Article

Efficient Prioritization and Processor Selection Schemes for HEFT Algorithm: A Makespan Optimizer for Task Scheduling in Cloud Environment

Sachi Gupta [1], Sailesh Iyer [2], Gaurav Agarwal [3], Poongodi Manoharan [4,*], Abeer D. Algarni [5], Ghadah Aldehim [6] and Kaamran Raahemifar [7,8,9]

1 Department of Information Technology, IMS Engineering College, Ghaziabad 201015, India
2 Department of Computer Science & Engineering, Rai School of Engineering, Rai University, Ahmedabad 382260, India
3 Department of Computer Science & Engineering, IMS Engineering College, Ghaziabad 201015, India
4 Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Doha 500001, Qatar
5 Department of Information Technology, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia
6 Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia
7 College of Information Sciences and Technology, Data Science and Artificial Intelligence Program, Penn State University, State College, PA 16801, USA
8 School of Optometry and Vision Science, Faculty of Science, University of Waterloo, 200 University Ave W, Waterloo, ON N2L3G1, Canada
9 Faculty of Engineering, University of Waterloo, 200 University Ave W, Waterloo, ON N2L3G1, Canada
* Correspondence: dr.m.poongodi@gmail.com

Citation: Gupta, S.; Iyer, S.; Agarwal, G.; Manoharan, P.; Algarni, A.D.; Aldehim, G.; Raahemifar, K. Efficient Prioritization and Processor Selection Schemes for HEFT Algorithm: A Makespan Optimizer for Task Scheduling in Cloud Environment. *Electronics* **2022**, *11*, 2557. https://doi.org/10.3390/electronics11162557

Academic Editor: Cheng-Chi Lee

Received: 27 June 2022
Accepted: 9 August 2022
Published: 16 August 2022

Abstract: Cloud computing is one of the most commonly used infrastructures for carrying out activities using virtual machines known as processing units. One of the most fundamental issues with cloud computing is task scheduling. The optimal determination of scheduling criteria in cloud computing is a non-deterministic polynomial-time (NP)-complete optimization problem, and several procedures to manage this problem have been suggested by researchers in the past. Among these methods, the Heterogeneous Earliest Finish Time (HEFT) algorithm is recognized to produce optimal outcomes in a shorter time period for scheduling tasks in a heterogeneous environment. Literature shows that HEFT gives extraordinary results in terms of quality of schedule and execution time. However, in some cases, the average computation cost and selection of the first idle slot may not produce a good solution. Therefore, here we propose modified versions of the HEFT algorithm that can obtain improved results. In the rank generation phase, we implement different methodologies for calculating ranks, while in the processor selection phase, we modify the way of selecting idle slots for scheduling the tasks. This paper suggests enhanced versions of the HEFT algorithm under user-required financial constraints to minimize the makespan of a specified workflow submission on virtual machines. Our findings also suggest that enhanced versions of the HEFT algorithm perform better than the basic HEFT method in terms of lesser schedule length of the workflow problems running on various virtual machines.

Keywords: cloud computing; NP-complete; task scheduling; HEFT

1. Introduction

The cloud, or distributed computing worldwide, works on a "pay for every utilization" pattern where clients use services available on the cloud without realizing the hosting specifics and distribution policies [1–3]. This gives appropriate, on-request, and worldwide access permission to a common collection of assets [4] (i.e., computing machines, interconnecting networks, storage space, net facilities, and so forth.) for a shortened time to shop for initiatives and determine logical findings. These assets can be steadily given to clients with less effort and communication with the facility provider [5,6]. Cloud infrastructure aims to provide a simple-to-utilize workspace for continuously changing and adaptable applications. Such a workspace becomes obtainable through a combination of various computer hardware and software package services. These available facilities empower clients to send their submissions through cyberspace by indicating their accessibility, execution, and quality of service (QoS) necessities [7]. Because of the divergent configuration, arrangements, and deployment necessities of such submissions, the involved task scheduling and asset managing approaches [8,9] become essential in developing the global effectiveness and efficiency of the cloud framework. In a distributed framework, any job can be imagined as implementing various tasks involved in it. These tasks can be divided into two major categories: self-determining and reliant tasks. Self-determining tasks can be performed simultaneously by numerous virtual machines (VMs), while reliant tasks should be planned by fulfilling their dependency relationships. The dependency relationships can be presented in the form of a directed acyclic graph (DAG) in which vertices of the graph signify tasks and edges denote interconnections among the tasks [10,11]. It is necessary to perform tasks having precedence constraints in a scheduling sequence, which can reduce the complete scheduling makespan. However, for a task scheduling problem, discovering the optimal results is recognized as NP-complete [10].

Scheduling of tasks in a multiprocessor environment can be generally divided into two key classes: deterministic and non-deterministic scheduling. Deterministic (compile-time) scheduling can be again classified into two subcategories, i.e., guided random search-based (GRSB) [12–14] and heuristics-related [15,16]. Deterministic task scheduling is also known as static scheduling. The GRSB algorithms (genetics-based methods) are more costly than heuristics-related scheduling algorithms as GRSB methods require additional repetitions to generate an improved makespan. On the other hand, the heuristics-related methods provide estimated solutions in less than the polynomial time. The heuristics-related techniques can be classified as duplication-related [17,18], list-based [19–21], and clustering-based [22,23]. The heuristics based on the duplication concept have higher time complexity, while those based on clustering are increasingly reasonable for homogeneous or similar frameworks. In this paper, list-based heuristics are considered because of their reduced time-involvedness and advanced proficiency to deliver a shorter makespan. The list-based heuristics work in two stages for scheduling the queue of tasks. In the initial stage, rank is calculated for individual tasks before arranging them in descending order. In the subsequent stage, the task with the maximum rank value is scheduled on the available machine. Among the list scheduling procedures for diverse computing, the HEFT procedure [24] has gained much popularity due to its low cost and high-performance trade-off. HEFT produces shorter schedule lengths in contrast with other scheduling procedures at a lower cost.

Objectives of the study: In this work, modified versions of the HEFT algorithm are proposed in which different options are used for computing the ranks of the tasks as well as for selecting idle slots, preserving the insertion-based policy of the HEFT algorithm. We find that options other than the average value of computation cost (basic HEFT algorithm) significantly affect the schedule length.

The above-mentioned objective of the paper will be achieved by performing the following tasks in this paper:

- We will lay out the issue of task scheduling on varied machines and the related features in the cloud framework for proficiently arranging the specified jobs on the accessible VMs by including the dependency constraints among the tasks, task's heterogeneity, and the accessible cloud asset's heterogeneity in settling the scheduling conclusions.
- Three versions of the basic HEFT algorithm will be designed and developed.

  MXCT (HEFT with maximum computation cost).
  MNCT (HEFT with minimum computation cost).
  AVBS (HEFT with average commutation cost and best idle slot).

- The above algorithms will compute the ranks of individual tasks and choose the suitable VM for each specified job so that the total finishing time of the arranged jobs is minimized.
- We will systematically assess and relate the suggested algorithms with the basic HEFT algorithm, named here as the AVCT algorithm, on arbitrarily created directed acyclic graphs of real-world applications.

Contributions of this study: This paper contributes to the literature by proposing and implementing some better-enhanced versions of the HEFT algorithm used in different schemes for rank calculation and processor selection. From computational experiments and analyses, it is noticed that there are noteworthy differences between the performance of the original HEFT method (AVCT approach) and modified versions of the HEFT algorithm (MXCT, MNCT, and AVBS) in terms of the produced schedule makespan. These indicate that the schedule length will be affected notably by the scheme used. It is also noticed that the use of an average value scheme for computing the ranks and selecting the first idle slot, as the AVCT algorithm does, is not always a good choice. Our findings also indicate that enhanced versions of the HEFT algorithm perform better in comparison with the basic HEFT procedure in terms of improved makespan of the workflow problems running on various virtual machines.

The structure of this paper is as follows: Section 2 is about related work; Section 3 describes the system model and objective function; Section 4 is the HEFT algorithm; Section 5 explains the proposed methodology; and Section 6 discusses tests and results. Section 7 concludes the paper.

2. Related Work

In the past, various list scheduling procedures have been suggested to solve task scheduling problems. The HEFT algorithm [15] iteratively estimates ascendant rank values of tasks with an average cost of communication and computation. To estimate ascendant rank values, standard deviation-based task scheduling (SDBATS) [10] practices the standard deviation of computing and transmission expenses. Critical Path on a Processor (CPOP) [15] adds ascendant and descendant rank values to make a precedence column and critical track. Performance effective task scheduling (PETS) [25] adds the average cost of computation, cost of data transmission, and cost of data reception to each step of a directed acyclic graph to fix the rank values of specified jobs. Duplication-based Heterogeneous Earliest Finish Time (HEFD) [18] practices task variance as a property of heterogeneousness to estimate computation as well as transmission costs between tasks. PEFT [24] is built on the look-ahead method and calculates an optimistic cost value table to estimate descendent jobs (OCT). The OCT is a two-dimensional array with rows and columns representing the number of jobs and processors. Moreover, each element OCT (t_i, p_i) shows the maximum of the shortest routes of t_i task to the leaving node, seeing that the machine p_i is nominated for task t_i. All of the calculations in these methods are based on the standard deviation or average of task weights on accessible machines, and do not take into account the framework's heterogeneity. The latest effort reflects standard deviation to include tasks and heterogeneity on existing machines. Based on the above literature survey, various task scheduling algorithms [26,27], parameters, tools, improvements, and limitations of algorithms have been analyzed (see Table 1).

Table 1. Review of task scheduling algorithm.

S. No.	Related Research	Working Model	Pross	Cons
1.	2002-Topcuoglu (HEFT) 2002-Topcuoglu (CPOP)	• Prioritizing tasks according to their rank value. • Allocate tasks to suitable processers (which provide the earliest finish time).	• Minimize the make spam. • Lower time complexity.	• provide approximate outcomes, which generally come with polynomial time complexity.
2.	2013-Munir (SDBATS)	• practices the standard deviation of computing and transmission expenses.	• significant reduction in the overall execution time	• when the cost of communication is too high, task scheduling is unfair.
3.	2005-Ilavasaran (PETS)	• adds the average cost of computation, cost of data transmission, and cost of data reception.	• Minimize the makespan.	• it does not consider the path length as used in HEFT
4.	2010-Tang (HEFD)	• practices task variance as a property of heterogeneousness to estimate computation as well as transmission costs between tasks	• Lower time complexity.	• provide approximate outcomes
5.	2013-Arabnejad (PEFT)	• built on the look-ahead method and calculates an optimistic cost value table to estimate descendent jobs	• puts forward the priority weights OCT table by introducing a look-ahead feature	• does not give a reasonable allocation strategy
6.	2016-Bansal	• cost and load balancing metrics	• Improved performance in comparison with the traditional methods.	• Comparison is done only with the traditional methods.
7.	2016-Abdullahi	• Imbalance degree, makespan time, and overall execution time	• Improved performance through reducing the degree of imbalance and timespan	• Only work for load balancing.

Based on the findings, it can be stated that the HEFT method can compete with the current scheduling algorithms in the cloud environment also, but its efficiency can be improved by modifying its prioritization and processor selection methods. We propose that the efficiency of the HEFT algorithm can be enhanced by considering the three versions of the basic HEFT algorithm. This work proposes two schemes of rank calculation and one different approach to idle slot selection. Then, we analyze the makespan of the schedules generated by each version and consider the minimum length makespan the final result. This may increase the cost of the algorithm to some extent, but it is a trade-off between performance and time complexity. Our wide-ranging assessment illustrates that the projected versions produce high-value schedules in terms of reduced schedule length and increased effectiveness.

3. The System Model and Objective Function

The model of scheduling structure contains a submission (application), a target computational framework, and scheduling standards. An application or a problem can be denoted as a DAG = G (T, E, R, C) (see Figure 1), where $\boldsymbol{T} = \boldsymbol{t_i}, \boldsymbol{i} = \boldsymbol{0}, \boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{n} - \boldsymbol{1}$ is a group of n tasks [28–35]. Symbol E signifies a group of edges between tasks $\boldsymbol{E} = \{\boldsymbol{e_{i,j}}, \boldsymbol{i} < \boldsymbol{j}\}$, and $e_{i,j}$ illustrates the precedence restrictions between two related tasks. Tasks $\boldsymbol{t_i}, \boldsymbol{t_j} \in \boldsymbol{T}$, which are linked to one another, denoting the precedence restriction of task t_j being conditional on task t_i for its running. It similarly demonstrates that the result of task t_i will be applied as an input value for task $\boldsymbol{t_j}$, and t_j cannot begin its execution before t_i. Task t_j is the descendant of t_i and t_i is the ancestor of t_i. Here, $\boldsymbol{R}$ represents a 2D matrix of size vxm, and $\boldsymbol{R_{ij}}$ in $\boldsymbol{R}$ represents the expected running time of on $\boldsymbol{j}$**th** processor. A matrix $\boldsymbol{CMC(txt)}$ represents the cost of communication between any two tasks t_i and t_j. In the provided directed acyclic graph, an entering job is one without an ancestor, and an exit task is one without a descendant.

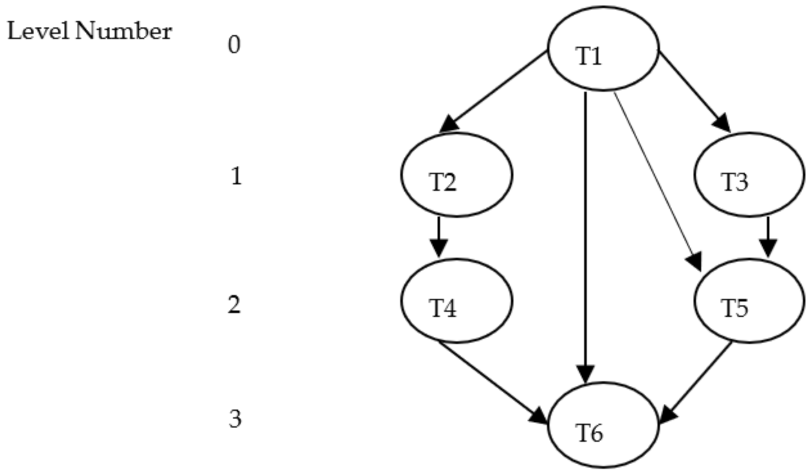

Figure 1. A model DAG.

A cloud framework includes a set $\mathbf{VM} = \{\mathbf{vm_i}, \textbf{ where } \mathbf{i = 0, 1, 2, \ldots, m - 1}\}$ of $\mathbf{m}$ self-regulating VMs which are completely interconnected over a high-rate network (see Figure 2). Due to the varying network bandwidth of cloud infrastructure, the data transfer frequency (DTF) may change. DTF can be represented as a **mxn** two-dimensional array, and between any two VMs as **DTFmxm**. The probable execution cost (PEC) can be represented by an additional two-dimensional array **PECnxm** to perform a task t_i on a VM, $\mathbf{vm_j}$, where $\mathbf{0 \leq i \leq n - 1}$ and $\mathbf{0 \leq j \leq m - 1}$. The PEC builds upon the computational speed of a VM and can be distinct for each one of the VMs.

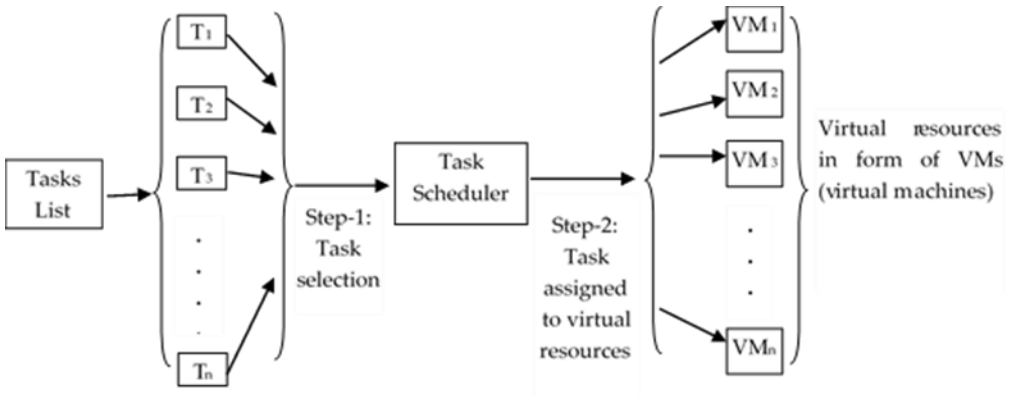
Tasks List
T1
T2
T3
Tn
Step-1: Task selection
Task Scheduler
Step-2: Task assigned to virtual resources
VM1
VM2
VM3
VMn
Virtual resources in form of VMs (virtual machines)

Figure 2. Task scheduling in a cloud-based framework.

The cost of communication between vm_x and vm_y rests on two different aspects. The initial one is the installed frequency at the processors on dual sides of communications and the next is the correspondence cost value of the frequency. We assume that each VM's workstations may send data to other VMs' workstations without causing congestion on the transmission channel. We also consider that jobs planned on the same virtual machine have no communication costs.

The purpose of the task arrangement challenge is to plan all the tasks of a provided submission to machines so that the completion time of a given application is lessened, fulfilling each precedence restriction.

4. Basic HEFT Algorithm

The earliest finish time algorithm in a heterogeneous environment, also called HEFT, is designed for scheduling the tasks of a DAG onto heterogeneous processors. The HEFT procedure has two major levels: the first is the rank generation level and the second is the processor selection phase. In the rank generation phase, HEFT calculates ranks for all the tasks and assigns priorities according to decreasing order of their rank values. Based on the average costs of computation and transmission, we first assign weights to each node and edge of the DAG for rank calculation. In the processor selection stage, HEFT selects the tasks as per their priority values and schedules each nominated task on its most suited processor, which can reduce the task's schedule length. HEFT also follows an insertion-oriented procedure in which a task can be arranged in an empty slot amid two previously planned jobs on a machine if precedence restrictions are well-preserved. The HEFT algorithm searches for an idle slot on a processor until it finds the first empty slot that can carry the cost of computation of the selected task.

The HEFT procedure uses average computation and communication costs as weights in the DAG for rank calculation, and for processor selection, it always considers the first idle slot to schedule the tasks. However, in some cases, the average computation cost and selection of the first idle slot may not generate a good solution. To clarify this, consider a sample DAG (see Figure 3). The edges of sample DAG are labelled with the average communication cost. The probable execution cost of every task on three distinct VMs is shown in Table 2. In this example, if tasks are prioritized using average computation cost over all the three VMs (as in basic HEFT), then scheduling order becomes $T_1, T_2, T_3, T_4, T_6, T_8, T_5, T_{10}, T_7, T_9$ and schedule length becomes 98 (see Figure 4). Suppose priorities are assigned using the maximum value of computation cost over all the three VMs on which a task may run. In that case, the scheduling order of tasks becomes $T_1, T_2, T_3, T_4, T_6, T_8, T_5, T_{10}, T_9, T_7$ and schedule length becomes to 96 (see Figure 5), which is smaller than the schedule length calculated by the basic HEFT algorithm. In a similar way, if priorities are assigned using the minimum value of computation cost over all the three VMs on which a task may run then the scheduling order of tasks and the scheduling length becomes the same as they were found in the basic HEFT algorithm (see Figure 6).

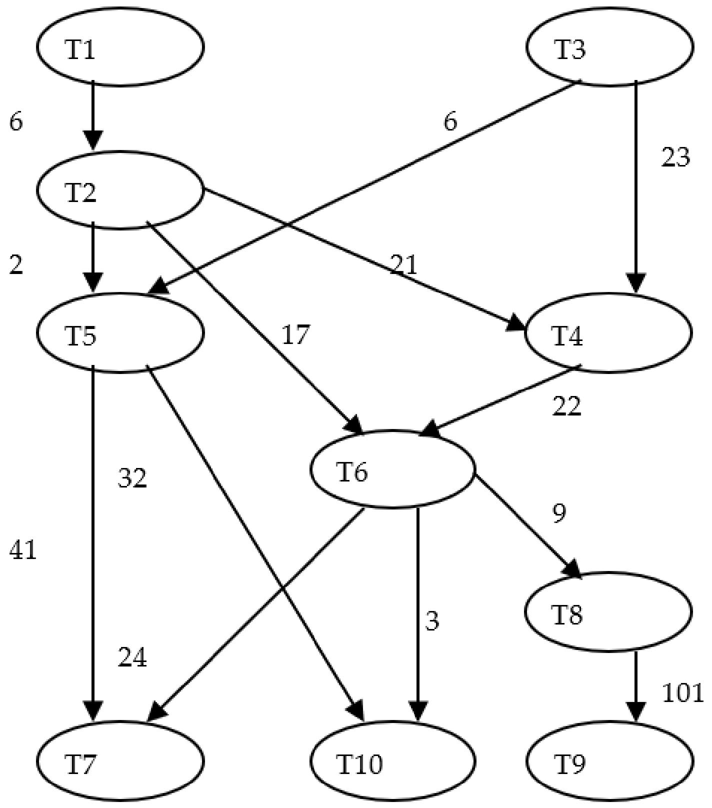

Figure 3. A model DAG with 10 tasks.

Table 2. Probable execution cost (PEC) matrix.

Task	VM1	VM2	VM3
T1	10	6	9
T2	10	23	23
T3	10	9	7
T4	18	17	18
T5	5	2	23
T6	7	5	16
T7	17	9	17
T8	40	16	19
T9	18	9	8
T10	26	10	24

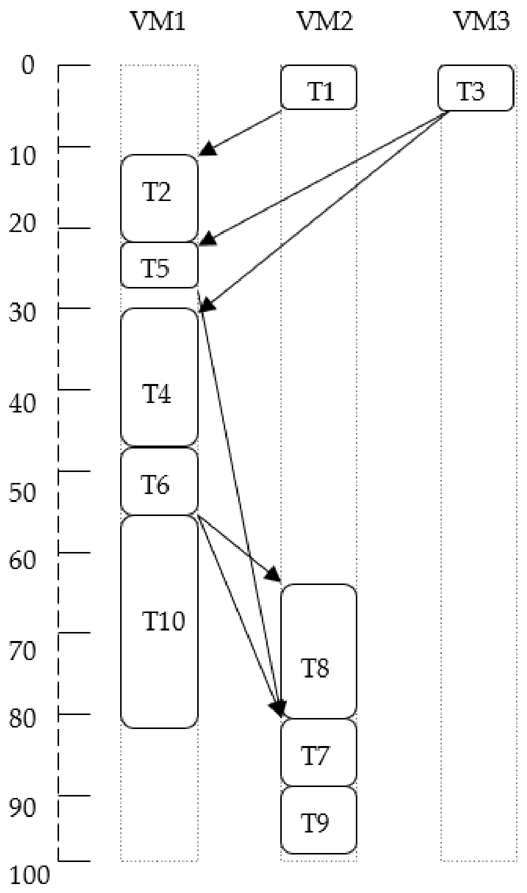

Figure 4. Scheduling of DAG with original HEFT algorithm (use of average computation cost in rank calculation). The schedule length is 98.

On the other hand, if the selection of an idle slot in the processor selection phase is modified, the length of schedules obtained may change. Here, rank calculation is executed by using the average value of computation cost. In the above example, if we select an idle slot in which a task has the least ending time instead of the very initial slot, the schedule length is reduced to 89 as the task $\mathbf{T_5}$ is now scheduled on VM2 (22 to 27, see Figure 7) instead of VM1 (24 to 26 in case of basic HEFT, see Figure 4). Here, we find that the average value of computation cost for calculating ranks and selection of the first idle slot for scheduling of tasks is not necessarily the best effective choices. More importantly, the length of the schedules obtained may vary from one alternative to another. The question that motivated us to perform this work is “what are the significant variations in schedule length” if we modify the basic HEFT algorithm by using different options to calculate the ranks of the tasks as well as different schemes to select an idle slot.

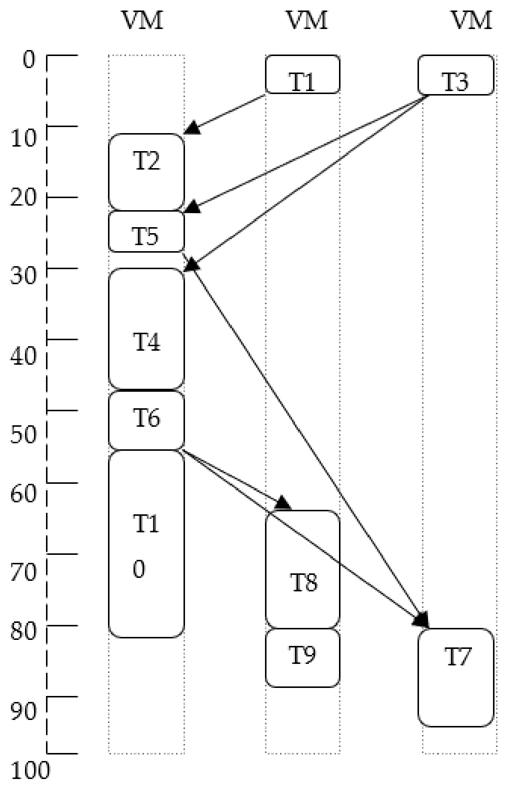

Figure 5. Scheduling of DAG with modified HEFT algorithm (use of maximum computation cost in rank calculation). The schedule length is 96.

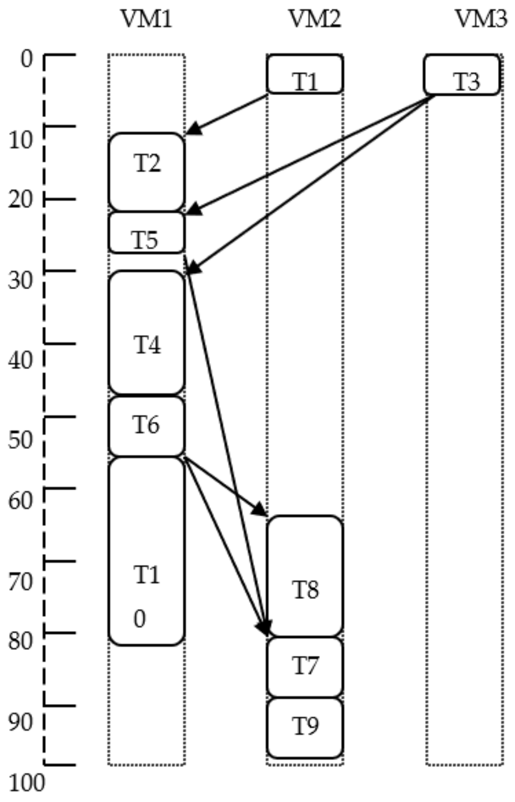

Figure 6. Scheduling of DAG with modified HEFT algorithm (use of minimum computation cost in rank calculation). The schedule length is 98.

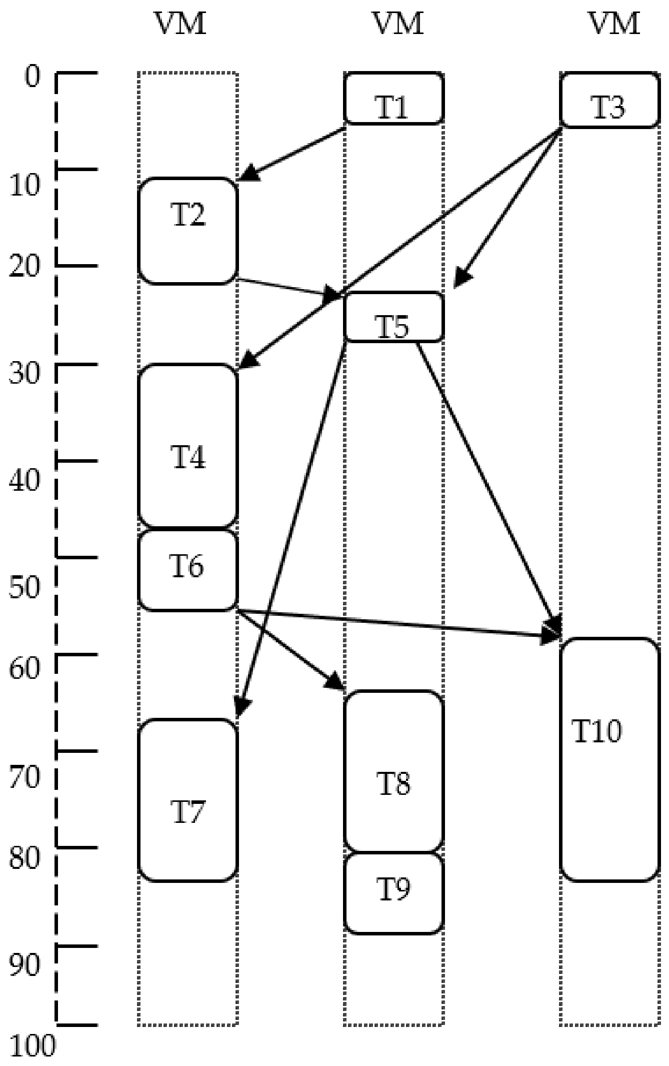

Figure 7. Scheduling of DAG with modified HEFT algorithm (use of minimum schedule length idle slot in processor selection). The schedule length is 89.

5. Proposed Methodology for Cloud Environment

Here we propose the modified versions of the basic HEFT algorithm to obtain more optimal results for task arrangement problems in the cloud environment. In the rank generation phase, we implement a different methodology for calculating ranks, while in the resource selection phase, we modify the way of selecting idle slots for scheduling the given tasks. These modifications do not include any additional cost compared to the basic HEFT algorithm. Proposed changes in the phases of the original HEFT algorithm are as follows (Algorithm 1):

Algorithm 1: HEFT

```
Begin
  Step 1. In DAG, label the nodes and edges with the average values of computation cost and
          communication cost respectively.
  Step 2. Calculate $Rank_{up}(T_i)/Rank_{dw}(T_i)$ using equation (1) or (2)/(3) or (4) for each task by
          passing through the DAG in an upward/ downward direction, starting from the last
          task/first task.
  Step 3. Arrange the tasks in a scheduling queue in order of their decreasing $Rank_{up}(T_i)$ or
          increasing $Rank_{dw}(T_i)$ values.
  Step 4. While there are unallocated tasks in the scheduling queue do
              Pick the first task, $n_i$,from the scheduling queue.
              For each virtual machine $p_k$ in the machine set do
              Calculate Earliest Finish Time ($n_i$,$p_k$) value using the insertion-based
              allocation policy
              Allocate task $n_i$ to the machine $p_j$ that minimizes the EFT value of task
              $n_i$
           End For
  Step 5.  End while.
End
```

5.1. Rank Generation Phase

At this level, the precedence of each task should be determined with the ascendant or descendant rank value. The subsequent formulas recursively compute the upward rank of a task:

If task $\mathbf{T_i}$ is an exit (leaf) task, then the rank of task $\mathbf{T_i}$ is specified by the following rank function:

$$\mathbf{Rank_{up}}(\mathbf{T_i}) = \mathbf{f}\left(\mathbf{w_i^1}, \ldots \mathbf{w_i^k}, \ldots \mathbf{w_i^n}\right) \tag{1}$$

Else

$$\mathbf{Rank_{up}}(\mathbf{T_i}) = \mathbf{f}\left(\mathbf{w_i^1}, \ldots \mathbf{w_i^k}, \ldots \mathbf{w_i^n}\right) + \max_{\forall \mathbf{T_z} \in \ \mathbf{suc}(\mathbf{T_i})} (\mathbf{avg}\ (\mathbf{comm_{i,z}}) + \mathbf{Rank_{up}}(\mathbf{T_z})) \tag{2}$$

Here, $\mathbf{w_i}^{\mathbf{k}}$ is the execution amount of task $\mathbf{T_i}$ on resource **k** and $\mathbf{1 \le k \le n}$, **suc($\mathbf{T_i}$)** is the group of the direct descendants of task $\mathbf{T_i}$ and **avg(comm$_{\mathbf{i,z}}$)** is the average communication cost between the tasks T_i and T_z. Here f denotes a function which can be the max, min, or average value of computation cost. The rank is called upward rank value because it is calculated recursively from the exit node.

If task $\mathbf{T_i}$ is an entry task, then rank of task $\mathbf{T_i}$ is specified by the rank function:

$$\mathbf{Rank_{dw}}(\mathbf{T_i}) = 0 \tag{3}$$

Else

$$\mathbf{Rank_{dw}}(\mathbf{T_i}) = \max_{\forall \mathbf{T_z} \in \ \mathbf{pre}(\mathbf{T_i})} \left(\mathbf{avg}\ (\mathbf{comm_{z,i}}) + \mathbf{Rank_{dw}}(\mathbf{T_z}) + \mathbf{f}\left(\mathbf{w_i^1}, \ldots \mathbf{w_i^k}, \ldots \mathbf{w_i^n}\right)\right) \tag{4}$$

where **pre($\mathbf{T_i}$)** is the set of immediate ancestors of task $\mathbf{T_i}$.

After computing the ranks for each task, a task list is produced by arranging the tasks according to their reducing order of $Rank_{up}$.

5.2. Resources' Selection Phase

We propose a different approach for determining the idle slot for chosen task. In the proposed methodology, the hunt for a suitable idle slot for a task on a resource starts when all predecessors of task T_i sent the required input data to that resource. The search persists until obtaining an idle slot that is competent enough to hold the computation cost of task T_i and in which selected task T_i has the least finish time.

6. Experiments, Results, and Discussion

In this effort, we have designed a system that implements the basic HEFT algorithm, permitting various possibilities for task prioritization and processor selection. Its inputs are the number of tasks, number of VMs, probable execution cost matrix, communication cost, and a DAG (which shows dependencies). To assess the efficiency of our projected algorithms, we produce scheduling problems of varying ranges and solve them by basic and modified versions of HEFT algorithms. Specifics of experimental arrangement and findings obtained by basic HEFT and modified versions of HEFT algorithm are mentioned below:

6.1. Experimental Arrangement

We developed a system that generates essential-size scheduling problems automatically. This is done to avoid biasing while selecting values for various parameters that are essential for the problems. Our scheme generates random values in suitable ranges for these parameters. With the following properties, we have generated problems for our tests:

- Problem size (No. of tasks) ranges from 50 to 80 with an interval of 5.
- Every task, excluding the exit task, has an arbitrary number of offspring between 0 and 10.
- Each task's implementation time can be any value between 1 and 20.
- The communication time between tasks is an arbitrary number that ranges from 1 to 50.
- The total quantity of VMs is considered as either 4 or 5.

In each DAG, the ranks of tasks are calculated by the upward rank calculation formula.

6.2. Investigation on Rank Generation Phase

Three techniques are used to calculate a value for the method f in rank function: (i) The AVCT (average computation cost) approach returns an average computation cost of a task over all the VMs (this approach is used in basic HEFT). (ii) The MXCT (maximum computation cost) approach returns the maximum computation cost of a task over all the VMs. (iii) The MNCT (minimum computation cost) approach returns the minimum computation cost of a task over all the VMs. In all three approaches, the very first idle slot that can hold the computation cost of a task is considered (as in basic HEFT). We run the basic HEFT algorithm (AVCT approach) as well as our proposed schemes (MXCT and MNCT approaches) on a hundred diverse problems with problem identification numbers (PIN) 800 to 899 for problem size 80. Our experiments show that for 33% of problems, all three algorithms give equal schedule lengths. For the remaining 67% of problems, the MXCT approach gives equal schedule lengths in 36% of cases, better schedule lengths in 39% of cases, and worse schedule lengths in 25% of cases in comparison with the AVCT approach. Variations in schedule lengths obtained from the MXCT algorithm and AVCT algorithm are shown in Figure 8. Similarly, for the rest, 67% of problems, the MNCT approach gives equal schedule lengths in 27% of cases, better schedule lengths in 40% of cases, and worse schedule lengths in 33% of cases compared to the AVCT approach. The variations in schedule lengths obtained from the MNCT algorithm and AVCT algorithm are shown in Figure 9. Table 3 represents a comparison of all three approaches for rank calculation.

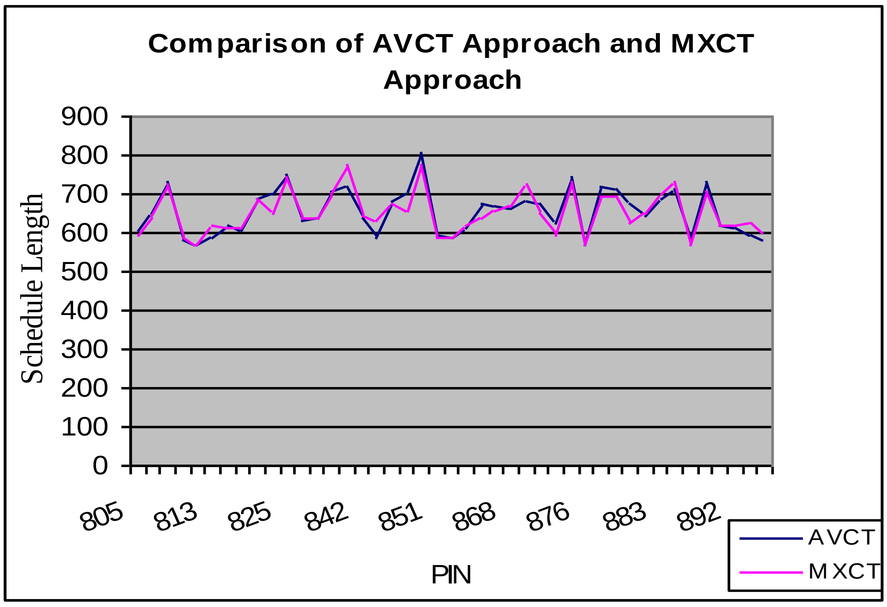

Figure 8. Variations in schedule length obtained from AVCT approach and MXCT approach.

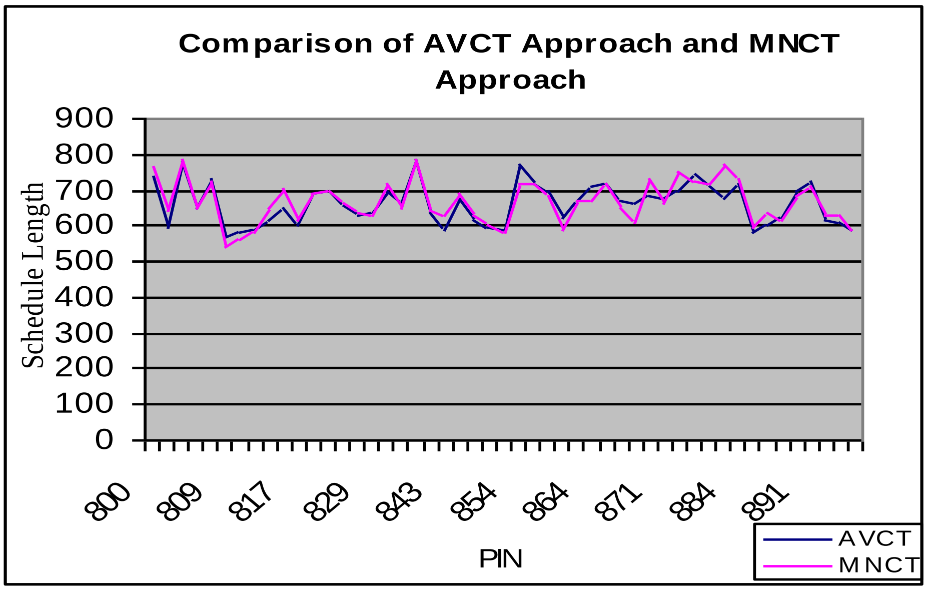

Figure 9. Variations in schedule length obtained from AVCT approach and MNCT approach.

Table 3. Comparison of AVCT, MXCT, and MNCT algorithms.

Problem Percentage	Approaches		Outcomes
33%	AVCT MXCT MNCT		Equal schedule length
67%	MXCT vs. AVCT	36% 39% 25%	Equal schedule length MXCT gives better schedule length MXCT gives worse schedule length
	MNCT vs. AVCT	27% 40% 33%	equal schedule length MNCT gives better schedule length MNCT gives worse schedule length

From the analysis, it is experiential that there are significant differences amid the performance of the basic HEFT method (AVCT approach) and modified versions of the HEFT algorithm (MXCT and MNCT approaches). It is also noticed that the use of an average value scheme for computing the ranks is not always a good choice. The result analysis shows that MXCT and MNCT approaches can give better schedule length in comparison with the basic HEFT algorithm, but it will be a time-consuming job to run all three algorithms and find the best results.

6.3. Investigation on Resource Selection Phase

Two approaches are used to select an idle slot for scheduling a task: (i) AVCT approach (basic HEFT algorithm—average commutation cost and very first idle slot). (ii) AVBS (average commutation cost and best idle slot) approach always uses average computation cost for calculating ranks and selects an idle slot in which the selected task has the least finish time. We have taken a hundred sample problems of each size, ranging from 50 to 80 with an interval of 5. We test both the algorithms on sample problems and analyze the results obtained from both the algorithms (see Table 4).

Table 4. Comparison of AVCT and AVBS algorithm.

Problem Size	No. of Resources	AVCT Algorithm (Average of Hundred Problems)	AVBS Algorithm (Average of Hundred Problems)
50	4	425.50	425.14
55	4	457.34	457.32
60	4	491.12	490.94
65	4	548.24	548.26
70	4	574.66	574.18
75	5	623.21	622.02
80	5	665.97	665.61

According to analysis, the AVBS approach outperforms the AVCT algorithm. The outcomes show that the AVBS procedure gives a shorter average schedule length in 86% of problem sets and a slightly larger average schedule length in only 14% of problem sets compared to the AVCT algorithm.

7. Conclusions

Cloud computing is a mode of computation where significantly scalable resources are provided as services to clients using the Internet. Therefore, a cloud service provider has to attend to more clients in the cloud computing framework. Task scheduling has consequently emerged as one of the major challenges in setting up a cloud computing environment. In this study, we proposed various versions of the heuristic-based algorithm (HEFT) that perform tasks' scheduling and assign resources optimally in the cloud computing environment. In terms of schedule length, we find that our suggested heuristic method performs better when compared to alternative methods, while the complexity of the proposed algorithms remains the same as that of the basic HEFT algorithm. It is observed that the efficiency of the original HEFT algorithm can be improved by selecting the best result from the schedules obtained by each approach. This will lead to more time consumption as three algorithms will be run, but there may be a trade-off between performance and running cost.

Future Work

Nature-inspired optimization algorithm-based scheduling can be further considered for attaining more efficient task scheduling in the cloud context. Moreover, the existing work can be extended for dynamic scheduling of tasks.

Author Contributions: Conceptualization, S.G. and S.I.; methodology, G.A. (Gaurav Agarwal); software, S.I.; validation, A.D.A. and G.A. (Ghadah Aldehim); formal analysis, A.D.A.; investigation, P.M.; resources, K.R.; data curation, G.A. (Gaurav Agarwal); writing—original draft preparation, S.G.; writing—review and editing, S.I.; visualization, A.D.A.; supervision, P.M.; project administration, G.A. (Ghadah Aldehim). All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2022R51), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Acknowledgments: The authors would like to acknowledge Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2022R51), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Conflicts of Interest: The authors declare that they have no conflict of interest to report regarding the present study.

References

1. Zomaya, A.Y. *Parallel and Distributed Computing Handbook*; McGraw-Hill: New York, NY, USA, 1996.
2. Buyya, R.; Yeo, C.S.; Venugopal, S.; Broberg, J.; Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* **2009**, *25*, 599–616. [CrossRef]
3. Tziritas, N.; Khan, S.U.; Xu, C.-Z.; Hong, J. An Optimal Fully Distributed Algorithm to Minimize the Resource Consumption of Cloud Applications. In Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems, Singapore, 17–19 December 2012; pp. 61–68.
4. Gupta, P.; Sharma, R.; Gupta, S. Resource Management, Issues, Challenges and Future Directions in Fog Computing: A Comprehensive Survey. *Des. Eng.* **2021**, *7*, 14580–14593.
5. Li, J.; Li, Q.; Khan, S.U.; Ghani, N. Community-based cloud for emergency management. In Proceedings of the 2011 6th International Conference on System of Systems Engineering, Albuquerque, NM, USA, 27–30 June 2011.
6. Hashemi, S.M.; Bardsiri, A.K. Cloud computing vs. grid computing. *ARPN J. Syst. Softw.* **2012**, *2*, 88–194.
7. Calheiros, R.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.F.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2010**, *41*, 23–50. [CrossRef]
8. Agarwal, G.; Maheshkar, V.; Maheshkar, S.; Gupta, S. Vocal Mood Recognition: Text Dependent Sequential and Parallel Approach. In Proceedings of the International Conference on Signals, Machines and Automation (SIGMA'18), New Delhi, India, 23–25 February 2018.
9. Shin, K.S.; Park, M.-J.; Jung, J.-Y. Dynamic task assignment and resource management in cloud services by using bargaining solution. *Concurr. Comput. Pract. Exp.* **2013**, *26*, 1432–1452. [CrossRef]
10. Munir, E.U.; Mohsin, S.; Hussain, A.; Nisar, M.W.; Ali, S. SDBATS: A Novel Algorithm for Task Scheduling in Heterogeneous Computing Systems. In Proceedings of the 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, 20–24 May 2013; pp. 43–53.
11. Radulescu, A.; Van Gemund, A.J. Fast and effective task scheduling in Heterogeneous system. In Proceedings of the 9th Heterogeneous Computing Workshop, Cancun, Mexico, 1 May 2000.
12. Gupta, S.; Mittal, V.; Agarwal, G. Task Scheduling in Multiprocessor System Using Genetic Algorithm. In Proceedings of the 2nd International Conference on Machine Learning and Computing (ICMLC-2010), Bangalore, India, 12–13 February 2010.
13. Gupta, S.; Agarwal, G.; Mittal, V. An Efficient and robust Genetic Algorithm for Multiprocessor Scheduling. *Int. J. Comput. Theory Eng.* **2013**, *5*, 1793–8201. [CrossRef]
14. Goldberg, D.C. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley: Boston, MA, USA, 1989.
15. Topcuouglu, H.; Hariri, S.; Wu, M.-y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
16. Gotoda, S.; Ito, M.; Shibata, N. Task scheduling algorithm for multi-core processor system for minimizing recovery time in case of single node fault. In Proceedings of the IEEE CCGRID, Ottawa, ON, Canada, 13–16 May 2012; pp. 260–267.
17. Mei, J.; Li, K.; Li, K. A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. *J. Supercomput.* **2014**, *68*, 1347–1377. [CrossRef]
18. Tang, X.; Li, K.; Liao, G.; Li, R. List scheduling with duplication for heterogeneous computing systems. *J. Parallel Distrib. Comput.* **2010**, *70*, 323–329. [CrossRef]
19. Agarwal, G.; Gupta, S.; Saxena, P.; Mukherjee, S. Web Graph Based Ranking Algorithm for Search Engines. In Proceedings of the International Conference on Network Communication and Computer (ICNCC-2011), New Delhi, India, 19–20 March 2011; ISBN 9781424495504.
20. Gupta, S.; Mukherjee, S.; Agarwal, G. List Scheduling Heuristic: Efficient Prioritization and Processor Selection Schemes for Heft Algorithm. In Proceedings of the International Conference on Industrial Applications of Soft Computing Techniques (IIASCT-2011), Odisha, India, 20–22 August 2011; ISBN 9789381361221.

21. Agarwal, G.; Gupta, S.; Mukherjee, S. Web Graph Based Search by Using Density of Keywords and Age Factor. In Proceedings of the International Conference on Computer Science and Information Technology (ICCSIT-2012), Hong Kong, China, 29–30 December 2012; ISBN 9789381693766.
22. Cirou, B.; Jeannot, E. Triplet: A clustering scheduling algorithm for heterogeneous systems. In Proceedings of the International Conference on Parallel Processing Workshop, Valencia, Spain, 3–7 September 2001.
23. Fiore, U.; Palmieri, F.; Castiglione, A.; de Santis, A. A cluster-based data-centric model for network-aware task scheduling in distributed systems. *Int. J. Parallel Program.* **2014**, *42*, 755–775. [CrossRef]
24. Arabnejad, H.; Barbosa, J.G. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 682–694. [CrossRef]
25. Ilavarasan, E.; Thambidurai, P.; Mahilmannan, R. Performance effective task scheduling algorithm for heterogeneous computing system. In Proceedings of the ISPDC. IEEE Computer Society, Lillie, France, 4–6 July 2005; pp. 28–38.
26. Bansal, N.; Awasthi, A.; Bansal, S. Task Scheduling Algorithms with Multiple Factor in Cloud Computing Environment. In *Information Systems Design and Intelligent Applications*; Springer: Berlin, Germany, 2016.
27. Abdullahi, M.; Ngadi, A.; Abdulhamid, S.M. Symbiotic Organism Search optimization based task scheduling in cloud computing environment. *Future Gener. Comput. Syst.* **2016**, *56*, 640–650. [CrossRef]
28. Dai, Y.; Zhang, X. A Synthesized Heuristic Task Scheduling Algorithm. *Sci. World J.* **2014**, *2014*, 465702. [CrossRef] [PubMed]
29. Ali, S.; Siegel, H.; Maheswaran, M.; Hensgen, D. Task execution time modelling for heterogeneous computing systems. In Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No. PR00556), Cancun, Mexico, 1 May 2000; pp. 185–199.
30. Agarwal, G.; Om, H. Parallel training models of deep belief network using MapReduce for the classifications of emotions. In *International Journal of System Assurance Engineering and Management*; Springer: Berline, Germany, 2021.
31. Rathore, M.S.; Poongodi, M.; Saurabh, P.; Lilhore, U.K.; Bourouis, S.; Alhakami, W.; Osamor, J.; Hamdi, M. A novel trust-based security and privacy model for Internet of Vehicles using encryption and steganography. *Comput. Electr. Eng.* **2022**, *102*, 108205. [CrossRef]
32. Poongodi, M.; Bourouis, S.; Ahmed, A.N.; Vijayaragavan, M.; Venkatesan, K.G.S.; Alhakami, W.; Hamdi, M. A Novel Secured Multi-Access Edge Computing based VANET with Neuro fuzzy systems based Blockchain Framework. *Comput. Commun.* **2022**, *192*, 48–56.
33. Ramesh, T.R.; Lilhore, U.K.; Poongodi, M.; Simaiya, S.; Kaur, A.; Hamdi, M. Predicitive Analysis of Heart Diseases with Machine Learning Approaches. *Malays. J. Comput. Sci.* **2022**, 132–148.
34. Poongodi, M.; Malviya, M.; Hamdi, M.; Vijayakumar, V.; Mohammed, M.A.; Rauf, H.T.; Al-Dhlan, K.A. 5G based Blockchain network for authentic and ethical keyword search engine. *IET Commun.* **2022**, *16*, 442–448.
35. Poongodi, M.; Malviya, M.; Kumar, C.; Hamdi, M.; Vijayakumar, V.; Nebhen, J.; Alyamani, H. New York City taxi trip duration prediction using MLP and XGBoost. *Int. J. Syst. Assur. Eng. Manag.* **2022**, *13*, 16–27. [CrossRef]