International Journal of Basic and Applied Sciences, 14 (7) (2025) 576-588

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/njb1me19
Research paper

An Intelligent Runtime Dependability Forecast Based Hybrid Task Scheduling in Cloud

B. Suganya *, Dr. R. Padmapriya

School of Computer Studies UG, RVS College of Arts and Science,
Sulur, Coimbatore-641 402, Tamil Nadu, India
**Corresponding author E-mail: suganyaaphd20@gmail.com*

Received: October 7, 2025, Accepted: November 1, 2025, Published: November 29, 2025

Abstract

Cloud computing has emerged as a promising solution for meeting the computational objectives of high-performance systems by efficiently scheduling cloud workloads to available resources. However, it becomes problematic to achieve multiple objectives simultaneously, such as optimizing throughput, minimizing makespan, and improving resource utilization for effective Task Scheduling (TS). In response to this issue, the TS, based on many optimization algorithms, was recently developed. These algorithms optimally allocate tasks to appropriate resources to achieve multiple objectives simultaneously. Most of these algorithms enhance the resource usage and the network throughput while minimizing the makespan, but ignore memory and bandwidth consumption. Combining plan-based and backfilling strategies, this article proposes a hybrid TS algorithm that could fulfill numerous criteria for effective TS. Initially, the reliability of the job runtimes is pre-dicted using a meta-learner called a Gated Recurrent Unit (GRU) network. Then, the tasks are sorted into two categories: reliable and unreli-able. The Dove Optimization Algorithm (DOA) is used for plan-based scheduling of jobs with higher runtime estimates, while the Analytic Hierarchy Process (AHP) method is used to backfill jobs with lower estimated runtimes. In addition, a dynamic allocation of CPU cores is made for backfilling based on the resource limit ratio of reliable jobs to newly requested jobs. Lastly, the simulation results show that the proposed algorithm outperforms the existing TS algorithms. GRU-DOA-AHP reduces makespan by 20.38%, increases throughput by 30%, and decreases memory usage by up to 26.19% compared to traditional algorithms.

Keywords: *Scheduled Tasks; GRU Network; Meta-Learning, Cloud Computing; Optimizing for Doves and AHP Backfilling.*

1. Introduction

Cloud computing has gained significant importance in recent times for various companies, providing on-demand services through web-based systems to meet consumer demands. A crucial aspect of cloud computing is Infrastructure-as-a-Service (IaaS), which automates the management of hosts, data centers, and virtual machines. However, the scheduling of cloud services can become unbalanced due to both preemptive and non-preemptive scheduling conditions, potentially disregarding some Virtual Machines (VMs) without the necessary re-sources for optimal performance. Efficiency demands that jobs be distributed concurrently across multiple VMs to ensure proper resource allocation. Therefore, the effective distribution of jobs hinges on Task Scheduling (TS). TS problems have been described by many re-searchers as bin-packing issues or metaheuristic issues that were handled using First Come First Serve (FCFS), Max-Min, and other algo-rithms. However, the heterogeneous behavior of TS in the cloud poses several problems for these algorithms, including trapping in local optima. Numerous optimization methods have produced near-optimal results for TS problems using one or more criteria. Specifically, the Multi-Objective Grey Wolf Optimization (TSMGWO) [1] may handle conflicting criteria and identify practically ideal solutions, which improves VM resource efficiency and reduces costs. Though this method optimizes network throughput, resource utilization, and job distribution among VMs, additional factors like bandwidth and memory consumption are needed to make TS more effective.
Therefore, this article proposes a hybrid TS method that uses the linear matching algorithm and backfilling as its two primary scheduling mechanisms. The main goal of this method is to enhance memory and bandwidth utilization while requiring minimal computational re-sources for optimal TS. To help classify incoming tasks in the cloud into reliable and unreliable categories, the GRU is employed as a meta-learning technique during job requests to forecast the reliable runtimes of the tasks. The plan-based scheduling approach, which depends on the DOA, is then applied to jobs with longer expected runtimes, and the AHP method is used to backfill the remaining jobs. To reduce resource consumption, a specific proportion of CPU cores is reserved for backfilling less reliable jobs. This ratio is dynamically adjusted based on the percentage of newly submitted jobs with expected resource constraints. As a result, this method can find the best resources to employ during TS to increase makespan, network throughput, resource utilization, and completion time.
Here is the format for the remaining portion of the paper: Section 2 covers a variety of TS techniques that have developed over the years. Section 3 explains the proposed TS algorithm, and Section 4 examines how well it works. Section 5 summarizes the entire work.

2. Literature Survey

For recasting the TS challenge as a Hybrid Flowshop Scheduling (HFS) problem, a Hybrid Discrete Artificial Bee Colony (HABC) algorithm [2] was introduced. To make the HABC algorithm's exploitation and exploration capabilities equal, an improved dynamical perturbation was employed. The exploitation capabilities were further enhanced by applying a deep-exploitation operator. However, it did not consider resource utilization, memory, and bandwidth use as objective functions for scheduling optimal jobs. According to [3], the TS problem was an irregular constrained optimization challenge. To solve it, they used the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) with the Entropy Weight Method (EWM). However, it did not take into account the time and resource constraints. To minimize the cost of executing a job request within a deadline, [4] developed a hybrid system that combines idle timeslot-aware rules with Particle Swarm Optimization (HPSO). A unique particle encoding was used to specify the VM category and scheduling order for each job. A scheduling solution was created by a particle using an idle timeslot-aware decoding process. A repair plan was also established to accommodate the job's incorrect priorities. However, each VM instance was expected to have the same network bandwidth, affecting usage. Using a novel hybrid TS technique called Chemical Reaction-PSO (CR-PSO), developed by Dubey and Sharma, multiple independent jobs were distributed among the available VMs [5]. The CR optimization and PSO were hybridized by combining the features for the optimal schedule sequence. However, the CR-PSO technique overlooks dependent jobs and requires additional factors, such as bandwidth, to enhance the efficacy of the TS. In [6], a metaheuristic model for Dynamic VM Allocation (MDVMA) or optimal TS in dynamic VM distribution in the cloud-computing paradigm was developed. To optimize TS, they utilized the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). This algorithm simultaneously reduces energy use, makespan, and cost. Nevertheless, this algorithm needs additional factors like memory and bandwidth for determining the objective function. A Hybrid Ant Genetic Algorithm (HAGA) was created in [7] for efficient work scheduling in cloud data centers. To determine the load on the VM, a novel technique was employed. The Genetic Algorithm (GA) mutation process has been altered to detect overloaded VMs and allocate tasks accordingly. To boost efficiency, it needs other factors like resource use, etc. In [8], a multi-objective TS using a decision tree technique for job planning and completion. However, it failed to consider memory, bandwidth, etc., when defining the objective function of an effective TS.

To implement QoS-based TS in cloud computing, [9] presented the Load-balancing Ant Colony Optimization (LBACO), which integrates GA and ACO. Nevertheless, to further improve efficiency while solving many objectives, it must use complex optimization methods. The Improved GA for Permutation-based Optimization Problems (IGA-POP) was presented as a semi-adaptive real-time TS technique [10]. To identify different permutations for jobs that have been received at all scheduling rounds, the IGA was implemented. Then, to achieve a greater trade-off between the makespan and the execution cost, the jobs were distributed to the VM according to the best permutation. However, the execution time was high as the number of jobs increased. The TS was developed in [11] using the Advanced Phasmatodea Population Evolution (APPE) algorithm. To prevent the algorithm from reaching local optimization and to equalize its exploration and exploitation capabilities, a restart strategy was used. In addition, the valuation factor was used to identify the best alternatives in terms of load balancing, makespan, and resource costs. However, it was only appropriate for resolving static TS challenges and was not useful for assigning resources at an interval based on the job arrival period. The multi-objective Gaussian Cloud and Whale Optimization Algorithm for TS Strategy (GCWOAS2) [12] was created to achieve TS by adopting an opposition-based learning strategy and a dynamic fine-tuning variable. An opposition-based learning strategy was applied to determine a more appropriate job-VM mapping, whereas the dynamic fine-tuning variable was used to choose the best scheduling. However, it did not perform well in terms of execution time. Table 1 summarizes the above-discussed methods in terms of objectives considered, merits, and demerits.

Table 1: Summary of Existing TS Methods in Cloud Computing

Ref. No.	Methods	Objectives considered	Merits	Demerits
[2]	HABC	Completion time and workload	It reduced completion period and workload concurrently.	It did not consider resource utilization, memory, and bandwidth use as objective functions for scheduling optimal jobs.
[3]	TOPSIS+EWM	Makespan, cost, and energy usage	It reduces the makespan and energy usage efficiently.	It did not take into account the time and resource constraints.
[4]	HPSO	Execution cost	It minimized execution cost significantly.	Each VM instance was expected to have the same network bandwidth, affecting usage.
[5]	CR-PSO	Cost, energy, and makespan	It significantly reduced the execution period.	It overlooks dependent jobs and requires additional factors, such as bandwidth, to enhance the efficacy of the TS.
[6]	NSGA-II	Energy use, makespan, and cost	It effectively decreased energy use, makespan, and cost of the cloud datacenter.	It needs additional factors like memory and bandwidth for determining the objective function.
[7]	HAGA	Execution time	It reduced the execution period of tasks.	To boost efficiency, it needs other factors like resource use, etc.
[8]	Decision tree	Makespan, load balance, and resource usage	It increased resource usage and load balance with reduced makespan.	It failed to consider memory, bandwidth, etc., when defining the objective function of an effective TS.
[9]	LBACO	Activity with a defined number of tasks and required resources	It can manage dynamic assignment of tasks with optimal resource use.	To further improve efficiency while solving many objectives, it must use complex optimization methods.
[10]	IGA-POP	Execution time and cost	It attained a good balance between the makespan and the overall execution cost.	The execution time was high as the number of jobs increased.
[11]	APPE	Makespan, resource cost, and load balancing degree	It has a higher resource use.	It was only appropriate for resolving static TS challenges and was not useful for assigning resources at an interval based on the job arrival period.
[12]	GCWOAS2	Completion time, load cost, and operational cost	It effectively minimized resource use.	It did not perform well in terms of execution time.

From this review, it can be inferred that most of the previous studies considered makespan (execution time and completion time), cost, load balance, energy use, or resource usage as objective criteria. There is a necessity to consider additional factors related to memory usage, bandwidth use, and throughput as objective criteria in multi-objective optimization schemes to further enhance the performance of TS. Thus, this work focuses on considering these additional objective factors to improve the TS efficiency in cloud computing.

3. Proposed Methodology

This section describes the presented hybrid TS algorithm in brief. Figure 1 depicts an overview of the TS model for cloud computing. A task list is used to send a large number of independent job requests to the cloud broker. The CSP checks to determine whether there are adequate cloud resources to meet all job execution criteria. Once there are enough available cloud resources, the scheduler will assign the desired jobs to them. In that case, the cloud service provider will have to hold off on fulfilling the user's request until the necessary resources become available.

3.1. Problem description

At first, the job is evaluated based on a specific deadline for submission and the need for resources. The user submits a runtime and resource demand value during the request. Suppose there are n separate jobs, and each j represents one of the following: submission period r_j, resource demand d_j, actual execution period p_j and requested execution period $\hat{p}_j$. Using m Units of resources, finish a batch of concurrent jobs with high demand on resources, and unpredictable runtime on the cloud. A hybrid planning model that incorporates a linear matching method and backfills jobs with unpredictable runtimes is introduced to address this challenge. The job request uses the GRU network to obtain an accurate estimate of how long the job will take to complete. Afterwards, the trained GRU network sorts jobs into reliable and unreliable queues. In addition, the DOA uses plan-based scheduling for jobs with longer expected runtimes, while the AHP fills the unreliable queue with jobs. There are two types of jobs: reliable and unreliable.

- Jobs with a predicted execution precision of 60% or higher are considered reliable.
- A job is considered unreliable if its estimated runtime precision is less than 60%.

3.2. Definition of objective functions

The presented algorithm takes six primary objectives for optimization, including makespan, resource usage, memory use, bandwidth consumption, execution period, and throughput.
Makespan: It is the total necessary duration from requesting the job to the execution of the job. It is computed by

$$\text{Makespan}=\max\left(MS(VM_x)\right),\ 1\leq x\leq n \tag{1}$$

In Eq. (1), n denotes the quantity of VMs.

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

In Eq. (2), CT stands for the computational time of the job. j_x on VM_y. If j_x is allocated on VM_y; then Assigned(x,y)=1; otherwise, it is 0. The calculation of the computational time required to execute j_x by VM_y is done by

$$CT_{xy}=\sum\left({}^{j_x.MI}\!/_{VM_y.MIPS}\right) \tag{3}$$

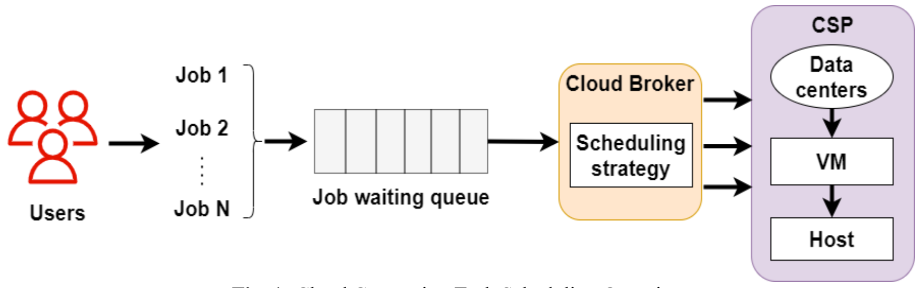

Fig. 1: Cloud Computing Task Scheduling Overview.

The variable j_x.MI in Eq. (3) represents the number of Million Instructions (MIs) of j_x, VM_y.MIPS represents the MI/second of VM_y. Accordingly, the first objective is to decrease the makespan:

$$f_1=\min(\text{Makespan}) \tag{4}$$

Resource Usage Percentage (RUP): It is calculated as the average of the makespan to the maximum makespan of the cloud and expressed as a percentage.

$$\text{Mean}_{RUP}=\frac{\text{Average makespan}}{\text{Cloud makespan}} \tag{5}$$

$$\text{Average makespan}=\sum{}^{MS(VM_x)}\!/_{n},\ 1\leq x\leq n \tag{6}$$

The maximum time it takes to complete all tasks in the cloud is the same as the makespan. If Mean_RUP Is small, then the cloud platform is not making significant use of its computational resources; if it's large, then the opposite is true. The second goal is to raise the Mean_RUP:

$f_2 = \max(\text{Mean\_RUP})$ (7)

Execution period: It is also termed the degree of imbalance, which estimates the Imbalance Degree (IB_Deg) of job scheduling among VMs as per their abilities. It is computed as:

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

In Eq. (8), $\text{Max}_{\text{CTime}_i}$, Min_CTime_i and Mean_CTime_j Denote the highest, lowest, and average execution period of j On all VMs, respectively. The smaller values of IB_Deg State that the job is balanced suitably, whereas greater values state that the load balancing is unsuccessful. Accordingly, the third objective is to decrease. IB_Deg:

$f_3 = \min(\text{IB\_Deg})$ (9)

Throughput: It is the number of tasks finished in a given frame is determined. It is calculated using the makespan as the basis by

$$\text{Throughput} = \frac{\text{Number of jobs}}{\text{Makespan}} \quad (10)$$

A greater value of the throughput is essential for an effective TS algorithm. Accordingly, the fourth objective is to improve throughput:

$f_4 = \max(\text{Throughput})$ (11)

Memory usage: It is the highest memory needed by all VMs for completing the jobs, and is computed by

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

In Eq. (12), AM_x represents the memory usage at the i^{th} VM before completing j, RM_j Stands for the memory that holds the request of j, and TM_x Represents the total memory that is accessible at the i^{th} VM. Consequently, reducing memory usage is the fifth objective:

$f_5 = \min(\text{Memory})$ (13)

Bandwidth consumption: It is the highest bandwidth needed by all VMs for completing the jobs, and is computed by

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

In Eq. (14), AB_x denotes the amount of bandwidth consumption before completing j at i^{th} VM, RB_j denotes the bandwidth holding the request of j, and TB_x denotes the overall bandwidth accessible at i^{th} VM. Accordingly, the sixth objective is to minimize the bandwidth use:

$f_6 = \min(\text{Bandwidth})$ (15)

3.3. Hybrid task scheduling system

The four main components of the suggested hybrid TS system for scheduling are depicted in Figure 2. These include the meta-learning component, the component for tuning the hybridization parameters, the centralized scheduler, and the repository. In the meta-learning part, the runtime of the job will be estimated, and the predicted precision is calculated while the job is being requested. To determine how long it will take for the new job to be deployed to the cloud, the forecasted values are then passed to the central scheduler. For the meta-learning and hybridization tuning of parameter components to take into account actual runtime data, the repository is updated after the job is executed. Included in the repository are the hyperparameters for the GRU network. Because the task is running in the cloud, its runtime value is being added to the repository.

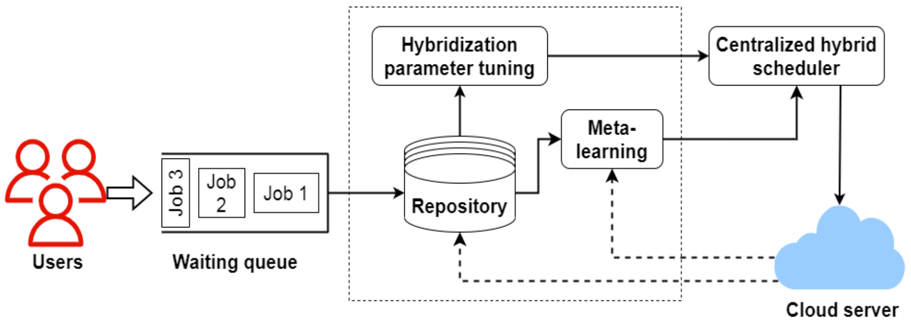

Fig. 2: Schematic Representation of the Proposed Hybrid Task Scheduling in Cloud Computing.

3.3.1. Central scheduler

Based on the data provided by the meta-learning component, tasks are categorized as either reliable or unreliable. The meta-learning unit uses the term "reliable" to describe tasks that it expects to complete. Schedulers that use both plan-based and backfilling approaches make up the hybrid scheduler. The first step is for the central scheduler to determine the beginning interval of each reliable task in the waiting queue using plan-based scheduling. The hybridization tuning of the parameters portion determines the precise proportion of CPU cores needed to complete this process. Using the remaining available resources, AHP with a backfilling scheduler determines the initial period of unreliable jobs after determining the deployment period of reliable jobs. To organize tasks on the cloud, a perfect timetable is applied. The process is maintained whether the job is being requested, completed, or closed.

3.3.2. Parameter tuning for hybridization

The percentage of available resources that are used by the plan-based programming method to manage tasks reliably. The hybridization tuning parameter phase adjusts α, which is called the hybridization parameter. The ratio of the total resource consumption by reliable jobs to the total resource demand by all jobs determines the value of α. This is how the tuning of α is defined:

$$\alpha_{t+1}=\alpha_t * \frac{\sum_{i\in \text{reliable}} d_i}{\sum_{j\in \text{all jobs}} d_j} \tag{16}$$

In Eq. (16), d_j Indicates the resource demand for j. The initial value of α (α_0) It is decided via a grid search.

3.3.3. Meta-learning

Both the estimated runtime and its accuracy are calculated using the meta-learning technique. The GRU estimates the runtime by taking into account the job at the time of request and calculating the probability of the runtime based on that. In addition to rebuilding the model every 24 hours, the hyperparameters of the model are saved in a repository. The meta-learning mainly checks if the jobs are in reliable or unreliable queues.
A point-wise metric is used to evaluate the accuracy of runtime predictions for each job. This metric compares the expected and actual job runtimes, or $(\hat{p}_j)$ and (p_j), respectively.

$$PR_j=\begin{cases}1, & \hat{p}_j=p_j\\ \frac{\hat{p}_j}{p_j}, & \hat{p}_j<p_j\\ \frac{p_j}{\hat{p}_j}, & \hat{p}_j>p_j\end{cases} \tag{17}$$

A. GRU Network Model as Meta-Learning
The GRU network is employed to predict the estimation precision of job runtimes. With the properties from Table 2 and the target values of previously completed jobs as inputs, the GRU model calculates an approximation of the accuracy of the most recently requested tasks.

Table 2: Characteristics applied for Job Runtime Estimation Precision Prediction

Characteristic group	Characteristic name
Running time as requested by the user	$\hat{p}_j$ (reqTime)
Actual CPU and runtime for previously run tasks by the same user	last, beflast, beflast2, lastcpu
Maximum CPU and runtime for previously run tasks by the same user	maxrt, maxcpu
Seasonality characteristics	tod1, tod2, dow1, dow2
The mean and standard deviation of the CPU and runtime for the identical user's jobs	meanrt, stdrt, meancpu, stdcpu
The total number of tasks completed by a single user	prevuser

Figure 3 shows that the GRU network is trained using feature vectors of executed jobs. $f_i=\{z_{1i},\ldots,z_{ki}\}$ and associated estimation precision PR_i. This network converts the attributes and the predicted execution time into precision metrics. The learned model is used to find the precision value of the job. i, $\widehat{PR}_i$, using the function $f_i=\{z_{1i},\ldots,z_{ki},\hat{p}_i\}$. This is followed by the runtime estimation accuracy of the newly required jobs.

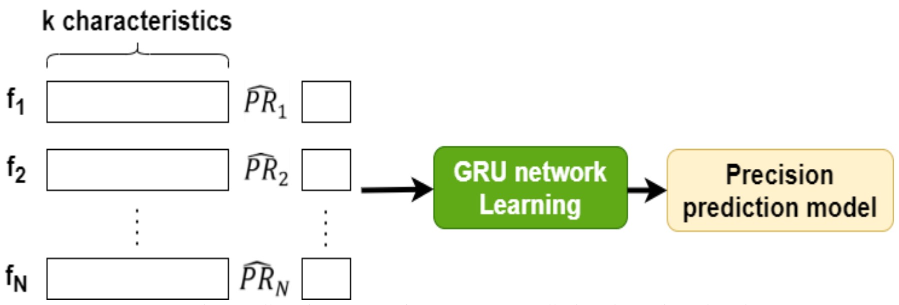

Fig. 3: Understanding the System for Accurate Prediction through Estimation.

Figure 4 shows that the learned model is used to find the precision value of the job. i, $\widehat{PR}_i$, using the function $f_i=\{z_{1i},\ldots,z_{ki},\hat{p}_i\}$. This is followed by the runtime estimation accuracy of the newly required jobs.

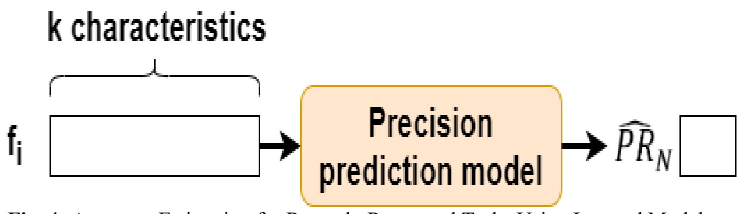

Fig. 4: Accuracy Estimation for Recently Requested Tasks Using Learned Model.

A simplified version of the Long Short-Term Memory (LSTM) network, the GRU combines input and forget gates into one update gate and adds a reset gate. Figure 5 shows that this network modulates unit data without a separate memory cell. At period t, the GRU's activation h_t^j is calculated by linearly interpolating the candidate activation h_{t-1}^j and the previous activation $\tilde{h}_t^j$.

$$h_t^j = \left(1-z_t^j\right) h_{t-1}^j + z_t^j \tilde{h}_t^j \qquad (18)$$

The intensity of the unit's activation modification is controlled by the update gate. z_t^j.

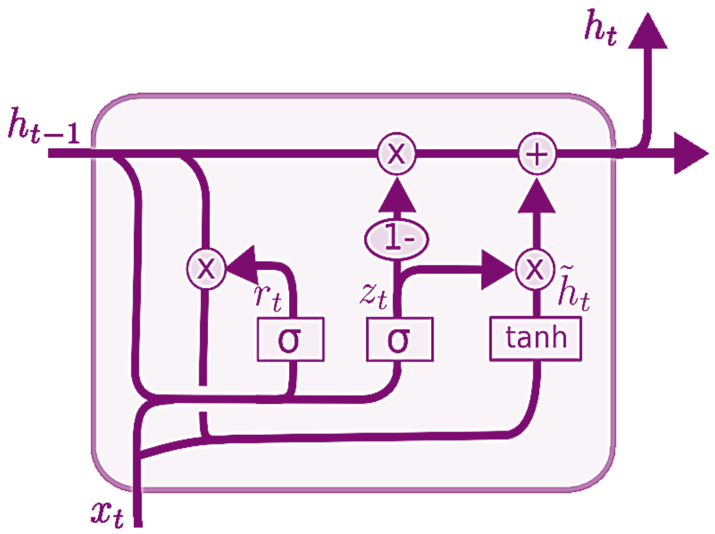

Fig. 5: Structure of Gated Recurrent Unit Network.

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1} + b_z)^j \qquad (19)$$

A calculation similar to that of the update gate is used to determine the candidate activation. $\tilde{h}_t^j$.

$$\tilde{h}_t^j = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)^j \qquad (20)$$

Equation (20) uses $\odot$ as an element-wise multiplication and r_t^j As a set of reset gates. If the reset gate is turned off $\left(r_t^j == 0\right)$The unit is able to erase its previous actions. This is similar to giving the unit access to the input's initial data. A calculation is made for the reset gate. r_t^j by

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1} + b_r)^j \qquad (21)$$

The weight matrices in the given equations are W_z, W_h, W_r, U_z, U_r and U_h, while the bias vectors are b_z, b_r, and b_h. The significance of the previous state can be adjusted by the update gate. z. Some units may have active resetting gates. r for short-term dependencies and active update gates z For long-term dependencies.
The first algorithm explains the steps to take to generate the GRU estimate (meta)˜ with input F. An input set of k-dimensional features with N dimensions, denoted as $F=\{f_1,\ldots,f_N\}$, and the associated target precision values $PR=\{PR_1,\ldots,PR_N\}$, are incorporated into the algorithm. The first step is to fix the GRU. $meta_0(F)$ between F and PR. A decision portion of feature space can be obtained for new data using the trained GRU. The average runtime of jobs in the identical sub-area is used to estimate the target value for new data. The number of epochs that this procedure is carried out for can vary.
Algorithm 1 GRU Network as the Meta-Learner

```
1) Initialize $meta_0 = \arg\min \sum_{i=1}^{N} L(PR_i, \gamma)$
2) for(m=1,…,M)
3)   for(i=1,…,N)
4)   Determine $r_{im} = -\left[\frac{\partial L PR_i meta(f_i)}{\partial meta(f(x_i))}\right]_{f=f_{m-1}}$;
5)   end for
6) Determine the optimal fit for the GRU network using the target $r_{im}$;
7) for(j=1,…,m)
```

```
8) Determine $\gamma_{jm}=\underset{\gamma}{\operatorname{argmin}} \sum_{x_j \in R_{jm}} \left(PR_i meta_{m-1}(f_{m-1}(x_i)+\gamma)\right)$;
9) end for
10) Modify $meta_m(x)=meta_{m-1}(x)+\sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right)$;
11) end for
```

Result: $\widetilde{meta}(x)=meta_M(x)$

3.3.4. Plan-based scheduling for reliable jobs using dove optimization

The DOA was designed with the foraging behavior of the dove [13]. The DOA was chosen because of its strong exploration versus exploitation ratio, which reduces the early convergence of the algorithm. This characteristic allows the algorithm to escape local optima when confronted with a complex planning environment. Its rapid convergence characteristics, when combined with a lightweight search strategy, reduce computational overhead, making it suitable for real-time cloud-task scheduling applications. Besides, multi-objective fitness functions are optimally handled by DOA, which also improves makespan, throughput, memory, and the utilization of bandwidth.

The objective function consists of six variables. $f(\underline{W})=[f_1,f_2,f_3,f_4,f_5,f_6]$, These represent the makespan, resource utilization, execution time (degree of imbalance), throughput, memory utilization, and bandwidth utilization. Each job $\underline{W}$ In the list is characterized as a location with crumbs, and the quantity of crumbs in this location. $\underline{W}$ contains $g(\underline{W})$ Crumbs. The optimal solution identifies the location with the most crumbs.

Step 1: Arrange the number of doves on the solution space according to the number chosen. Assume that N doves are uniformly distributed throughout the space.

Step 2: For each dove d, where, d=1,…,N, set the degree of satiety, s_d^e and the number of epochs, e=0. Two methods are used to initialize the location vector. $\underline{W}_d \subset R^M$ of dove d. The fundamental approach is to randomly start. $\underline{W}_d$ In the neighborhood of the solution space. An alternative approach is to set up a lattice initialized scheme. The procedures are detailed below.

Assume about the smallest hyper-rectangle that can fit in the parameter space. It has all the real numbers for the parameters, represented as $[l_1,u_1],\dots,[l_M,u_M]$ with l_a and u_a Being the bounds of the a-dimensional space in the resultant space. Reducing the n-dimensional hyper-rectangle to a 2D plane is the main idea behind this scheme, which effectively encloses the solution space in a 2D net. To move from 1 to A×B In the rectangular cells, use the integers. i and j.

- The four corner cells are initialized: At the outset, the four neurons positioned at the network's corners have their weight vectors defined as:

$$\underline{w}_{1,1}=(l_1,l_2,\dots,l_M)^T;\ \underline{w}_{A,B}=(u_1,u_2,\dots,u_M)^T;\ \underline{w}_{1,B}=\left(l_1,l_2,\dots,l_{\lfloor\frac{M}{2}\rfloor},u_{\lfloor\frac{M}{2}\rfloor+1},\dots,u_M\right)^T;\ \underline{w}_{A,1}=\left(u_1,u_2,\dots,u_{\lfloor\frac{M}{2}\rfloor},l_{\lfloor\frac{M}{2}\rfloor+1},\dots,l_M\right)^T \tag{22}$$

- Initialization of the cells along the four edges: Set the cell's value on the four edges using:

$$\underline{w}_{1,j}=\frac{\underline{w}_{1,B}-\underline{w}_{1,1}}{B-1}(j-1)+\underline{w}_{1,1}=\frac{j-1}{B-1}\underline{w}_{1,B}+\frac{B-j}{B-1}\underline{w}_{1,1},\ j=2,\dots,B-1$$

$$\underline{w}_{A,j}=\frac{\underline{w}_{A,B}-\underline{w}_{A,1}}{B-1}(j-1)+\underline{w}_{A,1}=\frac{j-1}{B-1}\underline{w}_{A,B}+\frac{B-j}{B-1}\underline{w}_{A,1},\ j=2,\dots,B-1$$

$$\underline{w}_{i,1}=\frac{\underline{w}_{A,1}-\underline{w}_{1,1}}{A-1}(i-1)+\underline{w}_{1,1}=\frac{i-1}{A-1}\underline{w}_{A,1}+\frac{A-i}{A-1}\underline{w}_{1,1},\ i=2,\dots,A-1$$

$$\underline{w}_{i,B}=\frac{\underline{w}_{A,B}-\underline{w}_{1,B}}{A-1}(i-1)+\underline{w}_{1,B}=\frac{i-1}{A-1}\underline{w}_{A,B}+\frac{B-i}{B-1}\underline{w}_{1,B},\ i=2,\dots,A-1 \tag{23}$$

$$\underline{w}_{i,j}=\frac{\underline{w}_{A,j}-\underline{w}_{1,j}}{A-1}(i-1)+\underline{w}_{1,j}=\frac{i-1}{A-1};\underline{w}_{A,j}+\frac{A-i}{A-1};\underline{w}_{1,j}=\frac{i-1}{A-1}\left(\frac{j-1}{B-1}\underline{w}_{A,B}+\frac{B-j}{B-1}\underline{w}_{A,1}\right)+\frac{A-i}{A-1}\left(\frac{j-1}{B-1}\underline{w}_{1,B}+\frac{B-j}{B-1}\underline{w}_{1,1}\right)$$

$$=\frac{\left((j-1)(i-1)\underline{w}_{A,B}+(j-1)(A-1)\underline{w}_{1,B}+(B-j)(i-1)\underline{w}_{A,1}+(B-j)(A-i)\underline{w}_{1,1}\right)}{(B-1)(A-1)} \tag{24}$$

- Initialization of the residual cells: Initialization is done on the weight vectors of the four neurons located at the network's corners. The residual neurons are initialized in a left-to-right and top to down.

Step 3: For each epoch, find the fitness function of the dove by adding up all the crumbs at the position of the d^th dove, where

$$f\left(\underline{w}_j^e\right),\ j=1,\dots,N$$

Step 4: Using the maximum criterion at epoch ePosition the dove d_j^e As it is closest to the greatest number of crumbs.

$$d_j^e=\operatorname{argmax}\left\{f\left(\underline{w}_j^e\right)\right\},\ j=1,\dots,N \tag{25}$$

Step 5: Adjust the level of fullness for all doves by

$$S_j^e=\lambda S_j^{e-1}+e^{\left(f\left(\underline{w}_j\right)-f\left(\underline{w}_{d_f}\right)\right)},\ j=1,\dots,N \tag{26}$$

Step 6: Select the dove with the highest level of satisfaction, denoted as d_s^e Using the following criteria:

$$d_s^e = \underset{1\le j\le N}{\operatorname{argmax}}\{S_j^e\},\ j=1,\ldots,N \tag{27}$$

The dove d_s Chosen by Eq. (27) is the dove with the best foraging efficiency and deserves to be limited by the other doves in the flock.
Step 7: Change the location vector of all doves by

$$\underline{w}_j^{e+1} = \underline{w}_j^e + \eta\beta_j^e\left(\underline{w}_{d_s}^e - \underline{w}_j^e\right) \tag{28}$$

$$\text{Where } \beta_j^e = \left(\frac{S_{d_s}^e - S_j^e}{S_{d_s}^e}\right)\left(1 - \frac{\left\|\underline{w}_j^e - \underline{w}_{d_s}^e\right\|}{\text{maxDist}}\right) \tag{29}$$

$$\text{maxDist: } \max_{1\le j\le N}\left\|\underline{w}_j - \underline{w}_i\right\| \tag{30}$$

In Equation (28), η represents the learning rate for modifying the dove location vector.
Step 8: To reach the termination criterion, go back to Step 3 and add 1 to the number of epochs $e=e+1$. Here is the definition of the termination criterion:

$$\left|f_{d_s}^e - T(e)\right| \le \text{maximum epoch} \tag{31}$$

When the optimization is based on the minimum condition, it indicates that the best solution is to obtain the minimum. $f\left(\underline{w}_j^e\right)$Then the following updates can be made to the equations. (25) and (26):

$$d_j^e = \operatorname{argmin}\left\{f\left(\underline{w}_j^e\right)\right\},\ j=1,\ldots,N \tag{32}$$

$$S_j^e = \begin{cases} \lambda S_j^{e-1} + e^{\left(f\left(\underline{w}_j\right) - f\left(\underline{w}_{d_f}\right)\right)}, & f\left(\underline{w}_{d_f}\right) \neq 0 \\ \lambda S_j^{e-1} + 1, & f\left(\underline{w}_{d_f}\right) = 0 \end{cases},\ j=1,\ldots,N \tag{33}$$

To determine the best way to allocate resources for completing tasks, Algorithm 2 introduces the DOA.
Algorithm 2 Plan-based Scheduling for Predictable Tasks using DOA
Input: With $j\in\{1,\ldots,J\}$ and $x\in\{1,\ldots,X\}$, find the number of reliable jobs PT_j and the number of VM_x.
Output: Designing the most efficient and trustworthy work schedules

```
1) Begin
2) Set the starting parameters for the number of epochs e, the degree of satiety, the number of doves, and their initial locations;
3) while($e<e_{max}$)
4)         Compute the fitness value of all doves;
5)         Place the dove closest to the largest amount of crumbs;
6)         Modify all doves' satiety degree values;
7)         Choose the most satisfied dove with the maximum degree of satiety;
8)         Modify all doves' location vector;
9) end while
10) Optimal fitness and the best dove (reliable work schedules) should be located in the search space.
11) End
```

3.3.5. Backfilling for unreliable jobs using the AHP method

For tasks with unpredictable execution times, the AHP method is used in conjunction with backfilling. This technique uses a Saaty rating scale to establish a reference value for each criterion and to find their relative importance through pairwise comparisons. According to this, the matrix $A_{n\times n}$ is calculated, where n Denotes the number of parameters considered for unreliable jobs in the decision hierarchy. All elements of A Denote the significance of ith parameter related to jth Parameter. The only variables taken into account are the duration of execution (E) and deadline (DT), as this method is designed to resolve conflicts among unpredictable tasks. When A is calculated, it is done by

- When $a_{ij}>1$, ith The parameter is more significant compared to the jth Parameter.
- When $a_{ij}<1$, ith The parameter is less significant compared to the jth Parameter.
- When $a_{ij}=1$Two parameters have equal significance. The entries a_{ij} and a_{ji} Should satisfy below condition:

$$a_{ij} * a_{ji} = 1 \tag{34}$$

The criteria matrix A of unreliable jobs is calculated by

$$A_{n\times n}=\begin{bmatrix}\frac{a_{11}}{a_{11}} & \cdots & \frac{a_{11}}{a_{1n}}\\ \vdots & \ddots & \vdots\\ \frac{a_{n1}}{a_{11}} & \cdots & \frac{a_{n1}}{a_{1n}}\end{bmatrix} \tag{35}$$

The matrix A It is then normalized as:

$$\bar{a}_{ij}=\sum_{j=1}^{n} a_{ij} \tag{36}$$

The weight matrix W Is determined as:

$$W_{n\times 1}=\begin{bmatrix}w_1\\ \vdots\\ w_n\end{bmatrix} \tag{37}$$

$$w_i=\frac{\bar{a}_{ij}}{\sum_{l-1}^{n}\bar{a}_{il}} \tag{38}$$

In the above equations, i,j,l={1,2,…,n}. The eigenvector obtains the relative weights.
It should satisfy AW=λW, where λ denotes the eigenvector of A and W. Therefore, the eigenvector E_1 and E_2 are calculated for the matrix A. The Consistency Index (CI) is calculated to obtain the consistency among the parameters. For each perfectly stable decision maker, CI=0 or λ_{max}=n.

$$CI=\frac{\lambda_{max}-n}{n-1}\lambda_{max} \tag{39}$$

Therefore, the consistency matrix CI is calculated by

$$CI=E_1-E_2 \tag{40}$$

In Eq. (40), E_1 and E_2 Are determined by below steps:

- A=A*A.
- Normalized the row entries by splitting all entries by the summation of the sum of the rows.
- Consider the total row mean and determine the matrix order of n×1 is named an eigen matrix E_1.
- Likewise, calculate E_2 for the matrix A, where A=A*A.

Eigenvectors for DT (E_{DT}) and execution time (E_E) Are calculated. After that, the matrix E is evaluated by

$$E=\begin{bmatrix}E_{DT}(1,1) & \cdots & E_E(1,1)\\ \vdots & \ddots & \vdots\\ E_{DT}(p,1) & \cdots & E_E(p,1)\end{bmatrix} \tag{41}$$

The decision matrix D Defines the references among unreliable jobs. This algorithm considers $\max(D(i,j))$ As the maximum referenced job is compared to the others to solve conflicts among unreliable jobs in scheduling.

$$D=E*E_2 \tag{42}$$

```
Algorithm 3 AHP Backfilling
Input: Unpredictable task set $S=\{UT_1,\ldots,UT_j\}$
Output: Backfill task $UT_b$
1) Initialize $Q'\leftarrow S$;
2) for(i=1,…,j)
3) Calculate a criteria matrix A using Eq. (35);
4) end for
5) Normalize matrix A using Eq. (36);
6) for(i=1,…,n)
7)         Calculate the weight vector W using Eqns. (37) & (38) for each criterion;
8) end for
9) Obtain Eigenvector $E_1$ and $E_2$;
10) Calculate CI using Eq. (40);
11) Obtain the Eigenvector for all criteria $E_{DT}$ and $E_E$;
12) Calculate the final Eigen matrix E using Eq. (41);
13) Calculate the decision matrix D using Eq. (42);
14) Choose $\max(D(i,j))$ and referred to the job as $UT_b$;
15) Return $UT_b$
```

By fusing plan-based and backfilling strategies, this hybrid TS algorithm may plan tasks effectively while improving resource utilization and meeting other requirements.

4. Simulation Results

This section uses the CloudSim API 3.0.3 simulation to evaluate the GRU-DOA-AHP algorithm's performance. Table 3 contains the parameters used in the simulation. The experimental setup determines these parameters; for example, a Windows 10 64-bit machine with an Intel® Core i5-4210 CPU running at 2.80 GHz, 4 GB of RAM, and a 1 TB hard drive is used to simulate the experiments. Current algorithms such as TSMGWO [1], HABC [2], HPSO [4], IGA-POP [10], and GCWOAS2 [12] are compared to the proposed algorithms using different metrics. Makespan, average RUP, throughput, imbalance degree, utilization of memory, and bandwidth consumption are some of the metrics that are taken into consideration; these are described in Section 3.2.

Table 3: Simulation Environment and Parameter Settings

Type	Parameter Settings	Value
Host	Quantity of hosts	100
	Categories of hosts	HP ProLiant ML110 G4 HP ProLiant ML110 G5
The ML110 G4 from HP	Level of Processing Power (PEs) on a single host	4
	MIPS of PE	2060
	Host memory	8GB
	Bandwidth	3Gbps
HP ProLiant ML110 G5	MIPS of PE	3560
	Bandwidth	3Gbps
	Host memory	8GB
	Number of PEs per host	4
VM	Number of VMs	450
	Variety of VMs	Medium Instance with a High-CPU Extra Large Instance Small Instance Micro Instance
High-CPU Medium Instance	MIPS of PE	2500
	Number of PEs per VM	5
	VM memory	1GB
	Bandwidth	118Mbps
Extra Large Instance	MIPS of PE	2000
	Number of PEs per VM	4
	VM memory	4GB
	Bandwidth	118Mbps
Small Instance	Bandwidth	118Mbps
	Number of PEs per VM	3
	VM memory	2GB
	MIPS of PE	1000
Micro Instance	VM memory	1.5GB
	Number of PEs per VM	2
	MIPS of PE	500
	Bandwidth	118Mbps
Cloudlets	Task count	1000
	Total number of instructions (MI) for the task	2500*simulation limit
	Sum of PEs for each demand	2
GRU	Learning rate	0.001
	Window size	25
	Number of neurons in the hidden state	40
	Batch size	40
	Dropout rate	0.3
	Number of epochs	100
	Loss function	Mean square error
DOA	Number of population	120
	λ	0.9
	η	0.18~0.375

4.1. Makespan

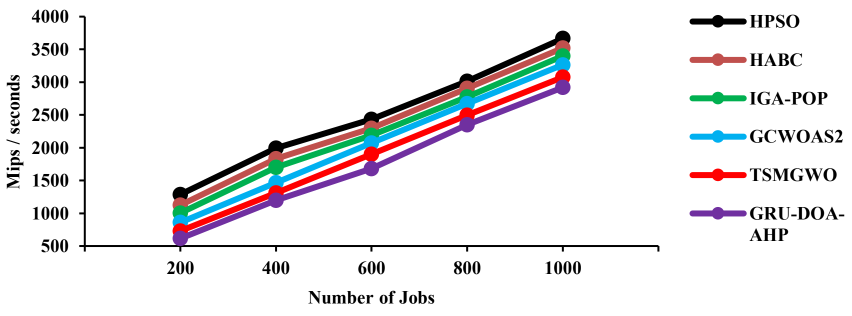

Fig. 6: Makespan vs. Job Count.

The makespan values for the proposed and established hybrid TS algorithms are compared in Figure 6. It is addressed that, compared to the current algorithms, the suggested GRU-DOA-AHP-based TS algorithm achieves higher efficiency using a minimum makespan. Results show that, for 1000 cloud jobs, GRU-DOA-AHP reduces the makespan by 20.38% compared to HPSO, 17.1% to HABC, 14.15% to IGA-POP, 10.6% to GCWOAS2, and 5.1% to TSMGWO algorithms. Accordingly, it is concluded that the GRU-DOA-AHP can attain a significant makespan reduction for cloud systems. This is due to the efficient searchability in the optimization and combining plan-based scheduling with backfilling.

4.2. Mean RUP

Figure 7 displays the average RUP values for both the current and proposed hybrid TS methods. For 1000 cloud jobs, the GRU-DOA-AHP algorithm shows the largest improvement in resource usage, up to 26.03% compared to HPSO, 15% compared to HABC, 9.52% compared to IGA-POP, 5.75% compared to GCWOAS2, and 2.22% compared to TSMGWO algorithms. Accordingly, it is understood that a significant improvement is accomplished in mean resource usage for cloud systems because of scheduling both reliable and unreliable jobs.

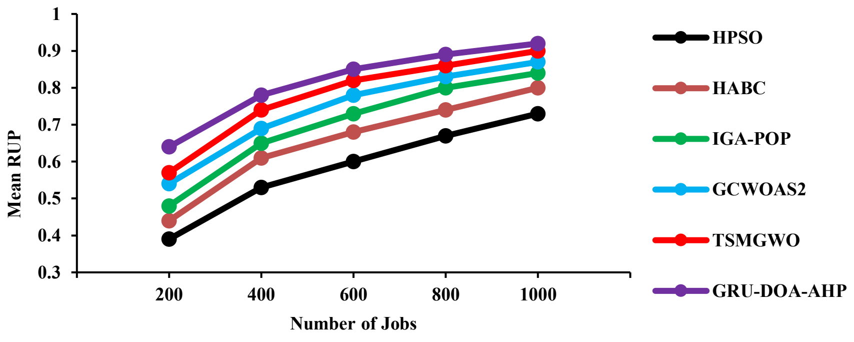

Fig. 7: Mean RUP vs. No. of Jobs.

4.3. Imbalance level

Figure 8 displays the results of the level of imbalance for both existing and proposed hybrid TS algorithms. By reducing the degree of imbalance, the GRU-DOA-AHP algorithm achieves higher efficiency than the other algorithms that are currently available. For 1000 jobs in the cloud system, the GRU-DOA-AHP algorithm offers the largest discount in degree of imbalance at 65% compared to HPSO, 58.82% compared to HABC, 50% compared to IGA-POP, 36.36% compared to GCWOAS2, and 22.22% compared to TSMGWO. Accordingly, it is concluded that the maximum discount in the degree of imbalance is realized by the GRU-DOA-AHP algorithm for cloud systems.

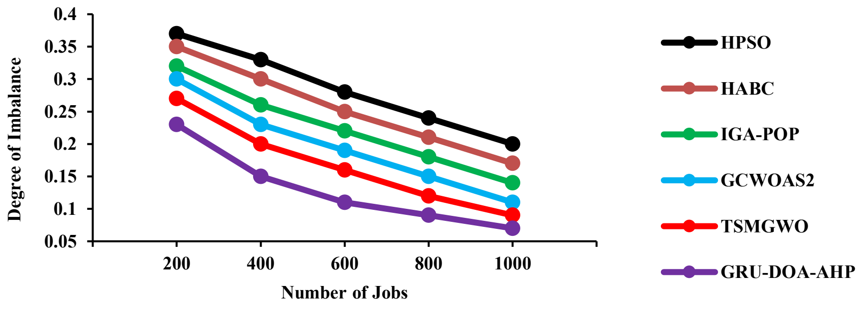

Fig. 8: Degree of Imbalance vs. No. of Jobs.

4.4. Throughput

Figure 9 displays the throughput values obtained for both existing and proposed TS algorithms. When compared to other algorithms, the GRU-DOA-AHP algorithm outperforms them all thanks to its higher throughput values. For 1000 cloud jobs, the GRU-DOA-AHP algorithm achieves a maximum throughput increase of 30% compared to HPSO, 21.88% compared to HABC, 16.42% compared to IGA-POP, 11.43% compared to GCWOAS2, and 4% compared to TSMGWO. The suggested algorithm's enhanced optimization capabilities and hybridization of plan-based and backfilling TS methods allow it to significantly improve throughput for cloud systems.

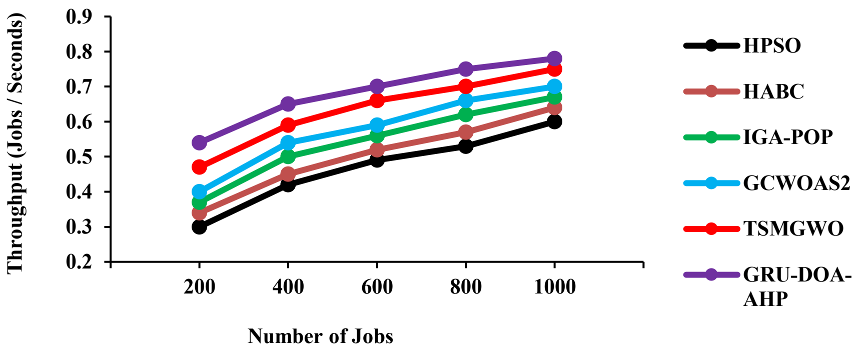

Fig. 9: Throughput vs. Number of Jobs.

4.5. Memory usage

Figure 10 exhibits the memory usage results obtained by the different hybrid TS algorithms. It is observed that the GRU-DOA-AHP algorithm realizes improved performance by lowering the memory usage compared to the other existing algorithms. The GRU-DOA-AHP provides the greatest decrease in memory usage, up to 26.19% over HPSO, 22.5% over HABC, 18.42% over IGA-POP, 13.89% over GCWOAS2, and 8.82% over TSMGWO algorithms for 1000 jobs in the cloud. So, it is understood that a significant minimization is obtained in the memory usage by the proposed algorithm for cloud systems owing to its enhanced searchability, as well as the hybrid TS strategies.

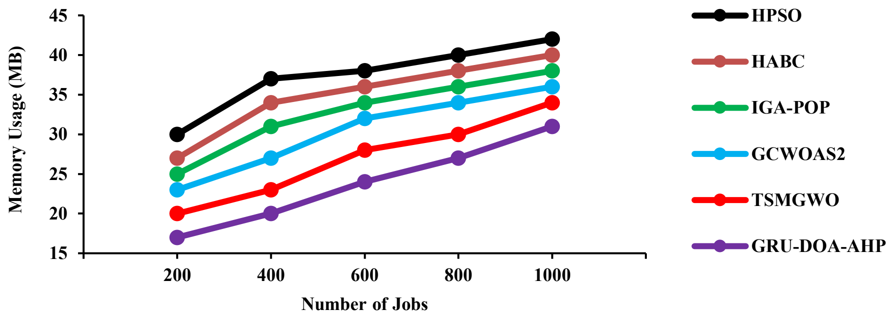

Fig. 10: Memory Usage vs. Number of Jobs.

4.6. Bandwidth consumption

Figure 11 displays the bandwidth consumption results of the various hybrid TS algorithms. When compared to other algorithms, the GRU-DOA-AHP algorithm outperforms them all while consuming significantly less bandwidth. For 1000 cloud jobs, the GRU-DOA-AHP can reduce bandwidth consumption by 16.6%, 15.52%, 10.91%, 8.41%, and 5.31% compared to the HPSO, HABC, IGA-POP, GCWOAS2, and TSMGWO algorithms, respectively. This indicates that the suggested algorithms improve optimization capabilities, and the hybridization of plan-based and backfilling TS methods enables maximum reduction of bandwidth consumption for cloud systems.

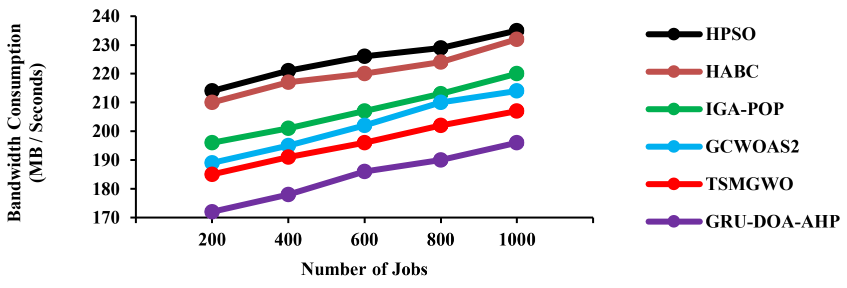

Fig. 11: Bandwidth Consumption vs. Number of Jobs.

5. Discussion

The experimental results indicate that the GRU-DOA-AHP hybrid model makes task scheduling much more efficient than other methods. It benefits from dividing tasks into predictable and unpredictable subsets, eliminating the need for a single optimizer. The GRU meta-learner uses comprehensive behavioral features to enhance real-time prediction accuracy, allowing the scheduler to implement context-specific strategies. Additionally, the adaptive exploration-exploitation framework of the DOA algorithm performs well in scenarios with known execution times, addressing the problem's adversarial challenges. This mechanism averts premature convergence and consistently refines job-VM mappings, leading to improved TS performance. On the other hand, the backfilling method of AHP works well for tasks that are less predictable and helps settle disagreements. It sorts tasks by how long they will take to complete and how important the deadline is, so that shorter or more urgent tasks can be assigned to empty VM slots without affecting the availability of resources for bigger tasks.
The proposed system showcases consistent advantages over competing schemes such as TSMGWO. TSMGWO is great at optimizing multiple objectives; however, it exploits the linear same scheduling for each job category. This system can be inefficient when job runtimes are uncertain, which can lead to VM underutilization or delays. The GRU-DOA-AHP model exhibits dynamic adaptability to workload fluctuations, integrating global task optimization through the DOA algorithm for predictable tasks, while employing local task optimization via the AHP method for uncertain tasks. These two approaches enhance throughput, lessen resource waste, and prevent load imbalance. Consequently, it points out the importance of smart task classification and hybrid scheduling in bolstering the framework's overall performance.

6. Conclusion and Future Work

To improve TS performance in cloud computing, this study develops a hybrid algorithm that relies on both backfilling and linear matching. To estimate the accuracy of job runtime predictions, the GRU network model was initially implemented as a meta-learner. The tasks were categorized as reliable or unreliable according to this assessment. In addition, the DOA was responsible for scheduling the reliable jobs, while the AHP method was used to backfill the unreliable ones. Finally, the results of the simulation experiments demonstrated that, in comparison to the current TS algorithms in cloud computing, the suggested hybrid TS algorithm significantly improves resource utilization and other job completion requirements.

Future research could expand the framework to manage dependent or workflow-based tasks, taking into account precedence constraints and inter-task exchange. Furthermore, implementing the framework in federated or hybrid cloud architectures could be considered, particularly focusing on evaluating the overhead during migrations, latency variations, and coordination among multiple clouds.

References

[1] Alsadie D (2021), TSMGWO: optimizing task schedule using multi-objectives grey wolf optimizer for cloud data centers, *IEEE Access* 9, 37707–37725. https://doi.org/10.1109/ACCESS.2021.3063723.

[2] Li JQ & Han YQ (2020), A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing systems, *Cluster Computing* 23(4), 2483–2499. https://doi.org/10.1007/s10586-019-03022-z.

[3] Kumar MS, Tomar A & Jana PK (2021), Multi-objective workflow scheduling scheme: a multi-criteria decision making approach, *Journal of Ambient Intelligence and Humanized Computing* 12(12), 10789–10808. https://doi.org/10.1007/s12652-020-02833-y

[4] 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. https://doi.org/10.1109/JAS.2021.1003982.

[5] Dubey K & Sharma SC (2021), A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing, *Sustainable Computing: Informatics and Systems* 32, 1–20. https://doi.org/10.1016/j.suscom.2021.100605

[6] Alsadie D (2021), A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers, *IEEE Access* 9, 74218–74233. https://doi.org/10.1109/ACCESS.2021.3077901.

[7] Ajmal MS, Iqbal Z, Khan FZ, Ahmad M, Ahmad I & Gupta BB (2021), Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers", *Computers and Electrical Engineering* 95, 1–15. https://doi.org/10.1016/j.compeleceng.2021.107419.

[8] Mahmoud H, Thabet M, Khafagy MH & Omara FA (2022), Multiobjective task scheduling in cloud environment using decision tree algorithm, *IEEE Access* 10, 36140–36151. https://doi.org/10.1109/ACCESS.2022.3163273.

[9] Sharma N & Garg P (2022), Ant colony based optimization model for QoS-based task scheduling in cloud computing environment, *Measurement: Sensors* 24, 1–9. https://doi.org/10.1016/j.measen.2022.100531.

[10] Abohamama AS, El-Ghamry A & Hamouda E (2022), Real-time task scheduling algorithm for IoT-based applications in the cloud–fog environment, *Journal of Network and Systems Management* 30(4), 1–35. https://doi.org/10.1007/s10922-022-09664-6

[11] Zhang AN, Chu SC, Song PC, Wang H & Pan JS (2022), Task scheduling in cloud computing environment using advanced phasmatodea population evolution algorithms, *Electronics* 11(9), 1–16. https://doi.org/10.3390/electronics11091451

[12] Ni L, Sun X, Li X & Zhang J (2021), GCWOAS2: multiobjective task scheduling strategy based on Gaussian cloud-whale optimization in cloud computing, *Computational Intelligence and Neuroscience* 2021, 1–17. https://doi.org/10.1155/2021/5546758

[13] Su MC, Chen JH, Utami AM, Lin SC & Wei HH (2022), Dove swarm optimization algorithm, *IEEE Access* 10, 46690–46696. https://doi.org/10.1109/ACCESS.2022.3170112.