applied sciences

MDPI

Article

MONWS: Multi-Objective Normalization Workflow Scheduling for Cloud Computing

Vamsheedhar Reddy Pillareddy and Ganesh Reddy Karri *

School of Computer Science and Engineering, VIT-AP University, Amaravati 522237, India
* Correspondence: ganesh.reddy@vitap.ac.in

Abstract: Cloud computing is a prominent approach for complex scientific and business workflow applications in the pay-as-you-go model. Workflow scheduling poses a challenge in cloud computing due to its widespread applications in physics, astronomy, bioinformatics, and healthcare, etc. Resource allocation for workflow scheduling is problematic due to the computationally intensive nature of the workflow, the interdependence of tasks, and the heterogeneity of cloud resources. During resource allocation, the time and cost of execution are significant issues in the cloud-computing environment, which can potentially degrade the service quality that is provided to end users. This study proposes a method focusing on makespan, average utilization, and cost. The authors propose a task's dynamic priority for workflow scheduling using MONWS, which uses the min-max algorithm to minimize the finish time and maximize resource utilization by calculating the dynamic threshold value for scheduling tasks on virtual machines. When the experimental results were compared to existing algorithms, MONWS achieved a 35% improvement in makespan, an 8% increase in maximum average cloud utilization, and a 4% decrease in cost.

Keywords: cloud computing; min-max; threshold value; task scheduling; workflow scheduling

check for updates

Citation: Pillareddy, V.R.; Karri, G.R. MONWS: Multi-Objective Normalization Workflow Scheduling for Cloud Computing. *Appl. Sci.* **2023**, *13*, 1101. https://doi.org/10.3390/app13021101

Academic Editors: Ce Li and Bei Guan

Received: 3 December 2022
Revised: 6 January 2023
Accepted: 11 January 2023
Published: 13 January 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Many scientific tasks in the fields of bioinformatics, health care, physics, and astronomy have been observed to require the execution of complex jobs which are, in turn, represented as workflows [1]. Such workflows can be defined as a Direct Acyclic Graph (DAG) [2]. In DAG, nodes are represented as tasks and the edges indicate the data dependency between two nodes. In the aforementioned fields, the workflows are large and complicated in structure [3]. This necessitates more storage and higher computational power. Therefore, various projects are designed in a grid environment such as GrADS, Pegasus, or ASKALON for pre-processing and managing the complex workflows. In recent years, cloud computing has been noted to be the most suitable for executing large workflow applications due to its provision of unlimited resources using the virtualization concept [4].

In cloud computing, users are charged on a pay-per-use model [5,6] because resources are set up dynamically and are based on the need of workflows and their deadlines [7]. This allows for a better management of both the quality and quantity of resources than other distributed systems, such as grids, can achieve [8,9]. However, in order to schedule large scientific-workflow applications, efficient scheduling schemes for processing are deemed essential.

In cloud computing [10], the primary challenge of workflow scheduling [11] lies in assigning each job or task to a virtual machine for execution [12]. Figure 1 illustrates the sample workflow, defined by a set of nine tasks with dependencies between tasks. The output of one task serves as the input for another task.

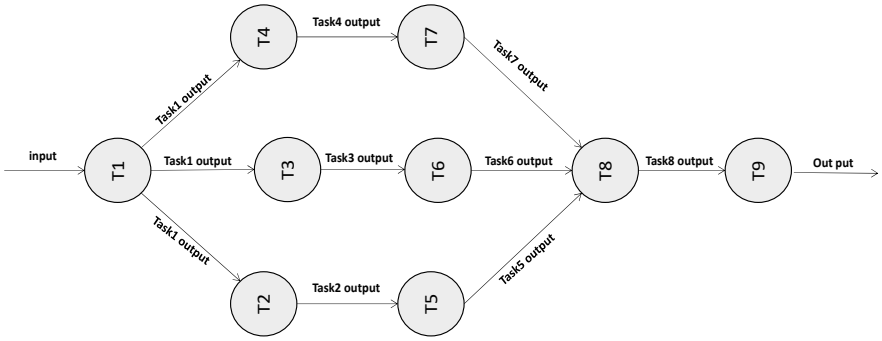


Figure 1. Sample Workflow with nine tasks.

Scheduling plays a vital role in minimizing energy consumption, makespan, and cost while maximizing resource utilization. The existing methods are inadequate for finding optimal solutions. To address this issue, this paper presents a dynamic priority approach for workflow scheduling in cloud computing.

The main contributions of this paper are as follows:

(1) Identifying the best workflow is a challenging task in scheduling. In this paper, based on the study, we select "Cybershake" as the best workflow;
(2) Handling dynamic traffic in cloud computing is very difficult. Static thresholds fail to schedule tasks optimally. In our proposed approach, we define a dynamic threshold value based on the task execution time, workflow model, and availability of resources;
(3) In this paper, we present an extensive literature review for selecting multi-objective parameters for task scheduling;
(4) We exemplify our proposed work with standard numerical values to prove the efficiency of a framework for cloud computing;
(5) To make our proposed algorithm more realistic, we include the data transfer time between tasks;
(6) To calculate the efficiency of proposed algorithm, simulation results are compared with the five existing algorithms.

The article is organized as follows. Section Two discusses the related work in brief. Section Three presents the scheduling problem formation and Section Four elucidates the proposed algorithm. Section Five presents the results of the simulation and performance evaluations and Section Six notes the conclusions derived from the current study.

2. Related Work

Numerous heuristic scheduling algorithms have been proposed in past studies to address the DAG-workflow scheduling problem [13]. Heuristic scheduling algorithms are further classified into list scheduling, cluster scheduling, and group scheduling.

List scheduling is a widely used scheduling algorithm wherein [14] each task and edge is assigned a weight. Based on the assigned weightage, a task list order is then created by prioritizing each task [15]. Following this, tasks are then chosen by their priority to be scheduled on the host, which helps to minimize predetermined cost functions. HEFT [16] and CPOP [17] are two other common list scheduling heuristics used. Group Scheduling is the most popular heuristic scheduling algorithm. In Group Scheduling [18], the tasks are grouped based on their constraints. In DAG, the tasks are partitioned into different levels; therefore, no dependency exists among tasks grouped in the same level. These tasks are then assigned using static scheduling algorithms. Cluster scheduling, on the other hand, is a type of scheduling algorithm that is used for an unbounded number of computing resources.

Zhou et al. [19] proposed a new algorithm, the MW-HBDCS, which is a budget-based scheduling system that works with deadline constraints. The algorithm has two stages: (1) task selection and (2) resource selection. In the first phase, the task-ready pool is formed by assigning each task a rank or priority. In the second phase, the resources are presented

with pooled tasks based on their priority or rank. This algorithm outperformed the existing methods in cases of deadline and budget constraints. However, the study did consider important parameters, such as cost and makespan, for making them more efficient.

In another study, Nasr et al. [20] proposed a hybrid algorithm for workflow scheduling based on deadline constraints. The study combined chemical-reaction optimization (CRO) and ant-colony optimization (ACO) as CR-AC for minimizing the workflow scheduling costs for cloud computing. Initially, the study noted that the CRO found the best optimal solutions with low time complexity. In the next phase, ACO was applied to reduce the overall cost and enhance the quality of the solution. The CR-AC outperformed existing algorithms in terms of cost and makespan. However, the study also noted that the time complexity increased energy consumption, and load balancing was not considered for the research.

Verma et al. [21] proposed a cost- and time-constraint-based workflow scheduling for cloud computing to place the trade-off in cost between the cloud consumer and the provider. BDHEFT consists of two phases: service level and task level scheduling. In the service level scheduling, tasks are scheduled in a descending order depending upon rank, whereas in task level scheduling, tasks are scheduled on the best-possible resource available at minimum cost. The experimental results indicated that existing works were efficient in terms of time and cost, whereas a complex workflow process resulted in inefficiency.

A study by Chakravarthi et al. [22] proposed a normalized, budget-aware workflow scheduling (NBWS) algorithm. The algorithm proceeded through a min-max normalization process, which was then followed by calculating an expected reasonable budget for assigning workflow tasks to one of the VMs. The proposed NBWS consisted of the phases of task selection and resource selection. In the task selection phase, tasks were partitioned at different levels of workflow based on the priorities and dependencies of tasks. In the resource selection phase, an estimated reasonable budget was calculated to schedule the task on VMs. This technique outperformed existing methods in terms of budget when executed on standard workflow applications. However, the impact of communication among tasks was not considered in this study.

Zhou et al. [23] presented a new-list workflow scheduling using the fuzzy-based HEFT(FDHEFT), an algorithm that was classified into two phases of task prioritization and instance selection. In the task prioritization phase, tasks were sorted in descending order of high-rank values, i.e., high-rank values were assigned with higher priority. In the second phase, the FDHEFT sorted all the scheduled solutions produced according to their fuzzy dominance values and chose the best K solutions. The main objective of this algorithm was to minimize the makespan and cost of scheduling. However, the study noted that this algorithm increased the computation time.

Overview of workflows:

Cloud computing has recently been used by commercial and scientific applications to access storage and computing infrastructure. These applications are usually created as a workflow.

Depending on the complexity of the problem, workflows in past studies have been grouped as memory-intensive-, CPU-intensive-, or input/output-intensive-based workflows [24].

Montage workflow:

The Montage workflow (Figure 2) is an input/output-intensive astronomy workflow used to construct custom mosaics of the sky depending on the size of the input images. The workflow structure is modified to accommodate for increases in inputs, which then correspond to an increase in the computational jobs.

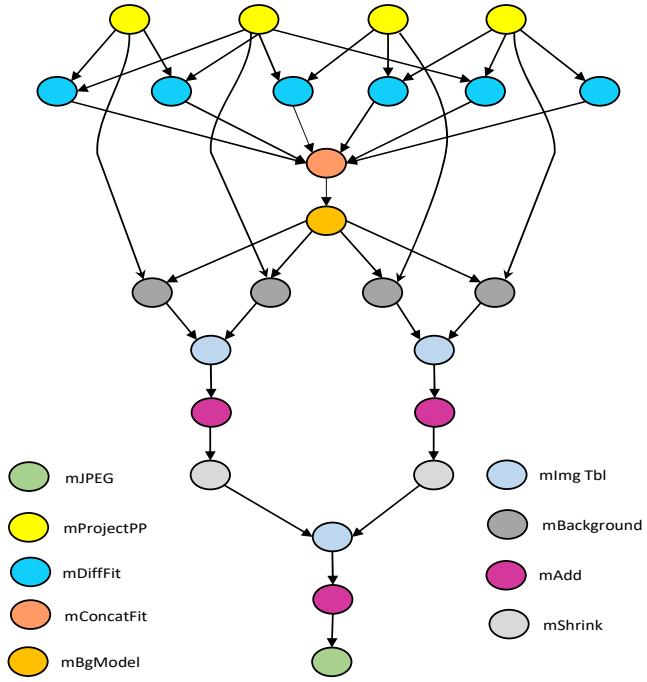


Figure 2. Montage workflow.

Broadband workflow:

The Broadband workflow(Figure 3) was primarily designed to integrate the simulation codes for earthquake motions and their associated calculations. Based on the types of computations performed at each stage, the broadband workflow can be divided into four distinct phases.

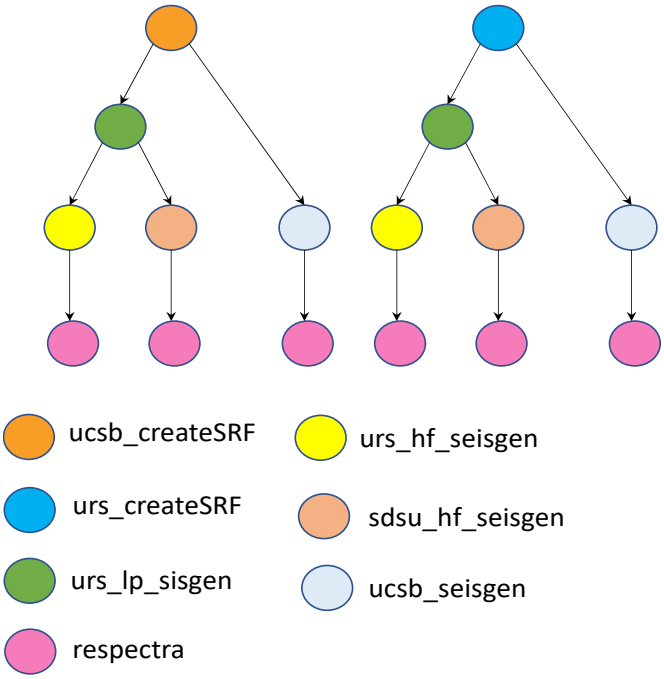


Figure 3. Broadband workflow.

LIGO:

The LIGO (Figure 4) is a memory-intensive application used in the field of physics to detect gravitational waves. This workflow is extremely complex and is made up of several sub-workflows.

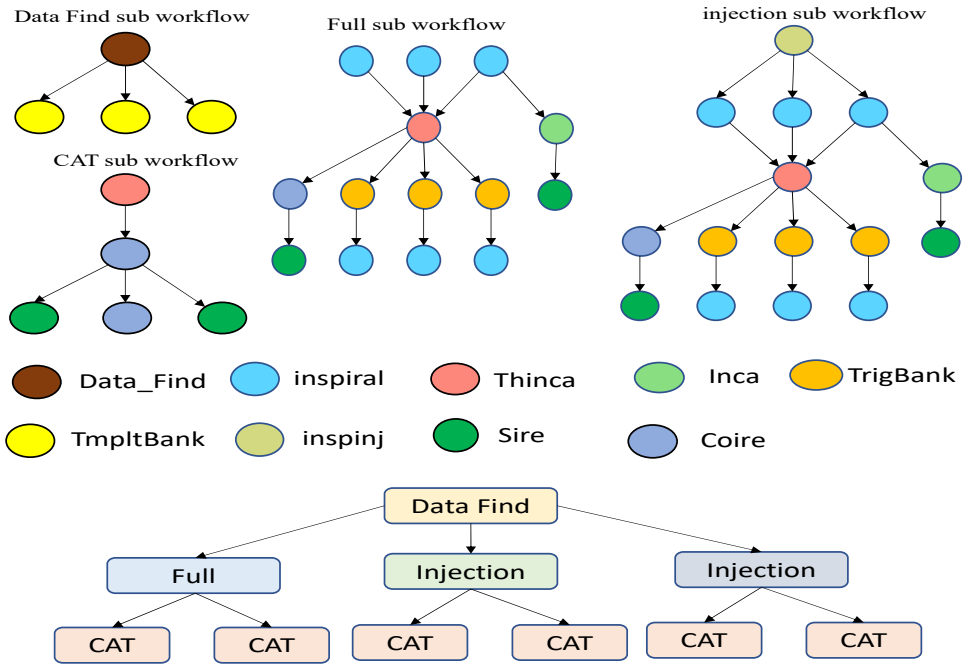


Figure 4. LIGO.

Epigenomics:

Epigenomics (Figure 5) is a largely pipelined application, with multiple pipelines processing independent data chunks in parallel. It begins with FastQSplit jobs and splits the lane into multiple chunks depending on the partition factor. The noise and contaminated data are then filtered from the chunks. The sol2sanger utility then converts each chunk's data into the Maq DNA sequence-mapping software format. The Fastq2bfq then converts the data into the binary fastQ format for faster processing and reduced disk usage. The map utility then aligns the remaining sequences with the reference genome. Next, mapMerge jobs combine individual map process results. After merging, the maqIndex utility retrieves reads about a specific region from the merged alignment file.

Cybershake:

Cybershake workflow (Figure 6) is a data- and memory-intensive workflow application used by the Southern California Earthquake Center. Cybershake can perform computations on particularly large datasets. The workflow starts by generating the strain green tensors (SGT) data from finite simulations derived from the master SGT files. The ExtractSGT job extracts rupture-specific SGT files from the master SGT. Thus, ExtractSGT jobs can be considered to be data partitioning. Seismogram-synthesis jobs generate synthetic seismograms for each rupture variation. Peak intensity values, specifically pertaining to spectral acceleration, are then calculated for each synthetic seismogram by the PeakValueCalcOkaya jobs. The ZipSeismograms and ZipPeakSA jobs collect and compress the synthesized seismograms and peak intensities, which are then staged and archived. These jobs are categorized as simple data collection jobs.

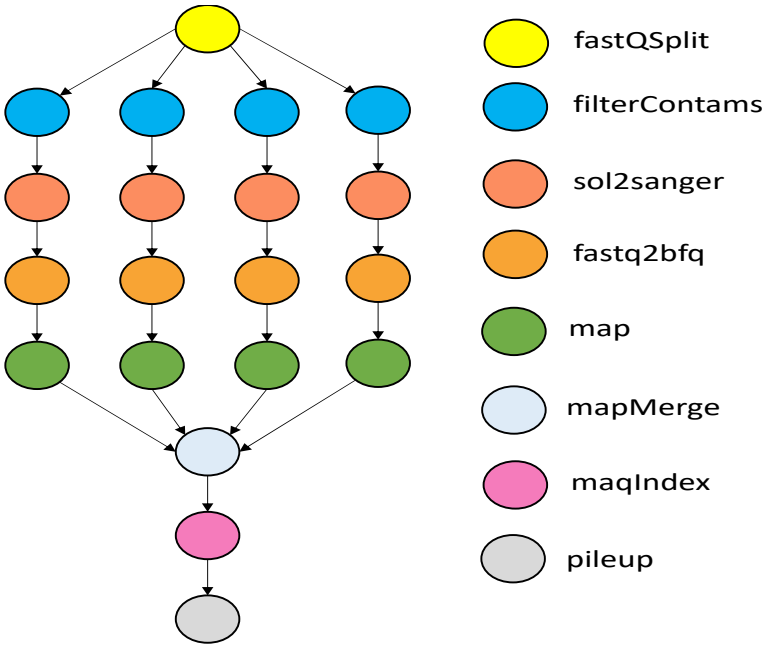


Figure 5. Epigenomics.

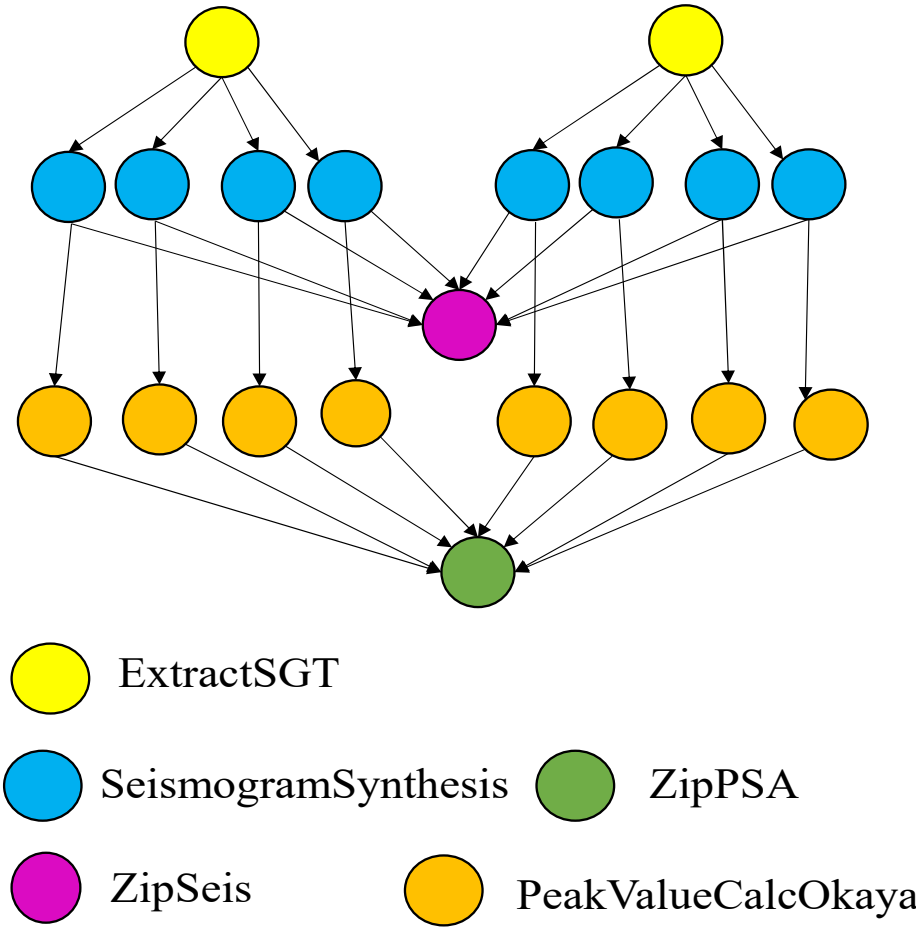


Figure 6. Cybershake.

Table 1 summarizes the existing workflow-scheduling algorithms for cloud computing. This summary also notes where existing algorithms can be improved pertaining to scheduling performance by considering parameters such as cloud utilization, resource utilization, load balancing, and energy consumption. In order to overcome the above unaddressed issues, the current study proposes the MONWS algorithm to enhance the performance of scheduling in cloud computing and Table 2 summarizes the list of notations.

Table 1. Summary of workflow scheduling algorithms for cloud computing.

Ref.	Technique	Objectives										Tools
		Makespan	Energy Consumption	Cost	Budget	Cloud Utilization	Deadline	Execution Time	Response Time	Resource Utilization	Load Balancing	
[25]	Modified HEFT	✓	-	-	-	-	-	-	-	-	-	Cloudsim
[26]	BDAS	-	-	-	✓	-	-	✓	-	-	-	Cloudsim
[27]	MinIncreaseInEnergy, NoIdleMachineECTC, MaxUtilECTCandNoIdleMachineMaxUtil	-	✓	-	-	-	-	-	-	✓	-	Cloudsim
[22]	NBWS	-	-	-	✓	-	-	-	-	-	-	Cloudsim
[28]	BHGALO-GOA	✓	-	✓	-	-	-	-	-	-	-	Workflowsim
[13]	DAGMap	-	-	-	-	-	-	✓	-	✓	-	Grid
[29]	NMMWS	✓	-	-	-	✓	-	-	-	-	-	MATLAB
[19]	MW-HBDCS	-	-	-	✓	-	✓	-	-	-	-	SimGrid
[23]	FDHEFT	-	-	✓	✓	-	-	-	-	-	-	Realtime
[21]	BDHEFT	-	-	✓	-	-	-	-	-	-	-	Cloudsim
[30]	Resource ranking and partitioning	-	-	-	-	-	-	-	✓	-	-	Cloudsim
[31]	BB-BC	-	-	✓	-	-	-	✓	-	-	-	Cloudsim
[32]	PEFT	-	-	-	-	-	-	-	-	-	✓	Workflowsim
[33]	MCDM	✓	-	✓	-	-	-	-	-	-	-	Realtime
[34]	HBCWSP	✓	-	-	✓	-	-	-	-	-	-	Realtime
[35]	HEFT-ACO	✓	-	✓	-	-	-	-	-	-	-	Workflowsim
[36]	EM-WOA	-	✓	-	✓	-	-	-	-	-	-	cloudsim
[37]	NRBWS	✓	-	-	✓	-	-	-	-	-	-	Cloudsim

Note: ✓ indicates considered by author and—indicates not considered by authors.

Table 2. List of Notations.

Abbreviations	Definitions
DTT	Data transfer time
DTTp,q	Data transfer time from T_p to T_q
CS	Cloud server
E^{ct}	Estimated computation time
Ms	Makespan
A_{cu}	Average cloud utilization
t_p	Predecessor task
t_s	Successor task
E^{st}	Early start time
E^{ft}	Early finish time
N_E^{ct}	Normalized estimated computation time
Pt_{vm}	Processing time on VM
R_{same}	Same resource
Th(dTi)	A threshold value of each task, T_i
RLi	Ready List of tasks at level 1 of DAG
$T_N_E^{ct}$	Temporary normalized estimated computation time
T_E^{ct}	Temporary estimated computation time
Act	Actual completion time

3. Scheduling Problem Formation

This section addresses the dynamic workflow scheduling problem of allocating resources for each task in the workflow by minimizing the makespan, cost, and average cloud utilization.

Application Model

The workflow is a directed acyclic graph (DAG) [38] which is represented by D = (T, E), where T = {T_1, T_2, . . . , T_n} is indicative of a set of tasks and E represents the set of directed edges between the tasks, standing dependencies, or communication between the set of tasks [39]. Each directed edge e_{ps} = (T_p,T_s) belongs to E, where tasks T_p and T_s are indicative of precedence constraint. This means that the task, T_s, cannot start its execution until task T_p is completed. In workflow applications, a T_p without any predecessors is referred as an entry task, T_{entry}, and a T_s without any successor task is referred to as an exit task, T_{exit}. If the workflow application contains multiple instances of T_{entry} or T_{exit}, then a virtual edge with zero computation time and zero execution time is to be assumed.

Scheduling Factors:

Data Transfer Time:

$$DTT = \begin{array}{c} \\ T_1 \\ T_2 \\ \vdots \\ T_n \end{array} \begin{array}{c} \begin{array}{cccc} T_1 & T_2 & \dots & T_n \end{array} \\ \begin{bmatrix} 0 & DTT_{1,2} & \dots & DTT_{1,n} \\ 0 & 0 & \dots & DTT_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \end{array} \quad (1)$$

In Equation (1), DTT refers to the data transfer time and the element $DTT_{p,q}$, $1 \leq p \leq n$ and $1 \leq q \leq n$ indicates the data transfer time between T_p and T_q.

Estimated Computation Time:

The task execution time may vary depending on the VMs. The estimated computation time (E^{ct}) is calculated according to Equation (2).

$$E^{ct} = \begin{array}{c} \\ \\ T_1 \\ \vdots \\ T_n \end{array} \begin{array}{c} \begin{array}{cccccc} cs_1 & & \cdots & & cs_m & \\ vm_1 & \cdots & vm_{n|cs_n|} & \cdots & vm_{n|cs_m|} & \end{array} \\ \begin{bmatrix} Et_{1,1} & \cdots & Et_{1|CS_1|} & \cdots & Et_{1|CS_m|} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ Et_{n,1} & \cdots & Et_{n|CS_1|} & \cdots & Et_{1|CS_m|} \end{bmatrix} \end{array} \tag{2}$$

where $Et_{p,q}$, $1 \leq p \leq n$ and $1 \leq q \leq |CS_m|$ indicates the estimated computation time of T_p on the VM of the cloud server $|CS_m|$.

Makespan:

Makespan is referred to the time it takes to complete a given task. Minimizing makespan is a popular optimization criterion when scheduling tasks because most users want their applications to run as quickly as possible. Makespan, M_s, is calculated as per Equation (3):

$$M_s = \max(\text{Makespan}(CS_m)), 1 \leq CS \leq m \tag{3}$$

where M_s is the overall makespan of tasks on the cloud server, CS_m.

Average Cloud Utilizations:

The optimal utilization [40] of resources on cloud servers is known as average cloud utilization and it is calculated as represented in Equation (4).

$$A_{cu} = \frac{1}{N} \sum_{m=1}^{N} U(CS_m), 1 \leq m \leq N \tag{4}$$

In this equation, A_{cu} indicates the average cloud utilizations, CS_m indicates cloud servers, and $U(CS_m)$ is the utilization of m number of cloud servers, calculated as per Equation (5).

$$U(CS_m) = \frac{1}{|CS_m|} \sum_{n=1}^{CS_m} U(vm_n), 1 \leq n \leq N \tag{5}$$

In the above equation, $U(vm_n)$ indicates the utilization of virtual machines and it is calculated as per Equation (6).

$$U(vm_n) = \frac{M_s(vm_n)}{M_s(CS_m)} \tag{6}$$

The utilization of virtual machines is defined as the overall consumption time taken by virtual machines with respect to the makespan of cloud servers, CS_m.

Early start time:

The early start time of a task, t_i, on virtual machines is calculated using Equation (7).

$$E^{st} = \max\{access_{vm}, \max_{t_p \in pre(t_i)} \{AFT_{t_p} + DTT_{t_p,i}\} \tag{7}$$

where $access_{vm}$ is the minimum access time of the VMs, $AFT_{tp,i}$ indicates the actual finish time of the predecessor task, t_p, and $DTT_{tp,i}$ represents the data transfer time of predecessor task, t_p, to the current task, t_i.

Early Finish time:

The early start time of a task t_i on VMs is the summation of the estimated computation time and the early finish time on the VM, which is represented by Equation (8).

$$E^{ft} = E^{ct} + E^{st} \tag{8}$$

Normalized Estimated Computation Time:

$$N_E^{ct} = \frac{E^{ct} + \min_{E^{ct}}}{\max_{E^{ct}} - \min_{E^{ct}}} \tag{9}$$

By using Equation (9), the N_E^{ct} is initially calculated at level 1 of the workflow. Following this, the NE^{ct} is updated as $N_T_E^{ct}$ because when any task is assigned to the VM the previous E^{ct} needs to be updated as T_E^{ct}.

Execution Cost (EC):

$$EC = \frac{Pt_{vm} * C_i}{B_i} \tag{10}$$

The execution cost on VMs is calculated as per Equation (10). It is the ratio of the product between the processing time, Pt_{vm}, and cost per interval C_i to billing intervals B_i, in which the processing time on VMs is calculated according to Equation (11)

$$Pt_{vm} = E^{ct} + \sum_{1}^{d} (DTT_{t_i} + Rsame) + delay \tag{11}$$

Rsame is 0 when T_p and T_s are running on the same resource (cloud server). Otherwise, it is considered as 1.

4. Proposed Algorithm

The main objective of this study was to develop an algorithm (Algorithm 1). that could potentially minimize makespan and cost while maximizing cloud utilization.

Initially, all tasks of a given workflow are divided into a bag of tasks at each level. Next, we begin iterating the task schedule with the following first bag of tasks (BoT). Using the min-max algorithm, the estimated computation time value for all tasks in the BoT is normalized. Further, we divide all the tasks of the BoT into b_large and b_small groups by applying a dynamic threshold value. Finally, we schedule each batch using a minimum completion time algorithm. The large batch is processed first, followed by the small batch.

The proposed MONWS algorithm works in two distinct phases viz. the normalize ECT phase and the VM selection phase.

4.1. Normalize ECT

In this phase, the unscheduled tasks from the workflow are moved into the ready list, RLi, in each level. Initially, the b_large and b_small groups will be empty.

In each level of the DAG (workflow), the W,E^{ct} of each task T_i is normalized using the min-max algorithm to compute N_E^{ct}. The threshold value for each task is then calculated as follows.

$$Th(d_{T_i}) = \frac{\sum \max(E^{ct})}{\sum \max(DTT_{p,q})} \qquad \text{ifTaskparent} T_p = !0 \tag{12}$$

$$Th(d_{T_i}) = 1 \text{ fixed} \qquad \text{ifTaskparent} T_p = 0 \tag{13}$$

If a task has no parent and no data transfer time, then Threshold $Th(d_{Ti})$ is fixed (i.e., 1). If the task has parent, then the above Equations (12) and (13) are calculate and $N_E^{ct} \geq Th(d_{Ti})$ is checked. A move into b_large is then made; otherwise, a move into b_small is made. This is done from top to bottom in the workflow.

4.2. VM Selection

In this phase, tasks are scheduled for VMs by selecting the $E^{ct}i,j$ using the minimum execution time. E^{ct} is then updated as $N_T_E^{ct}$ for subsequent tasks. This procedure is repeated until all present tasks have been scheduled.

Algorithm 1: Proposed MONWS Algorithm

Input: workflow with n number of tasks
Output: Scheduling on available VMs on m number of cloud servers

```
Read E^ct and DTT of the given workflow D (T, E)
while
   for every task T_i in each level
     /*Generate Ready Task List*/
      while
            if tasks are unscheduled do b_large is empty
                  for every task T_i in each level
                  add T_i to Ready List RLi
                  end for
            end if
      end while
     /*Compute Early finish time*/
      for every task T_i Ready List RL_i do
            if task T_i has no T_p
                  E^ct= available_vm+Ect_vm
            else
                  for every vm and T_p of T_i
                        if taskvmmap(T_p) =! Vm
                              if E^ct<E^st
                                 E^ct=E^st
                              endif
                        else
                              if E^ct<max(available_vm,Act)
                                 E^ct=max(available_vm,Act)
                              endif
                        endif
                  endfor
            endif
      endfor
/*Partition of tasks*/
          Find min(E^ct)
          Find max(E^ct)
          Find N_E^ct
          Find threshold Th(dt)
          if max(N_E^ct)>= th(dt)
                add in to b_large
          else
                add into b_small
          endif
     if b_large =! Empty
                find min(E^ft(i,j)
                taskvmmap(T_i)=vm
                Act = E^ft
                E^st = 1
                remove T_i from b_large
     else
                find min(E^ft(i,j)
                taskvmmap(T_i) = vm
                Act = E^ft
                E^st = 1
      endif
endfor
endwhile
```

4.3. MONWS Framework

This study considers the Cybershake workflow application as an example, which is illustrated in Figure 7. The design has twenty tasks, T = {T_1, T_2, ...,T_{20}}, with five levels and thirty-two edges between the tasks. The Data Transfer Time matrix, represented as DTTp,s, is highlighted in Table 3 and is representative of the data transfer time among the different tasks.

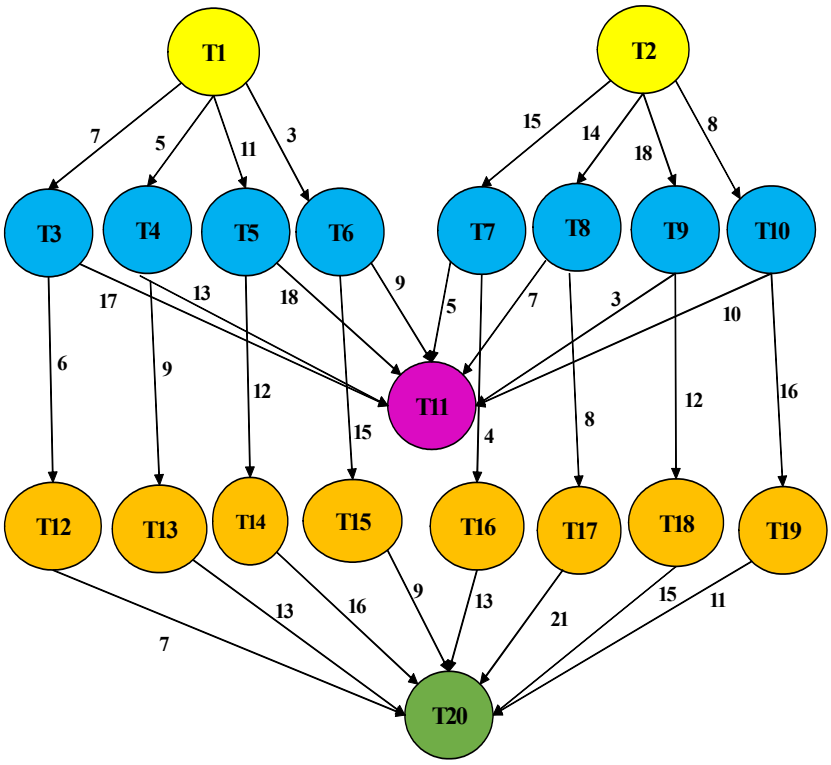


Figure 7. MONWS Workflow application.

Table 3. Data Transfer Time (DTT) between the tasks.

Tasks	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}
T_1	-	-	7	5	11	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-
T_2	-	-	-	-	-	-	15	14	18	8	-	-	-	-	-	-	-	-	-	-
T_3	-	-	-	-	-	-	-	-	-	-	17	6	-	-	-	-	-	-	-	-
T_4	-	-	-	-	-	-	-	-	-	-	13	-	9	-	-	-	-	-	-	-
T_5	-	-	-	-	-	-	-	-	-	-	18	-	-	12	-	-	-	-	-	-
T_6	-	-	-	-	-	-	-	-	-	-	9	-	-	-	15	-	-	-	-	-
T_7	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	4	-	-	-	-
T_8	-	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	8	-	-	-
T_9	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	12	-	-
T_{10}	-	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-	16	-
T_{11}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
T_{12}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7
T_{13}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13
T_{14}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16
T_{15}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9
T_{16}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13
T_{17}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	21
T_{18}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	15
T_{19}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	11
T_{20}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

ECT refers to the estimated computation time for each task and has a different execution time on different virtual machines, illustrated in Table 4. It is defined in the ECT Equation (2).

Table 4. Corresponding computational matrix.

		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}	T_{19}	T_{20}
CS1	vm_1	17	14	19	13	19	13	15	19	13	19	13	15	18	20	11	16	22	18	14	19
	vm_2	14	17	17	20	20	18	15	20	17	15	22	21	17	18	18	15	18	16	12	17
CS2	Vm_3	13	14	16	13	21	13	13	13	13	16	14	22	16	13	21	19	22	11	16	10
	Vm_4	22	16	12	14	15	18	14	18	19	13	12	14	14	16	17	12	23	17	19	24

As is illustrated in Figure 7, the workflow has two tasks in level 1 that are in the Ready list, RLi. They are designated as T_1 and T_2, and these tasks can be mapped to any of the four VMs on both cloud servers.

At level 1, T_ECT is represented by the below equation.

$$T_E^{ct} = \begin{array}{c} \\ T_1 \\ T_2 \end{array} \begin{array}{c} \begin{array}{cccc} vm_1 & vm_2 & vm_3 & vm_4 \end{array} \\ \begin{pmatrix} 17 & 14 & 13 & 22 \\ 14 & 17 & 14 & 16 \end{pmatrix} \end{array}$$

In the workflow example, tasks T_1 and T_2 have no predecessor tasks, which leads to no data transfer time between tasks T_1 and T_2. If DTTp,s is zero, then the threshold value Th(dTi) will be also zero, as seen in Equation (11). In this situation, the Th(dTi) value is fixed as 0.99 and is then compared with the maximum value in $N_T_E^{ct}$; wherein the maximum $N_T_E^{ct}$ is 1. If the threshold Th(dTi) is lesser than the maximum $N_T_E^{ct}$ i.e.,Th(dTi)< max $N_T_E^{ct}$, T_1 is then placed in the b_large. By calculating E^{st} and E^{ft} as per Equations (7) and (8), T_1 is then mapped into the VM3 in cloud server2.

$$N_T_E^{ct} = \begin{array}{c} \\ T_1 \\ T_2 \end{array} \begin{array}{c} \begin{array}{cccc} vm_1 & vm_2 & vm_3 & vm_4 \end{array} \\ \begin{pmatrix} 0.44444444 & 0.11111111 & 0 & 1 \\ 0.11111111 & 0.44444444 & 0.11111111 & 0.33333333 \end{pmatrix} \end{array}$$

The updated T_E^{ct} is then calculated as follows.

$$\text{Updated } T_E^{ct} \begin{pmatrix} 14 & 17 & 27 & 16 \end{pmatrix}$$

The corresponding Normalized $N_T_E^{ct}$ is calculated as follows:

$$N_T_E^{ct} = T2 \begin{pmatrix} 0 & 0.23076923 & 1 & 0.15384615 \end{pmatrix}$$

The maximum $N_T_E^{ct}$ for T_2 is then compared with the Th(dTi), which is fixed at 0.99. In this case as well, the Th(dTi) < maximum $N_T_E^{ct}$, therefore leading to T_2 also being placed in the b_large. At this juncture, b_large is now designated as b_large (T_1, T_2). By calculating Est and Eft as per Equations (7) and (8), T_2 is then mapped into VM1 in cloud server1.

In considered workflow level 2

In level 2, there are eight tasks: T_3, T_4, T_5, T_6, T_7, T_8, and T_9 which are placed in the Ready list, RLi. T_1 and T_2 are already slotted in VM3 and VM1, respectively. Therefore, the tasks T_1 and T_2 of level 1 are scheduled (mapped) on VM3 and VM1, respectively. The VM available times, $available_{vm}$, for VM1, VM2, VM3, and VM4 are now calculated to be 14, 0, 13, and 0 respectively. However, at this point, T_3 has a predecessor with a DTT (Data transfer time) of 7. Therefore, the T_E^{ct} matrix for T_3, T_4, T_5, T_6, T_7, T_8, and T_9 would then be updated as follows.

$$T_E^{ct} = \begin{array}{c} \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \\ T_{10} \end{array} \begin{array}{c} \begin{array}{cccc} vm_1 & vm_2 & vm_3 & vm_4 \end{array} \\ \begin{pmatrix} 39 & 37 & 29 & 25 \\ 31 & 38 & 26 & 25 \\ 43 & 44 & 34 & 28 \\ 29 & 34 & 26 & 31 \\ 29 & 29 & 42 & 43 \\ 33 & 34 & 45 & 50 \\ 27 & 31 & 41 & 47 \\ 33 & 29 & 38 & 35 \end{pmatrix} \end{array}$$

The T_E^{ct} normalization is performed in a stepwise manner according to the threshold Th(dTi). The tasks are then allotted to a suitable virtual machine. These are then moved into either b_large or b_small, similar to the process that was observed at level 1. Table 5 represents the mapping of tasks on the VMs where rows indicate the tasks and the columns indicate the EST, EFT, ACT, and VMID.

Table 5. Tasks allocation and execution description.

Tasks	EST	EFT	ACT	VMID
1	0	13	13	3
2	0	14	14	1
3	13	25	12	4
4	13	26	13	1
5	13	28	15	4
6	13	26	13	1
7	16	29	13	3
8	20	33	13	3
9	14	27	13	1
10	16	29	13	4
11	39	51	12	4
12	25	39	14	1
13	29	43	14	4
14	28	51	13	3
15	26	37	11	1
16	29	41	12	4
17	37	55	18	2
18	32	43	11	3
19	33	45	12	2
20	65	75	10	3

5. Simulation and Performance Evaluations

The current study focuses on evaluating a proposed algorithm by considering the metrics of makespan, average cloud utilization, and cost. The proposed algorithm's performance was compared against benchmark algorithms min-min [13], max-min [13], NMMWS [29], HEFT [41], and DLS [13]. All the algorithms were implemented in the cloudsim tool. The simulation environment was set up with two data centers and four virtual machines.

To MONWS is evaluated with four different workflows: Montage, Cybershake, LIGO, and Epigenomics.

In this paper, for the purpose of evaluating the results of the MONWS algorithm in terms of performance, simulated experiments are carefully designed and carried out for the considered workflows with a varying number of tasks classified as small (1–25 tasks), average (25–100 tasks), and large (100–1000 tasks). The Pegasus [42] workflow generator is used to generate four representative scientific workflows with the same structure (Montage, Cybershake, Epigenomics, and LIGO).

Makespan:

Makespan was considered the most important evaluation metric for determining the effectiveness of a scheduler. We have inputted small, average, and large tasks to the cloud environment to compare the makespan of the proposed algorithm with existing algorithms such as NMMWS, min-min, max-min, DLS, and HEFT. We observed that our proposed algorithm has a minimum makespan of 82.5 ms when compared with any existing algorithm shown in Figure 8.

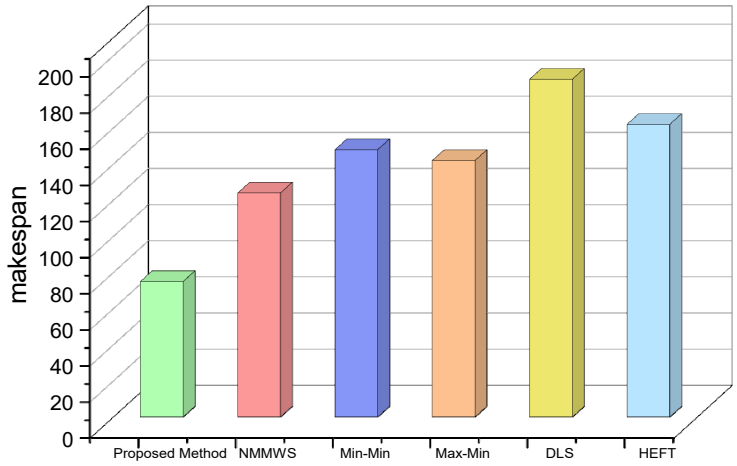


Figure 8. Makespan.

Average cloud utilization:

Average cloud utilization is another crucial metric in cloud computing from the perspective of the cloud provider. The average cloud utilization of the proposed algorithm in comparison to existing algorithms is shown in Figure 9. It can be noted that the proposed algorithm utilized 49% of cloud resources, while existing algorithms utilized 39%, 38%, 33%, and 37%, respectively. It can be inferred from the data that the proposed algorithm outperforms existing algorithms with respect to average cloud utilization.

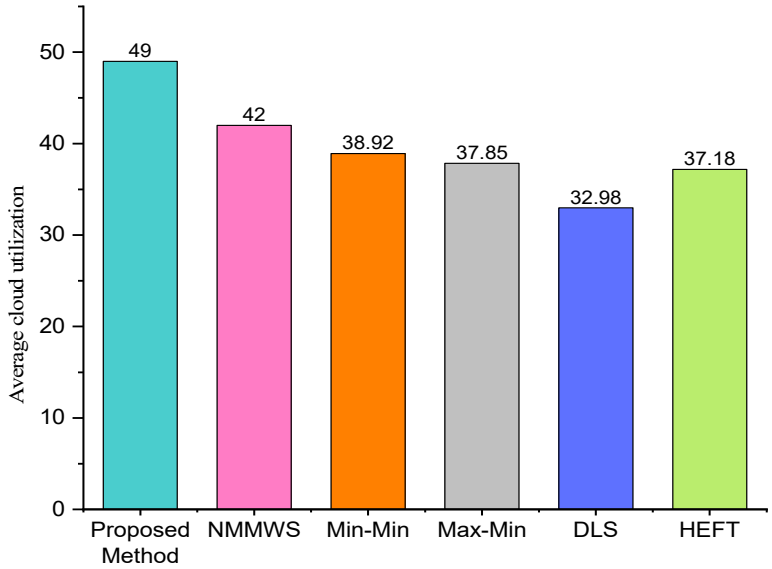


Figure 9. Average cloud utilization.

Cost:

Minimizing cost is another challenging parameter in cloud computing which affects both cloud consumers and providers alike. The cost incurred for the proposed algorithm in comparison to existing algorithms is shown in Figure 10. The costs were calculated using

Equation (10), and it is observed that the proposed algorithm incurred costs of 4613.94 ms, which is 8% better than NMMWS and 18% better than min-min.

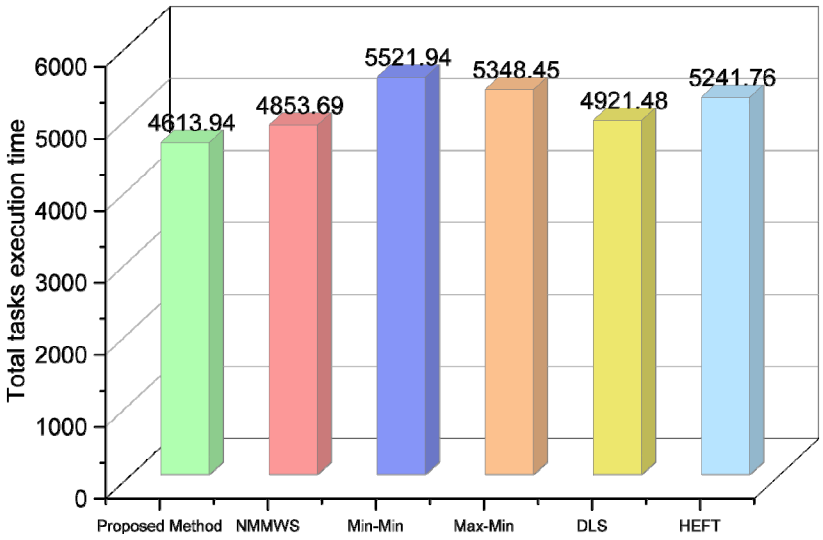


Figure 10. Cost.

VM Actual Working time:

In our proposed work, the actual working time of virtual machines for the given tasks is 130 ms for VM1, 100 ms for VM2, 234 ms for VM3, and 256 ms for VM4, as shown in Figure 11.

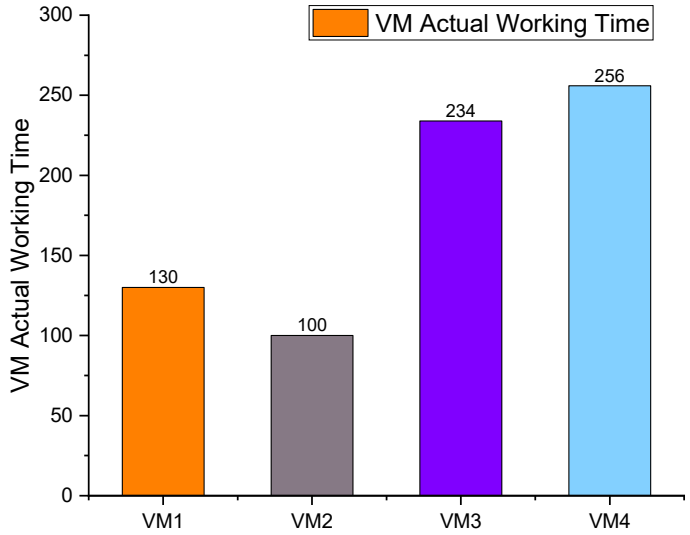


Figure 11. Actual working time of VMs.

Speedup:

The speedup value is the ratio of the minimum sequential execution time to the makespan. The sequential execution time is the total computation cost incurred when mapping all tasks sequentially to a single computing host. The proposed method outperforms the benchmark algorithms in terms of speed. The proposed method achieved a 20% speedup over the existing algorithm, as is shown in Figure 12. We have observed that speed is directly proportional to execution cost.

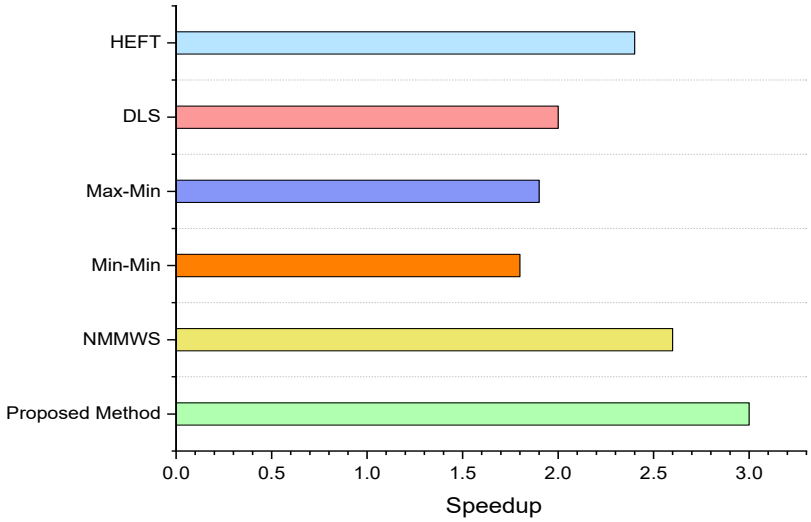


Figure 12. Speedup.

Load balance

Load balance is considered to be the most important metric for cloud service providers in order to maximize resource utilization. When comparing the load balance of the proposed method and the existing algorithms, the proposed method outperformed the benchmark algorithms by 7.23%, as is shown in Figure 13.

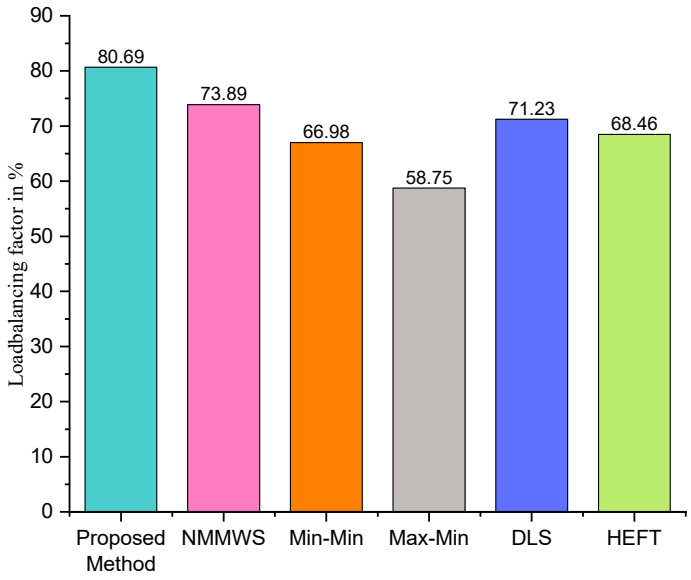


Figure 13. Load balance.

Evaluation of Success Rate:

The success rate, S_{rate}, is considered to be the ratio of the number of planning workflows and the total number of simulation runs. This rate is indicative of the performance of proposed and existing algorithms.

$$S_{rate} = \frac{\text{No.of Successful Planning Workflows}}{\text{Total no.of simulations}} \times 100 \quad (14)$$

From Figures 14–17, The success rate of the proposed algorithm's makespan is determined using Montage, Cybershake, Epigenomics, and LIGO, as well as 4, 8, and 12 iterations. We observed that, when the makespan was low, the success rate was consequently higher on the MONWS in comparison to HEFT, DLS, min-min, max-min, and NMMWS. As a result,

MONWS chooses the VMs with the earliest possible finish time (EFT) for optimally executing the workflow.

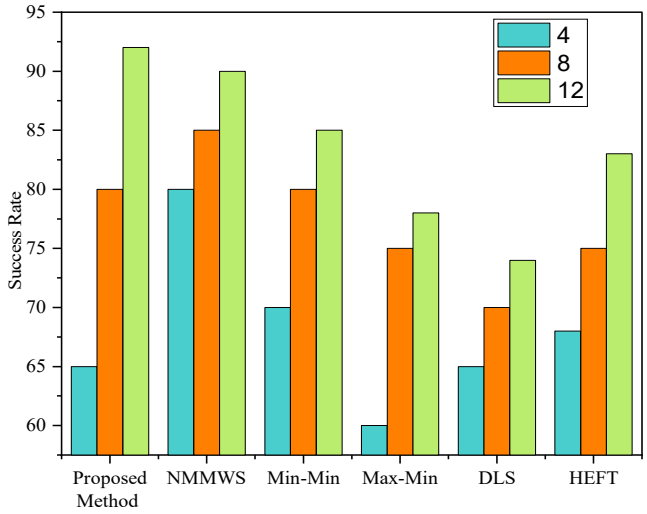


Figure 14. Success rate on the Montage workflow.

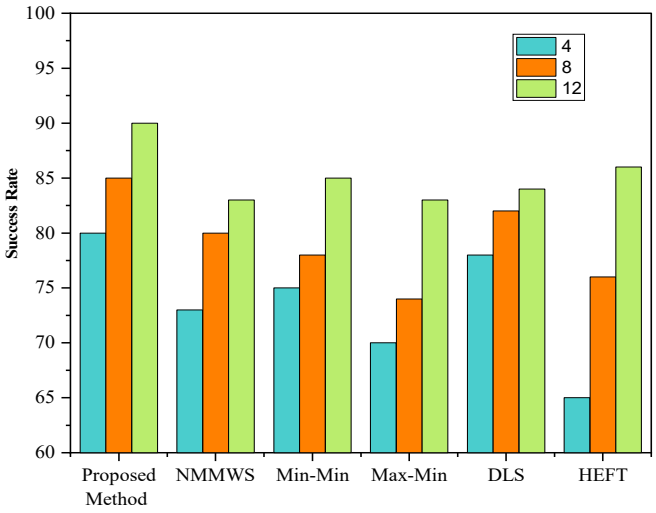


Figure 15. Success rate on the Cybershake workflow.

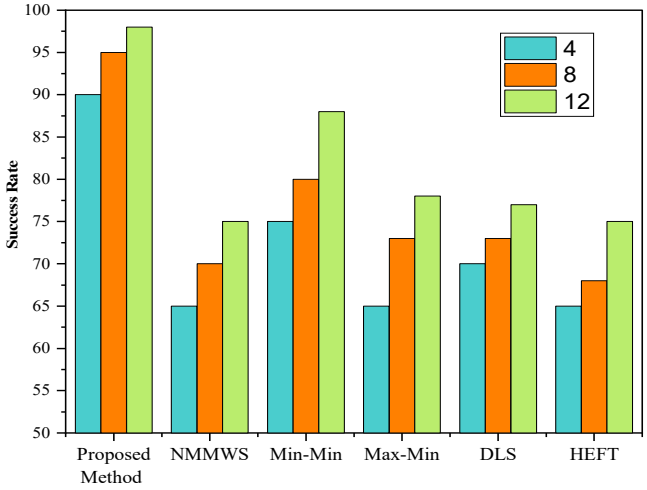


Figure 16. Success rate on the Epigenimics workflow.

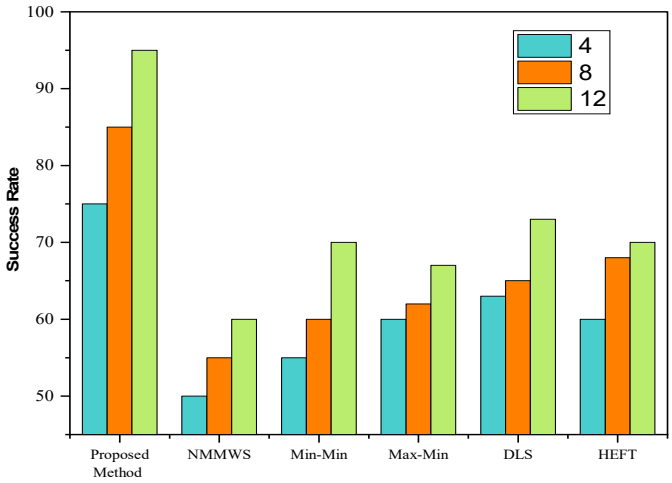


Figure 17. Success rate on the LIGO workflow.

6. Conclusions

This study proposed a novel MONWS algorithm by taking into consideration the dynamic priority of tasks based on min-max in order to minimize makespan and cost and maximize cloud utilization for optimum workflow scheduling. In the proposed algorithm in this study, the tasks were initially moved to the ready list, followed by the calculation of the estimated computation time and early finish time. Finally, the dynamic threshold values were compared in order to divide the tasks into groups (b_large and b_small) for scheduling the tasks on virtual machines. For implementation, the cloudsim simulation tool was utilized, and the results exhibited that the proposed algorithm outperformed the existing algorithms such as HEFT, DLS, min-min, max-min, and NMMWS with respect to real-world scientific workflow applications in terms of makespan, cost, cloud utilization, and load balancing. The MONWS proposed in this study achieved a 35% improvement in makespan, an 8% increase in maximum average cloud utilization, and a 4% decrease in cost.

However, the proposed algorithm needs improvement to handle cases of VM failure and checking energy consumption. The proposed algorithm also did not consider handling multiple workflows dynamically. These could potentially form the basis of future research.

Author Contributions: Conceptualization, V.R.P.; Methodology, G.R.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data of written code, all results of photo will be made available on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bardsiri, A.; Hashemi, S. A review of workflow scheduling in cloud computing environment. *Int. J. Comput. Sci. Manag. Res.* **2012**, *1*, 348–351.
2. Zhu, Z.; Zhang, G.; Li, M.; Liu, X. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 1344–1357. [CrossRef]
3. Juve, G.; Chervenak, A.; Deelman, E.; Bharathi, S.; Mehta, G.; Vahi, K. Characterizing and profiling scientific workflows. *Futur. Gener. Comput. Syst.* **2013**, *29*, 682–692. [CrossRef]
4. Rakrouki, M.A.; Alharbe, N. QoS-aware algorithm based on task flow scheduling in cloud computing environment. *Sensors* **2022**, *22*, 2632. [CrossRef]
5. Mangalampalli, S.; Keshari, S.; Vamsi, S.; Mangalampalli, K. Multi objective task scheduling in cloud computing using cat swarm optimization algorithm. *Arab. J. Sci. Eng.* **2021**. [CrossRef]

6. Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-Based task scheduling approach for cloud computing. *Sensors* **2022**, *22*, 920. [CrossRef]
7. Wu, Q.; Ishikawa, F.; Zhu, Q.; Xia, Y.; Wen, J. Deadline-Constrained cost optimization approaches for workflow scheduling in clouds. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3401–3412. [CrossRef]
8. Yu, J.; Buyya, R.; Ramamohanarao, K. *Workflow Scheduling Algorithms for Grid Computing*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 146. [CrossRef]
9. Zhang, H.; Shi, J.; Deng, B.; Jia, G.; Han, G.; Shu, L. MCTE: Minimizes task completion time and execution cost to optimize scheduling performance for smart grid cloud. *IEEE Access* **2019**, *7*, 134793–134803. [CrossRef]
10. Reddy, P.V.; Reddy, K.G. An analysis of a meta heuristic optimization algorithms for cloud computing. In Proceedings of the 2021 5th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 22–23 October 2021. [CrossRef]
11. Anwar, N.; Deng, H. A hybrid metaheuristic for multi-objective scientific workflow scheduling in a cloud environment. *Appl. Sci.* **2018**, *8*, 538. [CrossRef]
12. Malik, N.; Sardaraz, M.; Tahir, M.; Shah, B.; Ali, G.; Moreira, F. Energy-efficient load balancing algorithm for workflow scheduling in cloud data centers using queuing and thresholds. *Appl. Sci.* **2021**, *11*, 5849. [CrossRef]
13. Cao, H.; Jin, H.; Wu, X.; Wu, S.; Shi, X. DAGMap: Efficient and dependable scheduling of DAG workflow job in grid. *J. Supercomput.* **2010**, *51*, 201–223. [CrossRef]
14. Wu, Q.; Zhou, M.; Zhu, Q.; Xia, Y.; Wen, J. MOELS: Multiobjective evolutionary list scheduling for cloud workflows. *IEEE Trans. Autom. Sci. Eng.* **2020**, *17*, 166–176. [CrossRef]
15. Masdari, M.; ValiKardan, S.; Shahi, Z.; Azar, S.I. Towards workflow scheduling in cloud computing: A comprehensive analysis. *J. Netw. Comput. Appl* **2016**, *66*, 64–82. [CrossRef]
16. Durillo, J.J.; Fard, H.M.; Prodan, R. MOHEFT: A multi-objective list-based method for workflow scheduling. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, Taiwan, 3–6 December 2012; pp. 185–192. [CrossRef]
17. Habibi, M.; Jafari, N. Multi-objective task scheduling in cloud computing using an imperialist competitive algorithm. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 289–293. [CrossRef]
18. El Din Hassan Ali, H.G.; Ali, H.G.E.D.H.; Saroit, I.A.; Kotb, A.M. Grouped tasks scheduling algorithm based on QoS in cloud computing network. *Egypt. Inform. J.* **2017**, *18*, 11–19. [CrossRef]
19. Zhou, N.; Li, F.F.; Xu, K.; Qi, D. Concurrent workflow budget- and deadline-constrained scheduling in heterogeneous distributed environments. *Soft Comput.* **2018**, *22*, 7705–7718. [CrossRef]
20. Nasr, A.A.; El-Bahnasawy, N.A.; Attiya, G.; El-Sayed, A. Cost-effective algorithm for workflow scheduling in cloud computing under deadline constraint. *Arab. J. Sci. Eng.* **2019**, *44*, 3765–3780. [CrossRef]
21. Verma, A.; Kaushal, S. Cost-time efficient scheduling plan for executing workflows in the cloud. *J. Grid Comput.* **2015**, *13*, 495–506. [CrossRef]
22. Chakravarthi, K.K.; Shyamala, L.; Vaidehi, V. Budget aware scheduling algorithm for workflow applications in IaaS clouds. *Clust. Comput.* **2020**, *23*, 3405–3419. [CrossRef]
23. Zhou, X.; Zhang, G.; Sun, J.; Zhou, J.; Wei, T.; Hu, S. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Gener. Comput. Syst.* **2019**, *93*, 278–289. [CrossRef]
24. Rodriguez, M.A.; Buyya, R.A. Taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4041. [CrossRef]
25. Dubey, K.; Kumar, M.; Sharma, S.C. Modified HEFT algorithm for task scheduling in cloud environment. *Procedia Comput. Sci.* **2018**, *125*, 725–732. [CrossRef]
26. Arabnejad, V.; Bubendorfer, K.; Ng, B. Budget and deadline aware e-science workflow scheduling in clouds. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 29–44. [CrossRef]
27. Patra, S.S. Energy-Efficient task consolidation for cloud data center. *Int. J. Cloud Appl. Comput.* **2018**, *8*, 117–142. [CrossRef]
28. Mohammadzadeh, A.; Masdari, M.; Gharehchopogh, F.S. Energy and Cost-Aware Workflow Scheduling in Cloud Computing Data Centers Using a Multi-Objective Optimization Algorithm. *J. Netw. Syst. Manag.* **2021**, *29*, 1–34. [CrossRef]
29. Gupta, I.; Kumar, M.S.; Jana, P.K. Efficient workflow scheduling algorithm for cloud computing system: A Dynamic priority-based approach. *Arab. J. Sci. Eng.* **2018**, *43*, 7945–7960. [CrossRef]
30. Yakubu, I.Z.; Aliyu, M.; Musa, Z.A.; Matinja, Z.I.; Adamu, I.M. Enhancing cloud performance using task scheduling strategy based on resource ranking and resource partitioning. *Int. J. Inf. Technol.* **2021**, *13*, 759–766. [CrossRef]
31. Rawat, P.S.; Dimri, P.; Saroha, G.P. Virtual machine allocation to the task using an optimization method in cloud computing environment. *Int. J. Inf. Technol.* **2020**, *12*, 485–493. [CrossRef]
32. Kaur, A.; Kaur, B.; Singh, D. Meta-heuristic based framework for workflow load balancing in cloud environment. *Int. J. Inf. Technol.* **2019**, *11*, 119–125. [CrossRef]
33. Kamanga, C.T.; Bugingo, E.; Badibanga, S.N.; Mukendi, E.M. A multi-criteria decision making heuristic for workflow scheduling in cloud computing environment. *J. Supercomput.* **2022**, 1–22. [CrossRef]
34. Qin, S.; Pi, D.; Shao, Z. AILS: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment. *Expert. Syst. Appl.* **2022**, *198*, 116824. [CrossRef]

35. Belgacem, A.; Beghdad-Bey, K. Multi-objective workflow scheduling in cloud computing: Trade-off between makespan and cost. *Clust. Comput.* **2022**, *25*, 579–595. [CrossRef]
36. Zhang, L.; Wang, L.; Xiao, M.; Wen, Z.; Peng, C. EM_WOA: A budget-constrained energy consumption optimization approach for workflow scheduling in clouds. *Peer-Peer Netw. Appl.* **2022**, *15*, 973–987. [CrossRef]
37. Chakravarthi, K.K.; Neelakantan, P.; Shyamala, L.; Vaidehi, V. Reliable budget aware workflow scheduling strategy on multi-cloud environment. *Clust. Comput.* **2022**, *25*, 1189–1205. [CrossRef]
38. Gupta, I.; Kumar, M.S.; Jana, P.K. Compute-intensive workflow scheduling in multi-cloud environment. In Proceedings of the 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 21–24 September 2016; pp. 315–321. [CrossRef]
39. Panda, S.K.; Jana, P.K. Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment. *Inf. Syst. Front.* **2018**, *20*, 373–399. [CrossRef]
40. Panda, S.K.; Jana, P.K. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *J. Supercomput.* **2015**, *71*, 1505–1533. [CrossRef]
41. Topcuoglu, H.; Hariri, S. Society IC. In *Performance-Effective and Low-Complexity*; IEEE: New York, NY, USA, 2002; Volume 13, pp. 260–274.
42. Available online: https://confluence.pegasus.isi.edu/display/pegasus (accessed on 10 October 2022).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.