HEFT-IPF: A Three-Phase Scheduling Algorithm for Heterogeneous Multi-Cloud Environment

Sanjaya Kumar Panda, *Senior Member*, *IEEE*
Department of Computer Science & Engineering
National Institute of Technology (NIT) Warangal
Warangal - 506004, Telangana, India
sanjaya@nitw.ac.in

Amey Jadhav
Department of Computer Science & Engineering
National Institute of Technology (NIT) Warangal
Warangal - 506004, Telangana, India
ameyjadhav036@gmail.com

Abstract*—Workflow scheduling (WFS) in a heterogeneous multi-cloud (HMC) environment is a critical problem, aiming to minimize overall completion time (i.e., makespan) and maximize resource utilization. Numerous heuristic and metaheuristic algorithms have been developed to address the problem of WFS. One well-known and benchmark algorithm is called heterogeneous earliest finish time (HEFT). This algorithm allocates the precedence-constrained workflow tasks to the clouds by calculating the task prioritization, followed by the cloud selection. However, it does not consider the task characterization phases, namely initialization, processing, and finalization (IPF), which leads to poor makespan and resource utilization. Therefore, this paper introduces a WFS algorithm called HEFT-IPF to enhance the HEFT algorithm's performance by considering task characterization. HEFT-IPF algorithm overlaps the execution of tasks by executing their initialization and finalization phases while strictly preserving their precedence constraints. The HEFT-IPF algorithm performance is compared with that of the HEFT algorithm by considering various scientific workflows, namely epigenomics, laser interferometer gravitational-wave observatory (LIGO), cybershake, sRNA identification protocol using high-throughput technology (SIPHT), and montage. Two performance measures, makespan and resource utilization, are used to compare with HEFT and HEFT-IPF algorithms. Simulation results show that the HEFT-IPF algorithm outperforms the HEFT algorithm, achieving a 28.36% average reduction in makespan and a 23.33% average improvement in resource utilization.

***Index Terms*—Cloud Computing, Workflow Scheduling, Heterogeneous Earliest Finish Time, Initialization, Processing, Finalization, Scientific Workflow, Makespan.**

I. Introduction

Cloud computing delivers a wide range of services over the Internet. The services include computing, network, storage, database, software, and analytics [1], [2]. It provides an infrastructure to rent these services on-demand whenever and wherever required [3], [4]. It utilizes the resources efficiently and ensures service availability and reliability, reducing the risk of failures [5]. As a result, there is a surge in the wide range of users, particularly among small to medium-scale enterprises [6]. Therefore, efficient scheduling algorithms are essential to manage the resources without scaling up the existing infrastructure [7]–[12]. However, these algorithms may be designed to handle independent or dependent tasks structured as a workflow [13], [14]. On the other hand, one cloud may collaborate with another cloud to meet the dynamic demands. In such cases, WFS is quite challenging as these clouds are heterogeneous and provide services in a federated manner.

WFS aims to minimize makespan by executing the precedence-constrained tasks and to maximize the utilization of cloud resources [7]–[11], [15], [16]. These tasks differ in their computational needs and execution durations, which diverge from cloud to cloud due to the heterogeneity of available resources [5], [17], [18]. Researchers develop many heuristic and metaheuristic algorithms to handle the WFS problem in the HMC environment. Some of the existing algorithms include HEFT [8], critical path on a processor (CPOP) [7], cluster combining algorithm (CCA) [9], granularity score scheduling (GSS) [10], and performance effective task scheduling (PETS) [11]. However, HEFT is a well-known and benchmark algorithm that assigns tasks to the clouds in a two-step process: task prioritization and cloud selection [8]. The first step prioritizes the tasks by evaluating their communication and computation costs to determine the execution order. The later step assigns each task to a cloud in that order, which gives the earliest finish time to improve scheduling efficiency. However, this algorithm considers a task as a whole without dividing it into IPF, which characterizes the task. As a result, successor tasks must wait to complete their predecessor tasks, leading to poor makespan and resource utilization. However, the finalization of a task can overlap with the initialization of the successor task(s). This phenomenon inspires us to introduce a three-phase WFS algorithm.

This paper addresses the problem of assigning a workflow that consists of n dependent tasks to a set of m clouds to minimize the makespan and maximize resource utilization. For this, a three-phase WFS algorithm called HEFT-IPF is introduced, which enhances the performance of the existing algorithm (i.e., HEFT) by considering task characterization phases, IPF. The initialization phase includes data loading, memory configuration, and resource preparation. The computations are carried out in the processing phase. The finalization phase focuses on storing results and releasing resources. HEFT-IPF algorithm overlaps the execution of tasks by overlapping their initialization and finalization phases without violating their precedence constraints. It reduces idle time and enhances parallel execution, improving makespan and resource utilization.

We compare the existing algorithm, HEFT, and the proposed algorithm, HEFT-IPF, using various simulation runs by taking scientific workflows, namely epigenomics [19], [20], LIGO [21], cybershake [22], SIPHT [23] and montage [24]. Each algorithm is assessed by the makespan and resource utilization. The results show that the HEFT-IPF performs comparatively better than the HEFT across all workflows, with improvements in makespan ranging from 2% to 59% and resource utilization from 15% to 48%. However, the average improvement of the HEFT-IPF algorithm is 28.36% in makespan and 23.33% in resource utilization, showing the significance of the proposed algorithm.

A. Motivational Example

The assembly language instructions are used to illustrate how the HEFT algorithm works and the need for the HEFT-IPF algorithm, as shown in Fig. 1 and Fig. 2. For this, we consider two dependent tasks, namely T_1 and T_2, and two clouds, namely C_1 and C_2. Note that task T_2 is dependent on task T_1. Assume that task T_1 is allotted to cloud C_1 and task T_2 is allotted to cloud C_2 for execution. However, cloud C_2 can execute task T_2 only after the completion of task T_1 by cloud C_1. In the HEFT algorithm, cloud C_1 executes the task T_1 as follows. Register B is loaded with data 03H, and register A is loaded with a value from the memory location 2000H. Then, it adds the values of the registers and stores the result in the memory location 3000H. It completes the execution of task T_1. Note that cloud C_2 is completely idle until task T_1 is completed. Now, cloud C_2 executes the task T_2 by loading the results of T_1 from the memory location 3000H into the accumulator. Additionally, it loads the data 05H to register B. Then, it subtracts the values of the registers and stores the result in the memory location 4000H, as seen in Fig. 1. This completes the execution of task T_2. However, loading the data 05H into register B by cloud C_2 is entirely independent of any operations performed by cloud C_1. Therefore, the HEFT-IPF algorithm performs this independent operation or initialization before completing task T_1 by cloud C_1, especially during the finalization phase of task T_1, as seen in Fig. 2. This example shows that the HEFT-IPF algorithm improves parallel execution and is a better choice than the HEFT algorithm.

```
Cloud C1: Executing Task T1
MVI B, 03H ; Load data
LDA 2000H  ; Load data
ADD B      ; Perform computation
STA 3000H  ; Store result

Cloud C2: Executing Task T2
LDA 3000H  ; Wait for Task T1 to finalize
MVI B, 05H ; Load data
SUB B      ; Perform computation
STA 4000H  ; Store result
```

Fig. 1: An illustration to demonstrate the working of HEFT.

The other sections of the paper are arranged as follows. Section II discusses related work. Section III shows the problem statement. Section IV describes the HEFT-IPF and its illustration. Section V discusses the simulation setup, results, and discussion. Section VI presents the paper's conclusion and outlines directions for future work.

```
Cloud C1: Executing Task T1
MVI B, 03H ; Load data (Initialization)
LDA 2000H  ; Load data (Initialization)
ADD B      ; Perform computation (Processing)

Cloud C2: Executing Task T2
MVI B, 05H ; Load data (Initialization)

Cloud C1: Executing Task T1
STA 3000H  ; Store result (Finalization)

Cloud C2: Executing Task T2
LDA 3000H  ; Load data (Initialization)
SUB B      ; Perform computation (Processing)
STA 4000H  ; Store result (Finalization)
```

Fig. 2: An example to show the working of HEFT-IPF.

II. Related Work

Many WFS algorithms have been developed to execute the precedence-constrained tasks in a heterogeneous environment. Some of these algorithms are briefly discussed here. Topcuoglu et al. [8] have introduced the HEFT algorithm to schedule dependent tasks in different processors to minimize makespan. The HEFT algorithm does not support task overlapping, indicating that a successor task cannot initiate its initialization while the previous task is finalized. Samadi et al. [25] have proposed enhancing the HEFT algorithm by using matching game theory to distribute the tasks among the virtual machines. However, they have shown the results in two scientific workflows, cybershake and montage. Dubey et al. [26] have improved the distribution of tasks to reduce the makespan by introducing the modified HEFT algorithm. However, eleven tasks and three resources without scientific workflows show the algorithm's performance.

Sun et al. [27] have proposed the HEFT-dynamic algorithm by focusing on virtual machine selection and allocation. However, their algorithm has only been tested in three scientific workflows: cybershake, SIPHT, and montage. Hai et al. [28] have proposed an enhanced version of the HEFT algorithm by addressing its limitations regarding selecting the first available time slot and average computation cost. However, they have highlighted the trade-off between cost and performance and suggested applying nature-inspired optimization algorithms. Gupta et al. [29] have proposed an improved version of the HEFT algorithm by applying different ranking methodologies, namely average, maximum, and minimum computation cost, and determining idle slots for task scheduling. However, they have not considered the task characterization phases. Chopra et al. [30] have proposed a HEFT-based hybrid scheduling algorithm by introducing the deadline constraints and targeting the hybrid clouds. However, they have not used the complete workflow application in their performance analysis. The above

algorithms do not consider the task characterization phases. Therefore, the proposed algorithm is entirely different from those and compared with the baseline algorithm, HEFT.

III. Problem Statement

Consider a directed acyclic graph (DAG), $G = (T, E)$, where $T = \{T_1, T_2, T_3, \ldots, T_n\}$ denotes the set of n tasks, E represents the set of o edges. Let $C = \{C_1, C_2, C_3, \ldots, C_m\}$ denotes set of m heterogeneous clouds. Each edge $CO_{ii'}$ represents the communication time between task T_i, $1 \leq i \leq n$ and task $T_{i'}$, $1 \leq i' \leq n$, $i \neq i'$. The expected time to complete (ETC) of a task T_i, $1 \leq i \leq n$, on a cloud C_j, $1 \leq j \leq m$, is represented as ETC_{ij}. Note that $ETC_{ij} \neq ETC_{ij'}$, $1 \leq i \leq n$, $1 \leq j, j' \leq m$, $j \neq j'$. Given an ETC matrix, the WFS problem involves assigning n tasks to m clouds while respecting task precedence constraints, with the objectives of (1) minimizing makespan, and (2) maximizing resource utilization.

IV. Proposed Algorithm

The HEFT-IPF algorithm is developed to address the WFS problem in an HMC environment, with the dual objective of reducing makespan and increasing resource utilization. The algorithm characterizes the tasks by dividing their execution into three phases: initialization, processing, and finalization. It allows a dependent task to start the initialization phase in a cloud before its predecessor completes its finalization phase on another cloud. For instance, assume that the ETC of a task T_i and a successor task $T_{i'}$ on a cloud C_j is ETC_{ij} and $ETC_{i'j}$. This value represents the total time spent across three distinct phases. As a result, the ETC_{ij} or $ETC_{i'j}$ is divided using three values based on τ_1, τ_2, and τ_3, respectively. Specifically, the initialization (I), processing (P) and finalization (F) phases take $\tau_1 \times ETC_{ij}$, $\tau_2 \times ETC_{ij}$, and $\tau_3 \times ETC_{ij}$ time units for task T_i and $\tau_1 \times ETC_{i'j}$, $\tau_2 \times ETC_{i'j}$, and $\tau_3 \times ETC_{i'j}$ time units for task T_i', respectively. Note that $\tau_1 + \tau_2 + \tau_3 = 1$ and $0 < \tau_1, \tau_2, \tau_3 < 1$. Mathematically,

$$I(T_i, C_j) = \tau_1 \times ETC_{ij}, P(T_i, C_j) = \tau_2 \times ETC_{ij}, F(T_i, C_j) = \tau_3 \times ETC_{ij} \quad (1)$$

$$I(T_{i'}, C_j) = \tau_1 \times ETC_{i'j}, P(T_{i'}, C_j) = \tau_2 \times ETC_{i'j}, F(T_{i'}, C_j) = \tau_3 \times ETC_{i'j} \quad (2)$$

HEFT-IPF algorithm allows a dependent task $T_{i'}$ to start the initialization phase in a cloud (i.e., $\tau_1 \times ETC_{i'j}$) before its predecessor task T_i completes its finalization phase on another cloud (i.e., $\tau_3 \times ETC_{ij}$). The processing phase of the dependent task $T_{i'}$ can commence after the finalization of task T_i. HEFT-IPF algorithm performs the mapping process between the tasks and clouds by following matching and scheduling. In the matching, tasks are prioritized to determine their execution order according to computation and communication times. Let $priority_u(T_i)$ represent the upward priority of task T_i, which is calculated by adding the average ETC across all the clouds (i.e., $\overline{ETC_i}$) and the maximum sum of the communication time between task T_i and successor task(s), and the upward priority of the successor task(s). Let $T_{i'}$ represent the only successor of task T_i, and $succ(T_i)$ denote the set of successors of T_i. The upward priority is calculated as follows.

$$priority_u(T_i) = \overline{ETC_i} + \max_{i' \in succ(T_i)} (CO_{ii'} + priority_u(T_{i'})) \quad (3)$$

The upward priority helps to determine the critical tasks and ensures that those tasks are scheduled earlier. It plays a significant role in the HEFT-IPF algorithm by determining the task order to improve workflow efficiency. The upward priority of the exit task T_{exit} is determined as follows.

$$priority_u(T_{exit}) = \overline{ETC_{exit}} \quad (4)$$

Once the upward priorities of the tasks are calculated, they are sorted according to their priority values. Tasks are scheduled to the most suitable cloud based on the minimum start time (MST) and minimum completion time (MCT). Note that MST and MCT help in selecting the most appropriate cloud for a task while considering resource availability and task dependencies. The MST of task T_i on cloud C_j (i.e., $MST(T_i, C_j)$) is determined by considering the maximum of the ready time of cloud C_j (i.e., $ready(C_j)$) and the maximum of the MCT of all the predecessors of task T_i (i.e., $MCT(pred(T_i))$) if task T_i and its $pred(T_i)$ are assigned to the cloud in which one of the predecessors is assigned on the cloud C_j. Otherwise, the $MST(T_i, C_j)$ is calculated by taking the maximum of the $ready(C_j)$ and the maximum end time of the processing of all the predecessors of task T_i (i.e., $P_{end}(pred(T_i))$). Mathematically,

$$MST(T_i, C_j) = \begin{cases} max(ready(C_j), max(MCT(pred(T_i)))), \\ \text{if one of the predecessors is assigned on the cloud } C_j \\ max(ready(C_j), max(P_{end}(pred(T_i)))), \\ \text{Otherwise} \end{cases} \quad (5)$$

The MST of the entry task T_{entry} is calculated as zero. Once the MST is calculated, the MCT is calculated by adding the MST and ETC. For instance, the MCT of task T_i on cloud C_j is calculated as follows.

$$MCT(T_i, C_j) = MST(T_i, C_j) + ETC_{ij} \quad (6)$$

Once the HEFT-IPF algorithm has determined the upward priority, MST, and MCT, it allocates the tasks to the clouds as follows. The MST of the initialization phase of task T_i starts only when cloud C_j is ready, and the processing phase of its predecessor task (i.e., $pred(T_i)$) is completed. Mathematically,

$$MST(I(T_i), C_j) = max(ready(C_j), P_{end}(pred(T_i))) \quad (7)$$

Note that the $max(P_{end}(pred(T_i)))$ is not used in Eq. (7) as the cloud C_j is already determined using Eq. (6). The MCT of the initialization phase of task T_i on cloud C_j is calculated as the sum of $MST(I(T_i), C_j)$ and $I(T_i, C_j)$, where $I(T_i, C_j)$ be the duration of the initialization phase of task T_i on cloud C_j. Mathematically,

$$MCT(I(T_i), C_j) = MST(I(T_i), C_j) + I(T_i, C_j) \quad (8)$$

The MST of task T_i processing phase on cloud C_j starts only after completing its initialization phase and the finalization phase of all the predecessor tasks of task T_i. Mathematically,

$$MST(P(T_i), C_j) = max(MCT(I(T_i), C_j), MCT(pred(T_i))))) \quad (9)$$

The MCT of the processing phase of task T_i on cloud C_j is calculated as the sum of $MST(P(T_i), C_j)$ and $P(T_i, C_j)$, where $P(T_i, C_j)$ be the duration of the processing phase of task T_i on cloud C_j. Mathematically,

$$MCT(P(T_i), C_j) = MST(P(T_i), C_j) + P(T_i, C_j) \quad (10)$$

The MCT of the finalization phase of task T_i on cloud C_j (i.e., $MCT(T_i, C_j)$ or $MCT(F(T_i), C_j)$) is calculated as the sum of $MCT(P(T_i), C_j)$ and $F(T_i, C_j)$, where $F(T_i, C_j)$ be the duration of the finalization phase of task T_i on cloud C_j. Mathematically,

$$MCT(T_i, C_j) = MCT(P(T_i), C_j) + F(T_i, C_j) \quad (11)$$

The step-by-step process of the HEFT-IPF algorithm is presented in Algorithm 1.

Algorithm 1 HEFT-IPF: A Three-Phase Scheduling Algorithm

Inputs: $G(V, E)$, ETC, and CO
Outputs: Task ordering, makespan, and resource utilization

1: **for** each task T_i in reverse topological order **do**
2: Compute the upward priority of task T_i using Eq. (3)
3: **end for**
4: Sort the tasks in non-increasing order of priority and place them in queue Q
5: **while** there are tasks that are unscheduled in queue Q **do**
6: Select task T_i with highest-priority from queue Q
7: **for** each cloud C_j **do**
8: Calculate $MST(T_i, C_j)$ using Eq. (5)
9: Calculate $MCT(T_i, C_j)$ using Eq. (6)
10: **end for**
11: Allocate task T_i to cloud C_j with the minimum MCT
12: Remove task T_i from queue Q
13: Divide ETC_{ij} into $I(T_i, C_j)$, $P(T_i, C_j)$, and $F(T_i, C_j)$ using Eq. (1)
14: Calculate the MST and MCT using Eq. (7) to Eq. (11)
15: Update the $ready(C_j)$
16: Execute the task T_i
17: **end while**

A. Illustration

Consider a DAG with ten tasks (i.e., T_1 to T_{10}) as shown in Fig. 3 and three clouds (i.e., C_1 to C_3). The computation time for each task on the different clouds is presented in the second to fourth columns of Table I. The communication times between tasks are indicated alongside the edges in Fig. 3.

HEFT-IPF algorithm uses Eq. (3) to compute the priority of each task starting from the exit task, where the priority of the exit task is initialized using Eq. (4). The priority of task T_{10} is determined as 19 (i.e., $\overline{ETC_i}$) and the priority of task T_9 is determined as 46.67 (i.e., 17.67 + max(10.00 + 19.00)). Table I presents the task priority calculation in the last column. The tasks are then sorted in decreasing order based on their priority values, i.e., T_1, T_3, T_2, T_4, T_6, T_5, T_7, T_8, T_9, and T_{10}. Next, the MST and MCT are calculated using Eq. (5) and Eq. (6) to assign tasks to the clouds. For instance, the MST of task T_1 on three clouds is 0, and the MCT of task T_1 on three clouds is 13 (i.e., 0 + 13), 12, and 15, respectively. Since cloud C_2 gives MCT as 12, task T_1 is allocated to cloud C_2. Here, the execution time $ETC_{12} = 12$ is divided into three phases: initialization $I(T_1, C_2) = 1.2$, processing $P(T_1, C_2) = 9.6$, and finalization $F(T_1, C_2) = 1.2$. The minimum start and completion times for the initialization phase are $MST(I(T_1), C_2)$ = 0 and $MCT(I(T_1), C_2)$ = 1.2, respectively. The processing phase starts at $MST(P(T_1), C_2)$ = 1.2 and completes at $MCT(P(T_1), C_2)$ = 10.8. The MCT of task T_1 on cloud C_2 is $MCT(T_1, C_2)$ = 12.

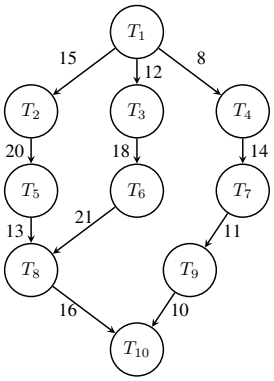

Fig. 3: A DAG with ten tasks.

TABLE I: Task computation times across different clouds and the calculation of task priorities

Task	C_1	C_2	C_3	$\overline{ETC_i}$	$succ(T_i)$	$priority_u(T_i)$
T_1	13	12	15	$\frac{13+12+15}{3}$ = 13.33	2, 3, 4	148.66
T_2	16	17	14	15.67	5	115.00
T_3	12	18	19	16.33	6	123.33
T_4	17	15	14	15.33	7	102.33
T_5	19	16	12	15.67	8	079.33
T_6	14	21	17	17.33	8	089.00
T_7	18	13	15	15.33	9	073.00
T_8	21	14	12	15.67	10	050.67
T_9	15	20	18	17.67	10	046.67
T_{10}	22	19	16	19.00	–	019.00

The next highest-priority task is T_3, which is scheduled as follows. The MST of task T_3 on clouds C_1, C_2, and C_3 is 10.8, 12, and 10.8, respectively. The corresponding MCTs are 22.8 (i.e., 10.8 + 12.0), 30, and 29.8. Since cloud C_1 gives the minimum completion time of 22.8, task T_3 is allotted to cloud C_1. Here, the execution time ETC_{31} = 12 is divided into three phases: initialization $I(T_3, C_1) = 1.2$, processing $P(T_3, C_1) = 9.6$, and finalization $F(T_3, C_1) = 1.2$. The minimum start and completion times for the initialization phase are $MST(I(T_3), C_1)$ = 10.8 and $MCT(I(T_3), C_1)$ = 12, respectively. The processing phase starts at $MST(P(T_3), C_1)$ = 12 and completes at $MCT(P(T_3), C_1)$ = 21.6. The MCT of task T_3 on cloud C_1 is $MCT(T_3, C_1)$ = 22.8. Similarly, other tasks are allotted to the respective clouds, as shown in Fig.

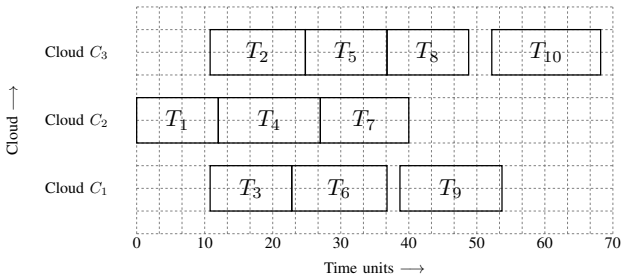

Fig. 4: Gantt chart of the HEFT-IPF algorithm.

4. The makespan of the HEFT-IPF and HEFT algorithms is 68.20 and 99 time units for executing ten tasks on three clouds. On the other hand, the resource utilization of the HEFT-IPF and HEFT algorithms is 79.13% and 53.87%. It shows the superior performance of the HEFT-IPF algorithm compared to the HEFT algorithm.

V. Simulation Setup, Results and Discussion

The simulations were conducted on an ASUS Vivobook Pro 15 system configured with Windows 11 operating system (OS), an advanced micro devices (AMD) Ryzen 5 5600H processor, integrated Radeon graphics, and 16 gigabytes (GB) of random access memory (RAM). The HEFT and HEFT-IPF algorithms were implemented in Python and executed using Google Colab. We used five standard scientific workflows, namely epigenomics, LIGO, cybershake, SIPHT, and montage, from the pegasus workflow gallery [13], [14], as shown in Fig. 5 to compare the HEFT and HEFT-IPF algorithms. These workflows are briefly discussed as follows. Epigenomics [20] workflow was developed by the University of Southern California (USC) epigenome center to automate genome sequencing tasks. It processes deoxyribonucleic acid (DNA) sequence data in parallel, converts file formats, filters out noisy and contaminating sequences, maps sequences to a reference genome, and generates a global map of sequence densities. LIGO [20], [21] workflow analyzes data to detect gravitational waves from cosmic events like the merging of black holes or neutron stars. It divides time-frequency data into smaller blocks and applies matched filtering to identify potential signals. Cybershake [20], [22] workflow was developed by the Southern California earthquake center and is used to study earthquake risks in different regions. It simulates fault movements, models ground shaking, generates synthetic seismograms, and estimates the probability and effects of earthquakes. SIPHT [20], [23] is a bioinformatics workflow that identifies small untranslated RNAs (sRNAs) using the national center for biotechnology information (NCBI) database. It processes genomic sequences, predicts transcriptional terminators, and annotates potential sRNA genes. Montage [20], [24] is an open-source application developed by the national aeronautics and space administration (NASA) or the infrared processing and analysis center (IPAC). It combines images in a specific format, called a flexible image transport system (FITS), to create custom sky mosaics. It adjusts the image scale and orientation of the input images, corrects background emission levels, and merges the results into a final mosaic. It can be executed in a grid computing platform like TeraGrid. Each workflow contains different levels of complexity and task dependencies, which helped us to measure and validate the existing and proposed algorithms' performance regarding the makespan and resource utilization metrics. The maximum completion time across all clouds is called the makespan (MS). Mathematically,

$$MS = max(MS(C_1), MS(C_2), \ldots, MS(C_m)) \tag{12}$$

where C_j, $1 \leq j \leq m$, represents the MS of cloud C_j in the HMC environment. Resource utilization (RU) is the ratio between the average makespan over all the clouds and the overall makespan. Mathematically,

$$RU = \frac{\frac{\sum_{j=1}^{m} MS(C_i)}{m}}{MS} \tag{13}$$

We took two datasets, namely small and large. In the small dataset, the number of tasks is 20 for epigenomics, cybershake, and montage. These workflows' computation and communication times are considered as [5 $\sim$ 21] and [9 $\sim$ 24], respectively. The number of tasks in LIGO and SIPHT is 40 and 30. LIGO's computation and communication times are [9 $\sim$ 21] and [9 $\sim$ 27]. SIPHT's computation and communication times are [5 $\sim$ 21] and [9 $\sim$ 24]. Fig. 6 and Fig. 7 show the comparison of MS and RU between the existing algorithm, HEFT, and the proposed algorithm, HEFT-IPF, in the small dataset. The HEFT-IPF algorithm improves the MS compared to the HEFT algorithm as follows. In the epigenomics workflow, the MS of the HEFT algorithm is 127 time units, while the MS of the HEFT-IPF algorithm reduces it to 94.30 time units. On the other hand, the RU of the HEFT-IPF algorithm achieves 86.82%, while the RU of the HEFT algorithm achieves 58.53%. We found similar improvements in the other scientific workflows, LIGO, cybershake, SIPHT and montage. The results clearly show that the HEFT-IPF algorithm minimizes the MS and utilizes the resources efficiently than the HEFT algorithm in all the workflows.

In the large dataset, the number of tasks, computation, and communication times for the scientific workflows are considered as per the pegasus workflow gallery [13]. Fig. 8 shows a makespan comparison between HEFT and HEFT-IPF algorithms. In the epigenomics workflow with 529 tasks, the HEFT-IPF algorithm achieves an MS of 3796.13 time units, compared to the MS of 3855.95 time units in the HEFT algorithm, showing its superiority. Similar improvements are observed in other workflows. Fig. 9 shows a RU comparison between the HEFT and HEFT-IPF algorithms. In the epigenomics workflow, the HEFT-IPF algorithm achieves an RU of 99.32%, compared to the RU of 98.36% in the HEFT algorithm. Similar improvements are observed in the remaining workflows. The simulation results in terms of MS and RU show that the HEFT-IPF algorithm effectively improves the HEFT algorithm across all the scientific workflows. Specifically, we found that the HEFT-IPF algorithm achieves a varying improvement of 2% to 59% in MS and 15% to 48% in

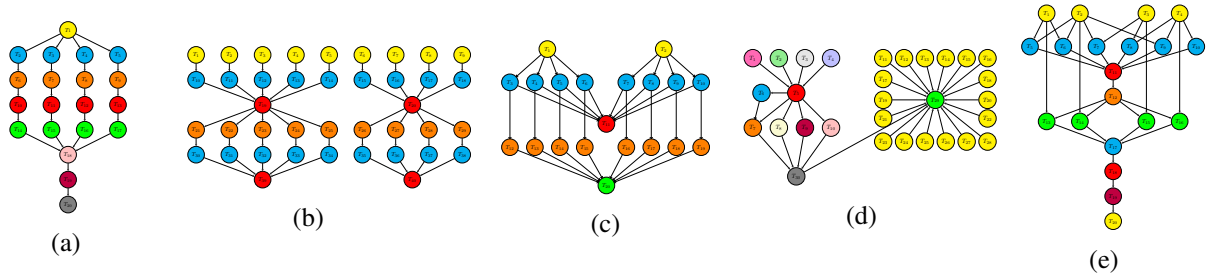

Fig. 5: Scientific workflow diagrams: (a) Epigenomics, (b) LIGO, (c) CyberShake, (d) SIPHT, and (e) Montage.

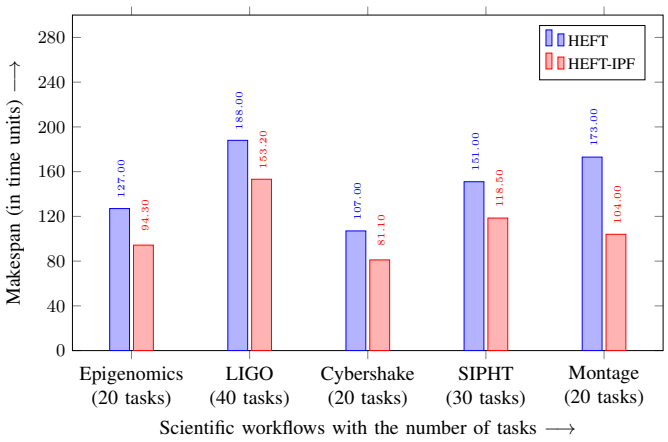

Fig. 6: Comparison of makespan across scientific workflows using HEFT and HEFT-IPF algorithms in the small dataset.

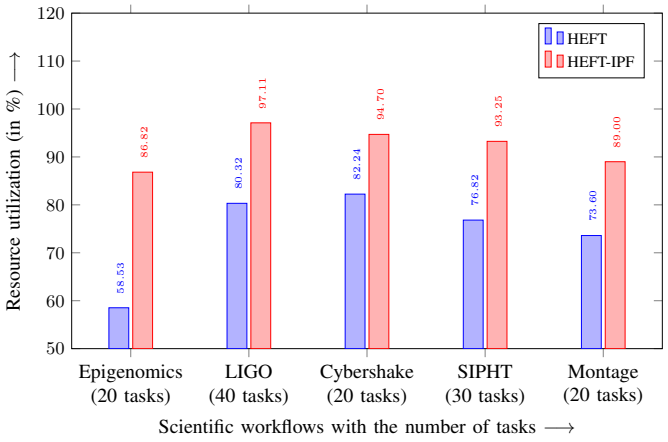

Fig. 7: Comparison of resource utilization across scientific workflows using HEFT and HEFT-IPF algorithms in the small dataset.

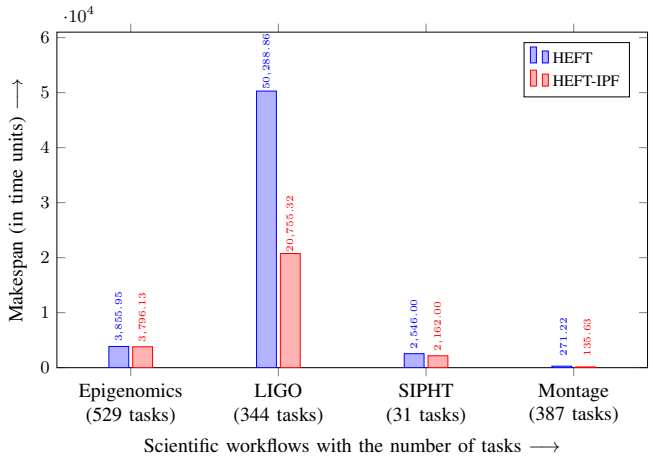

Fig. 8: Comparison of makespan across scientific workflows using HEFT and HEFT-IPF algorithms in the large dataset.

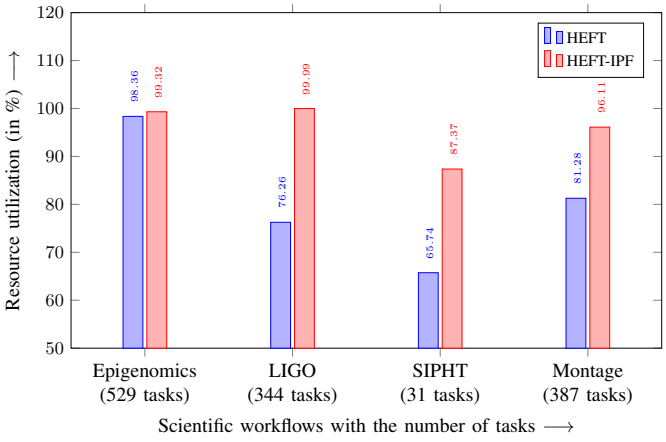

Fig. 9: Comparison of resource utilization across scientific workflows using HEFT and HEFT-IPF algorithms in the large dataset.

RU, compared to the HEFT algorithm. However, the average improvement of the HEFT-IPF algorithm is 28.36% in MS and 23.33% in RU, compared to the HEFT algorithm, showing the effectiveness of the proposed algorithm.

VI. Conclusion

We have proposed the HEFT-IPF WFS algorithm for an HMC environment. It characterizes the tasks into three phases: initialization, processing, and finalization, and it performs the dependent task(s) initialization during the finalization of the predecessor task(s). The HEFT-IPF algorithm's performance is compared with the existing algorithm HEFT by considering five scientific workflows and assessed using two performance metrics, MS and RU. The results indicate that the HEFT-IPF algorithm achieves an improvement of 28.36% in MS and

23.33% in RU compared to the HEFT algorithm. However, the proposed algorithm doesn't consider energy while mapping the tasks with clouds, which can be integrated to make it more realistic.

REFERENCES

[1] Rajkumar Buyya, Shashikant Ilager, and Patricia Arroba. Energy-efficiency and sustainability in new generation cloud computing: a vision and directions for integrated management of data centre resources and workloads. *Software: Practice and Experience*, 54(1):24–38, 2024.

[2] Sanjaya K Panda and Prasanta K Jana. An efficient energy saving task consolidation algorithm for cloud computing systems. In *2014 International Conference on Parallel, Distributed and Grid Computing*, pages 262–267. IEEE, 2014.

[3] Sanjaya K Panda and Prasanta K Jana. An efficient resource allocation algorithm for iaas cloud. In *Distributed Computing and Internet Technology: 11th International Conference, ICDCIT 2015, Bhubaneswar, India, February 5-8, 2015. Proceedings 11*, pages 351–355. Springer, 2015.

[4] Sanjaya K Panda and Prasanta K Jana. Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment. *Information Systems Frontiers*, 20:373–399, 2018.

[5] Sanjaya K Panda and Prasanta K Jana. Sla-based task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 73:2730–2762, 2017.

[6] Sanjaya K Panda, Indrajeet Gupta, and Prasanta K Jana. Task scheduling algorithms for multi-cloud systems: allocation-aware approach. *Information Systems Frontiers*, 21:241–259, 2019.

[7] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, pages 3–14. IEEE, 1999.

[8] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[9] Arash Deldari, Mahmoud Naghibzadeh, and Saeid Abrishami. Cca: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. *The journal of Supercomputing*, 73:756–781, 2017.

[10] Madhu Sudan Kumar, Indrajeet Gupta, Sanjaya K Panda, and Prasanta K Jana. Granularity-based workflow scheduling algorithm for cloud computing. *The Journal of Supercomputing*, 73:5440–5464, 2017.

[11] E Ilavarasan, P Thambidurai, and R Mahilmannan. Performance effective task scheduling algorithm for heterogeneous computing system. In *The 4th international symposium on parallel and distributed computing (ISPDC'05)*, pages 28–38. IEEE, 2005.

[12] Sanjaya K Panda and Prasanta K Jana. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *The Journal of Supercomputing*, 71:1505–1533, 2015.

[13] Pegasus Workflow Gallery. Pegasus workflow gallery, 2025. Available at: https://pegasus.isi.edu/workflow_gallery/.

[14] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[15] Sanjaya Kumar Panda and Pabitra Mohan Khilar. A m-level parallel task scheduling. In *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, pages 790–794. IEEE, 2012.

[16] Sanjaya Kumar Panda and Pabitra Mohan Khilar. Mssa: A m-level sufferage-based scheduling algorithm in grid environment. In *International Conference on Distributed Computing and Internet Technology*, pages 410–419. Springer, 2013.

[17] Sanjaya K Panda and Prasanta K Jana. Uncertainty-based qos min–min algorithm for heterogeneous multi-cloud environment. *Arabian Journal for Science and Engineering*, 41:3003–3025, 2016.

[18] Sanjaya Kumar Panda, Sohan Kumar Pande, and Satyabrata Das. Task partitioning scheduling algorithms for heterogeneous multi-cloud environment. *Arabian Journal for Science and Engineering*, 43:913–933, 2018.

[19] Virendra S Gomase and Somnath Tagore. Epigenomics. *Current drug metabolism*, 9(3):232–237, 2008.

[20] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*, pages 1–10. IEEE, 2008.

[21] Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gürsel, Seiji Kawamura, Frederick J Raab, David Shoemaker, Lisa Sievers, Robert E Spero, Kip S Thorne, et al. Ligo: The laser interferometer gravitational-wave observatory. *science*, 256(5055):325–333, 1992.

[22] Robert Graves, Thomas H Jordan, Scott Callaghan, Ewa Deelman, Edward Field, Gideon Juve, Carl Kesselman, Philip Maechling, Gaurang Mehta, Kevin Milner, et al. Cybershake: A physics-based seismic hazard model for southern california. *Pure and Applied Geophysics*, 168:367–381, 2011.

[23] Jonathan Livny, Hidayat Teonadi, Miron Livny, and Matthew K Waldor. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one*, 3(9):e3197, 2008.

[24] G Bruce Berriman, Ewa Deelman, John C Good, Joseph C Jacob, Daniel S Katz, Carl Kesselman, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, and Mei-Hu Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *Optimizing scientific return for astronomy through information technologies*, volume 5493, pages 221–232. SPIE, 2004.

[25] Yassir Samadi, Mostapha Zbakh, and Claude Tadonki. E-heft: enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pages 601–609. IEEE, 2018.

[26] Kalka Dubey, Mohit Kumar, and Subhash Chander Sharma. Modified heft algorithm for task scheduling in cloud environment. *Procedia Computer Science*, 125:725–732, 2018.

[27] Fuquan Sun, Jianjian Cao, and Zhenghao Lu. Heft-dynamic scheduling algorithm in workflow scheduling. In *2022 34th Chinese Control and Decision Conference (CCDC)*, pages 4885–4890. IEEE, 2022.

[28] Tao Hai, Jincheng Zhou, Dayang Jawawi, Dan Wang, Uzoma Oduah, Cresantus Biamba, and Sanjiv Kumar Jain. Task scheduling in cloud environment: optimization, security prioritization and processor selection schemes. *Journal of Cloud Computing*, 12(1):15, 2023.

[29] Sachi Gupta, Sailesh Iyer, Gaurav Agarwal, Poongodi Manoharan, Abeer D Algarni, Ghadah Aldehim, and Kaamran Raahemifar. Efficient prioritization and processor selection schemes for heft algorithm: A makespan optimizer for task scheduling in cloud environment. *Electronics*, 11(16):2557, 2022.

[30] Nitish Chopra and Sarbjeet Singh. Heft based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, 2013.