_i$ and $R_i$ using Eqns. (35) & (36);
$end\ for$
$for(i = 1, \dots, j)$
    Evaluate $Q_i$ using Eq. (40);
$end\ for$
Rank $S, R$ and $Q$ in descending order;
Apply Eq. (41) to determine the best alternative that is backfill task $UT_b$;
Return $UT_b$
```

Algorithm 3 VIKOR Backfilling

Thus, this hybrid TSA can schedule tasks efficiently with better resource utilization and other requirements by combining both plan-based and backfilling strategies.

4. SIMULATION RESULTS

The effectiveness of the LSTM-TSO-VIKOR is assessed by simulating it in CloudSim API 3.0.3 simulator. The simulation parameters are given in Table 3, which are set in the computer that has Intel ® Core ™ i5-4210 CPU @ 2.80GHz, 4GB RAM, 1TB HDD under Windows 10 64-bit operating system. A comparative analysis is also conducted between the proposed and existing algorithms including TSMGWO [13], HMSADE [14], HAGA [16], CR-PSO [19], and MOABCQ [26] in terms of various metrics. The considered metrics include makespan, mean resource use percentage, throughput, DoI, memory utilization, and bandwidth usage, which are defined in Section 3.2.

Table 3 Simulation Environment and Parameters

Type	Parameter	Value
Host	No. of hosts	100
	Host kinds	HP ProLiant ML110 G4
		HP ProLiant ML110 G5
HP ProLiant ML110 G4	No. of Processing Elements (PEs) per host	4
	Bandwidth	3Gbps
	Host memory	8GB
	MIPS of PE	2060
HP ProLiant ML110 G5	No. of PEs per host	4
	Bandwidth	3Gbps
	Host memory	8GB
	MIPS of PE	3560
VM	No. of VMs	450
	VM kinds	High-CPU Medium Instance
		Extra Large Instance
		Small Instance
		Micro Instance

High-CPU Medium Instance	MIPS of PE	2500
	No. of PEs per VM	5
	VM memory	1GB
	Bandwidth	118Mbps
Extra Large Instance	MIPS of PE	2000
	No.of PEs per VM	4
	VM memory	4GB
	Bandwidth	118Mbps
Small Instance	MIPS of PE	1000
	No. of PEs per VM	3
	VM memory	2GB
	Bandwidth	118Mbps
Micro Instance	MIPS of PE	500
	No. of PEs per VM	2
	VM memory	1.5GB
	Bandwidth	118Mbps
Cloudlets	No. of tasks	1000
	Task length (Million Instructions (MI))	2500*simulation limit
	No. of PEs per demand	2
LSTM	Learning rate	10^{-3}
	Layer size	2^5
	Dropout rate	0.5
	Number of epoch	30
TSO	a	0.7
	z	0.05

4.1. Makespan

Makespan is calculated by equation (1).

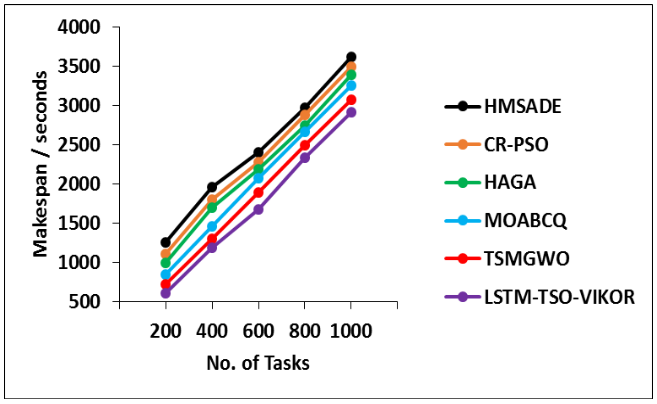

Figure 6 Makespan vs. No. of Tasks

Figure 6 explains the makespan achieved by the different hybrid TSAs. It is observed that the LSTM-TSO-VIKOR algorithm provides the highest reduction in makespan 51.87% over HMSADE, 45.2% over CR-PSO, 39.13% over HAGA, 28.99% over MOABCQ, and 16.9% over TSMGWO algorithms for 200 tasks in the cloud. This is because of an improved exploitation and exploration ability of the LSTM-TSOVIKOR algorithm compared to the other optimization algorithms for scheduling tasks to the VMs.

4.2. Mean Resource Utilization Ratio

Mean Resource Utilization Ratio is calculated by equation (4).

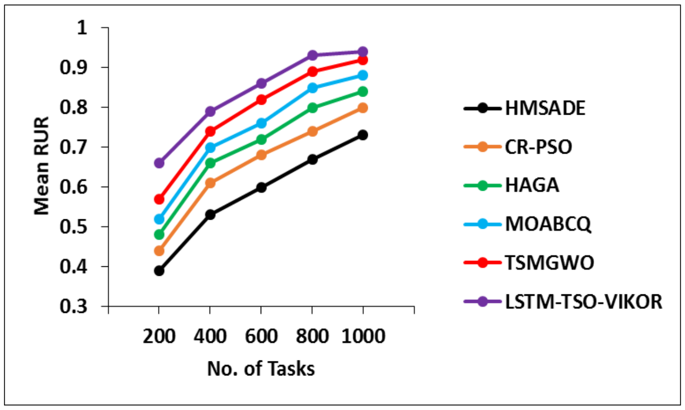

Figure 7 Mean RUR vs. No. of Tasks

In Figure 7, the mean RUR results for different hybrid TSAs are plotted. It is observed that the proposed LSTM-TSO-VIKOR-based TSA results in a high rise in resource use up to 69.23%, 50%, 37.5%, 26.92%, and 15.79% over HMSADE, CR-PSO, HAGA, MOABCQ, and TSMGWO algorithms, respectively, for 200 tasks in the cloud. This is because of finding optimal scheduling for both predictable and unpredictable tasks by enhancing the searchability of the LSTM-TSO-VIKOR algorithm.

4.3. Degree of Imbalance (DoI)

Degree of Imbalance (DoI) is calculated by equation (8).

Figure 8 compares the DoI results of different hybrid TSAs. It is noted that the LSTM-TSO-VIKOR-based TSA achieves better performance by providing less DoI compared to the other algorithms. The LSTM-TSO-VIKOR results in a high drop in the DoI up to 40.5% over HMSADE, 37.1% over CR-PSO, 31.3% over HAGA, 26.7% over MOABCQ, and 15.4% over TSMGWO algorithms for 200 tasks in the cloud. This is owing to the enhancing optimization procedure using TSO for scheduling predictable tasks and VIKOR technique for unpredictable tasks independently.

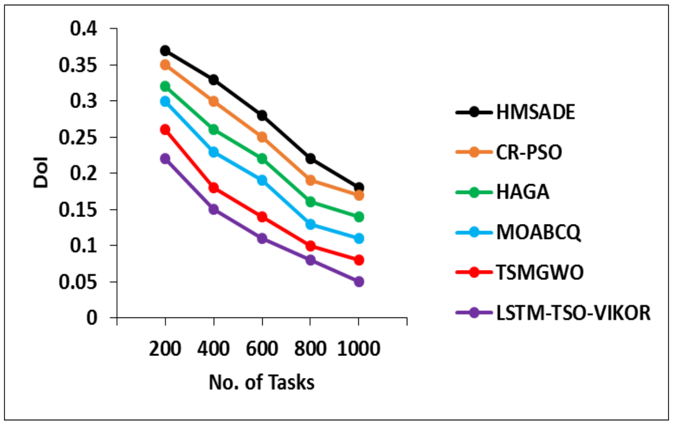

Figure 8 DoI vs. No. of Tasks

4.4. Throughput

Throughput is calculated by equation (10).

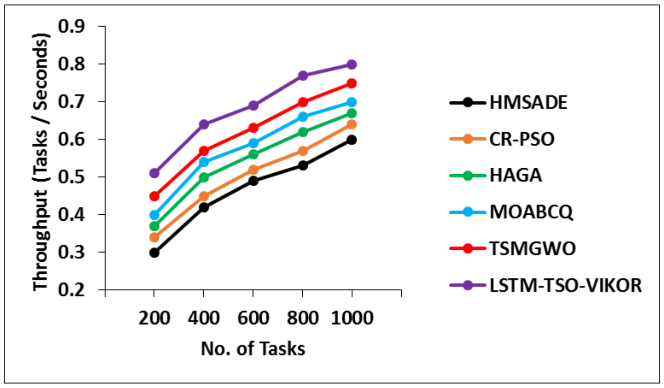

Figure 9 Throughput vs. No. of Tasks

In Figure 9, the throughput results achieved by various hybrid TSAs are drawn. It is realized that the LSTM-TSO-VIKOR creates an extreme rise in throughput up to 70% over HMSADE, 50% over CR-PSO, 37.8% over HAGA, 27.5% over MOABCQ, and 13.3% over TSMGWO algorithms for 200 tasks in the cloud. This is because of enhancing exploration and exploitation of optimizing multiple objectives for both predictable and unpredictable tasks scheduling efficiently.

4.5. Memory Utilization

Memory Utilization is calculated by equation (12).

Figure 10 exhibits the memory utilization results for different hybrid TSAs. It is noted that the LSTM-TSO-VIKOR creates a maximum reduction in memory utilization up to 40%, 33.33%, 28%, 21.74%, and 14.29% over HMSADE, CR-PSO, HAGA, MOABCQ, and TSMGWO algorithms, respectively for 200 tasks in the cloud. This is because the LSTM-TSO-VIKOR can enhance the optimization ability of both predictable and unpredictable task scheduling by using multiple objectives.

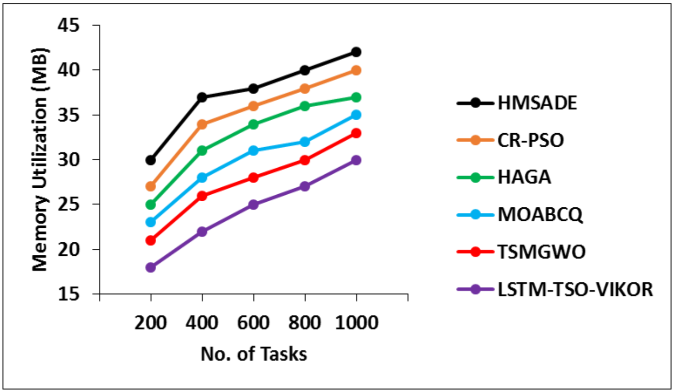

Figure 10 Memory Utilization vs. No. of Tasks

4.6. Bandwidth Utilization

Bandwidth Utilization is calculated by equation (14).

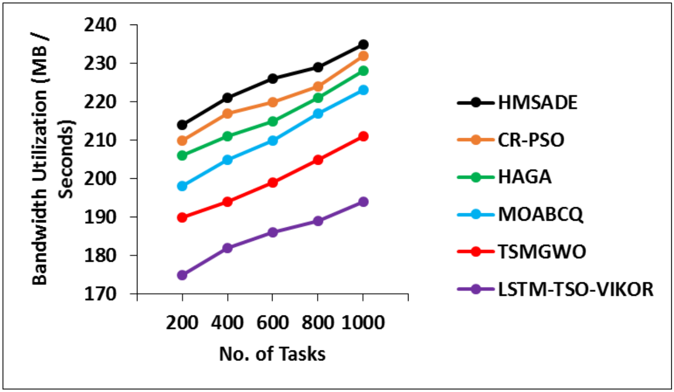

Figure 11 Bandwidth Utilization vs. No. of Tasks

In Figure 11, the bandwidth utilization results of various hybrid TSAs are plotted. It is shown that the LSTM-TSO-VIKOR-based TSA achieves a maximum reduction in bandwidth utilization up to 18.22%, 16.67%, 15.05%, 11.62%, and 7.89% compared to the HMSADE, CR-PSO, HAGA, MOABCQ, and TSMGWO algorithms for 200 tasks in the cloud. This is owing to the adoption of new optimization such as TSO and VIKOR for efficiently scheduling predictable and unpredictable tasks independently by predicting their runtime reliability using the LSTM model.

Thus, it is realized that a huge improvement in the network performance for a varying quantity of jobs utilizing the proposed LSTM-TSO-VIKOR algorithm. This is because of improving the tradeoff between the exploration and exploitation ability with fewer parameters during optimization, as well as, hybridizing both plan-based and backfilling TS strategies.

5. CONCLUSION

In this study, the hybrid TSA was developed depending on the combination of the linear matching method and backfilling for enhancing the performance of TS in cloud computing. First, the LSTM network model was applied as a meta-learner for estimating the prediction reliability of task runtimes. Based on this prediction, the tasks were split into 2 categories such as predictable and unpredictable. Further, the predictable tasks were scheduled by the TSO, whereas the remaining tasks were backfilled by the VIKOR technique. In the end, the experiments showed that the LSTM-TSO-VIKOR achieved a noteworthy improvement in the utilization of resources and other requirements for task execution compared to the existing TSAs in cloud computing.

REFERENCES

[1] Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., & Patterson, D. A. (2021). What serverless computing is and should become: the next phase of cloud computing. Communications of the ACM, 64(5), 76-84.

[2] Bello, S. A., Oyedele, L. O., Akinade, O. O., Bilal, M., Delgado, J. M. D., Akanbi, L. A., ... & Owolabi, H. A. (2021). Cloud computing in construction industry: use cases, benefits and challenges. Automation in Construction, 122, 1-18.

[3] Atieh, A. T. (2021). The next generation cloud technologies: a review on distributed cloud, fog and edge computing and their opportunities and challenges. ResearchBerg Review of Science and Technology, 1(1), 1-15.

[4] Shafiq, D. A., Jhanjhi, N. Z., Abdullah, A., & Alzain, M. A. (2021). A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE Access, 9, 41731-41744.

[5] Mishra, S. K., Sahoo, B., & Parida, P. P. (2020). Load balancing in cloud computing: a big picture. Journal of King Saud University-Computer and Information Sciences, 32(2), 149-158.

[6] Imran, M., Ibrahim, M., Din, M. S. U., Rehman, M. A. U., & Kim, B. S. (2022). Live virtual machine migration: a survey, research challenges, and future directions. Computers and Electrical Engineering, 103, 1-18.

[7] Houssein, E. H., Gad, A. G., Wazery, Y. M., & Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation, 62, 1-41.

[8] Bittencourt, L. F., Goldman, A., Madeira, E. R., da Fonseca, N. L., & Sakellariou, R. (2018). Scheduling in distributed systems: a cloud computing perspective. Computer Science Review, 30, 31-54.

[9] Asghari, A., Sohrabi, M. K., & Yaghmaee, F. (2020). Online scheduling of dependent tasks of cloud's workflows to enhance resource utilization and reduce the makespan using multiple reinforcement learning-based agents. Soft Computing, 24(21), 16177-16199.

[10] Jamil, B., Ijaz, H., Shojafar, M., Munir, K., & Buyya, R. (2022). Resource allocation and task scheduling in fog computing and internet of everything environments: a taxonomy, review, and future directions. ACM Computing Surveys, 10(1), 1-35.

[11] Murad, S. A., Muzahid, A. J. M., Azmi, Z. R. M., Hoque, M. I., & Kowsher, M. (2022). A review on job scheduling technique in cloud computing and priority rule based intelligent framework. Journal of King Saud University-Computer and Information Sciences, 34, 2309-2331.

[12] Al-Arasi, R., & Saif, A. (2020). Task scheduling in cloud computing based on metaheuristic techniques: a review paper. EAI Endorsed Transactions on Cloud Systems, 6(17), 1-19.

[13] Alsadie, D. (2021). TSMGWO: optimizing task schedule using multi-objectives grey Wolf optimizer for cloud data centers. IEEE Access, 9, 37707-37725.

[14] Abd Elaziz, M., Xiong, S., Jayasena, K. P. N., & Li, L. (2019). Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. Knowledge-Based Systems, 169, 39-52.

[15] Xu, J., Hao, Z., Zhang, R., & Sun, X. (2019). A method based on the combination of laxity and ant colony system for cloud-fog task scheduling. IEEE Access, 7, 116218-116226.

[16] Chen, X., Cheng, L., Liu, C., Liu, Q., Liu, J., Mao, Y., & Murphy, J. (2020). A WOA-based optimization approach for task scheduling in cloud computing systems. IEEE Systems Journal, 14(3), 3117-3128.

[17] Jia, L., Li, K., & Shi, X. (2021). Cloud computing task scheduling model based on improved whale optimization algorithm. Wireless Communications and Mobile Computing, 2021, 1-13.

[18] Wang, Y., & Zuo, X. (2021). An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules. IEEE/CAA Journal of Automatica Sinica, 8(5), 1079-1094.

[19] Dubey, K., & Sharma, S. C. (2021). A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing. Sustainable Computing: Informatics and Systems, 32, 1-20.

[20] Ajmal, M. S., Iqbal, Z., Khan, F. Z., Ahmad, M., Ahmad, I., & Gupta, B. B. (2021). Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers. Computers and Electrical Engineering, 95, 1-15.

[21] Calzarossa, M. C., Della Vedova, M. L., Massari, L., Nebbione, G., & Tessera, D. (2021). Multi-objective optimization of deadline and budget-aware workflow scheduling in uncertain clouds. IEEE Access, 9, 89891-89905.

[22] Kumar, M. S., Tomar, A., & Jana, P. K. (2021). Multi-objective workflow scheduling scheme: a multi-criteria decision making approach. Journal of Ambient Intelligence and Humanized Computing, 12(12), 10789-10808.

[23] Oudaa, T., Gharsellaoui, H., & Ahmed, S. B. (2021). An agent-based model for resource provisioning and task scheduling in cloud computing using DRL. Procedia Computer Science, 192, 3795-3804.

[24] Sharma, N., & Garg, P. (2022). Ant colony based optimization model for QoS-based task scheduling in cloud computing environment. Measurement: Sensors, 24, 1-9.

[25] Mahmoud, H., Thabet, M., Khafagy, M. H., & Omara, F. A. (2022). Multiobjective task scheduling in cloud environment using decision tree algorithm. IEEE Access, 10, 36140-36151.

[26] Kruekaew, B., & Kimpan, W. (2022). Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning. IEEE Access, 10, 17803-17818.

[27] Xie, L., Han, T., Zhou, H., Zhang, Z. R., Han, B., & Tang, A. (2021). Tuna swarm optimization: a novel swarm-based metaheuristic algorithm for global optimization. Computational Intelligence and Neuroscience, 2021, 1-22.

RESEARCH ARTICLE

Authors

Ms Ramya B is a research scholar in the department of computer science at Vellalar College for women, Erode, Tamilnadu. Her area of interest is cloud computing. Completed M.Sc (CT) in 2011 at Kongu Engineering College, Perundurai, Erode. Completed Mphil in Vasavi arts and Science College. Worked as Assistant professor in department of computer science, Kongu arts and science colleges, from 2016-2019.

Dr. E.S. Samundeeswari is working as Associate Professor in the Department of Computer Science at Vellalar College for Women, Erode, Tamilnadu. She has more than 33 years of academic experience. She completed her PG Degree in MCA in 1988 and achieved Doctoral Degree in 2008. Her area of interests include Computation Intelligence and Image Processing.

How to cite this article:

Ramya Boopathi, E.S. Samundeeswari, "Amended Hybrid Scheduling for Cloud Computing with Real-Time Reliability Forecasting", International Journal of Computer Networks and Applications (IJCNA), 10(3), PP: 310-324, 2023, DOI: 10.22247/ijcna/2023/221887.