EMCS: An Energy-Efficient Makespan Cost-Aware Scheduling Algorithm Using Evolutionary Learning Approach for Cloud-Fog-Based IoT Applications

Ranumayee Sing 1, Sourav Kumar Bhoi 2, Niranjan Panigrahi 2, Kshira Sagar Sahoo 3,4, Muhammad Bilal 5,* and Sayed Chhattan Shah 6

1 Faculty of Engineering (Computer Science and Engineering), BPUT, Rourkela 769015, Odisha, India
2 Department of Computer Science and Engineering, Parala Maharaja Engineering College (Govt.), Berhampur 761003, Odisha, India
3 Department of Computer Science and Engineering, SRM University, Amaravati 522240, Andhra Pradesh, India
4 Department of Computer Science, Umeå University, SE-901 87 Umeå, Sweden
5 Department of Computer Engineering, Hankuk University of Foreign Studies, Yongin-si 17035, Republic of Korea
6 Department of Information and Communication Engineering, Hankuk University of Foreign Studies, Yongin-si 17035, Republic of Korea

* Correspondence: m.bilal@ieee.org

Abstract: The tremendous expansion of the Internet of Things (IoTs) has generated an enormous volume of near and remote sensing data, which is increasing with the emergence of new solutions for sustainable environments. Cloud computing is typically used to help resource-constrained IoT sensing devices. However, the cloud servers are placed deep within the core network, a long way from the IoT, introducing immense data transactions. These transactions require heavy electricity consumption and release harmful CO₂ to the environment. A distributed computing environment located at the edge of the network named fog computing has been promoted to reduce the limitation of cloud computing for IoT applications. Fog computing potentially processes real-time and delay-sensitive data, and it reduces the traffic, which minimizes the energy consumption. The additional energy consumption can be reduced by implementing an energy-aware task scheduling, which decides on the execution of tasks at cloud or fog nodes on the basis of minimum completion time, cost, and energy consumption. In this paper, an algorithm called energy-efficient makespan cost-aware scheduling (EMCS) is proposed using an evolutionary strategy to optimize the execution time, cost, and energy consumption. The performance of this work is evaluated using extensive simulations. Results show that EMCS is 67.1% better than cost makespan-aware scheduling (CMaS), 58.79% better than Heterogeneous Earliest Finish Time (HEFT), 54.68% better than Bees Life Algorithm (BLA) and 47.81% better than Evolutionary Task Scheduling (ETS) in terms of makespan. Comparing the cost of the EMCS model, it uses 62.4% less cost than CMaS, 58.79% less than BLA, and 5.16% less than ETS. When comparing energy consumption, EMCS consumes 11.55% less than CMaS, 7.56% less than BLA, and 3.87% less than ETS. Results also show that with an increase in the number of fog and cloud nodes, the balance between cloud and fog nodes gives better performance in terms of makespan, cost, and energy consumption.

Keywords: cloud computing; fog computing; EMCS; task scheduling; evolutionary approach

1. Introduction

In recent times, IoT has revolutionized the complete domain of information and communication technology (ICT) environment. Every application these days is IoT-ized to some convenient extent suitably depending upon its technical requirement and economic viability. The heterogeneous, wired, or wireless IoT devices (sensors, actuators, smart
cameras, vehicles, smart metering, etc.) provide several different remote services on users' demand [1,2]. In most applications, the massive generated data need to be processed and stored to meet the imminent demands at further processing nodes or application consoles. However, limitations on different fronts such as storage capacity, battery life, network resources, computing capability, and processing power restrained the services, and this enforces the transmission of services to a cloud server.

Furthermore, the ambient world is now connected with rapidly growing IoT devices which is expected to reach 75.4 billion in 2025 [3]. As this number increases, the traditional centralized cloud computing system is unable to handle the ICT requirements within due precision. Most of the sensitive applications such as health monitoring, traffic control, vehicle network, etc. in such consequences may lead to devastating outcomes on the ground. To address this, the network loading needs to be significantly decluttered. Due to the large distance from IoT to the cloud, the transmission of extreme data through the Internet is a major load on network performance. It is also very expensive to use cloud services constantly. Depending on applications, many edge devices of the network (e.g., router, gateways, personal computer, workstations) are powered with some processing, storage and/or communication capabilities. Computing at such middle layer edge devices needs to be reinforced and are now represented as fog computing.

Fog computing [4], proposed by Cisco, is the new distributed computing paradigm at the edge of network for implementing IoT applications [5–7]. Fog computing extends cloud computing by being closer to the IoT devices. Any devices (controller, router, gateways, embedded server, surveillance camera, etc.) which can compute, store, and communicate can be considered as fog nodes or distributed servers. The tasks that are small or delay sensitive are given priority to be processed in the fog nodes, whereas the delay-tolerant and large-scale tasks are moved to the cloud computing platform. The only additional requirement here is that devices in every layer have to use power even if it is in idle mode. Eventually, a new model can be rolled out as cloud-fog computing addressing the decluttering through extensive protocol-based edge computing.

An estimation of the carbon emission share attributed to the energy consumption in cloud data centers puts it at 3.2% of total worldwide carbon emissions by 2025 [1,8]. This paradigm shift for cloud-fog computing protocol bears significance not only in the ICT arena but it also will be a promising solution in minimizing the cloud energy consumptions significantly contributing to restrict the already acute global worming causing issues.

The cloud-fog paradigm also brings with it lucrative advantages like minimum latency time, less network traffic, and energy efficiency etc. along with some associated challenges [3]. The most important challenges are resource allocation and task scheduling among the cloud and fog in such a way that the criteria of makespan, cost, energy, deadline, and security can be minimized. This task scheduling problem assigned to different environments is generally an NP-hard problem, and researchers proposed different meta-heuristic approaches for the solutions [9–12]. In the present work, a genetic algorithm (GA) is used to obtain an optimal solution for the directed acyclic graph (DAG) task-scheduling problem in the cloud-fog model. The main contributions are as follows:

1. An algorithm named EMCS task scheduling is proposed for cloud-fog-based IoT applications.
2. The multi-objective optimization algorithm minimizes execution time, cost, and energy consumption simultaneously, while the parameters are adjusted using an evolutionary method to obtain better performance.
3. A comparison study is evaluated among EMCS, CMaS [1], HEFT [13], BLA [14] and ETS [15] for performance evaluation and validation. The simulation results of EMCS show better performance and are more optimal.
4. The simulation results with various cloud and fog nodes also prove that the balance between cloud and fog nodes gives better performance.

The rest of the work is presented as follows. The study on task scheduling algorithms is discussed in Section 2. The system architecture and problem formulation are described
in Section 2. The proposed EMCS task scheduling algorithm is elaborated in Section 4. Section 5 presents the performance of EMCS algorithm. Section 6 presents with conclusion and gives a future scope of the work.

2. Related Works

Many researchers have tried to distinguish between cloud and fog, and they also focused on the challenges and issues in fog computing. Advanced concepts such as MCC (mobile cloud computing), MEC (mobile edge computing), dew computing, edge computing, fog computing were examined by Naha et al. [5]. The architecture and research directions of fog computing are also focused on in this work. The basic model for networking, latency, and energy is discussed well in a survey presented by Mukherjee et al. [6]. The issues and challenges of application offloading, resource management, heterogeneity, and SDN (software-defined networking)-based fog computing are also described.

A major focus area of different network paradigms is the scheduling of tasks. Table 1 summarizes the models, policies and limitations of the task-scheduling algorithms. Some task-scheduling algorithms are discussed as follows. Pham et al. [1] considered DAG-based tasks with priorities. With the concept of the reassignment of tasks on the basis of a critical path for completing the execution before the deadline, a cost-makespan aware scheduling algorithm is developed. Nguyen et al. [3] applied an evolutionary strategy in TCaS (Time–Cost-aware Scheduling) algorithm for scheduling the Bag-of-Tasks in the cloud-fog model. Bitam et al. [14] proposed a new optimization method inspired by the natural life of bees called BLA, which is based on the population concept for collaborating the behaviors of bees, including marriage (reproduction) and food foraging. The initial population \( N \) is evaluated with the fitness function. The whole population is categorized as queen (best individual), drone (next \( D \) best individuals) and workers (remaining). Among the two phases of marriage and food forage, the queen mates with some drones in the marriage phase and produces broods by crossover and mutation operations. The best individual replaces the previous queen, and the next \( D \) fittest bees become drones and others are \( W \) workers. In the food forage, each individual is trying to search for food, and the workers collect the food. The best workers will remain in the next generation. This algorithm improves the CPU execution time and memory utilization of tasks. On the basis of rank for prioritizing the tasks, Topcoglu et al. [13] developed two algorithms: HEFT and CPOP (Critical-Path-on-a-Processor). Two processes, prioritization of task and selection of processors for the task execution, are considered in HEFT, whereas the critical path of the processors is considered in CPOP for minimizing the earliest finish time. Liu et al. [16] designed an Adaptive Doublefitness Genetic Task Scheduling algorithm that considers makespan and communication cost as doublefitness for scheduling tasks and fog resources. On the basis of the transaction set for scheduling and association rules of the nodes, the task set is generated, and using the association rules, the tasks are scheduled among the fog nodes. Based on this mining concept, the I-Apriori algorithm is developed by Liu et al. [17] for minimizing average waiting time and total excution time. Xu et al. [18] applied laxity for sequencing the priority of the task and applied ant colony optimization to reduce the energy consumption on satisfying mixed deadlines in hybrid cloud-fog computing. Benblidia et al. [19] ranked the fog nodes by considering the preferences of the users and fog nodes features, applying a linguistic quantifier and fuzzy quantified proposition. Zhang et al. [20] designed a delay optimal task scheduling (DOTS) algorithm which reduces the finish time. Boveiri et al. [21] adopted an approach of the Max–Min Ant system which solved a scheduling problem represented as a task graph in a multiprocessor system. Aladwani [22] developed a method called Tasks Classifications and Virtual Machines Categorization (TCVC) based on max–min scheduling applied to a health monitoring system and reduced the total finish time, execution time, and waiting time. Abdulredha et al. [15] used an evolutionary task-scheduling algorithm which can minimize the makespan and cost.

Here, we also studied some articles related to energy consumption in both cloud and fog systems. Using power-aware task scheduling, Zhao et al. [20] tried to predict the power...
consumption of a heterogeneous cloud platform. Jena [23] modeled a clonal selection-based algorithm (TSCSA) to utilize the resources effectively and reduce energy and execution time. Alla et al. [24] ranked each task based on a TOPSIS multi-criteria decision-making strategy. The ranked tasks are sent to the proper VM from dynamic priority queues through an energy-efficient dispatcher. It effectively reduces the makespan and energy consumption. Wang and Li [25] formed a hybrid heuristic algorithm adopting two algorithms, IPSO and IACO, in which delay and energy consumption are the main objectives for task scheduling. In [26], the tasks generated from camera sensors and actuators are scheduled with knapsack-based symbiotic organism scheduling that reduces energy and energy. Similarly, in [27], a greedy-based knapsack scheduling approach was followed for allocating resources in fog networks to improve energy consumption, execution cost, and delay. Wu and Lee [28] obtained the optimal scheduling strategy for energy minimization by solving the Integer Linear Programming model. A heuristic algorithm called Energy Minimization Scheduling (EMS) was proposed for reducing energy consumption. A hyper-heuristic technique of GA, PSO, ACO, and SA for current workflow is designed by Kabirzadeh et al. [10], which selects the best algorithm according to the lowest Euclidean distance. Three different heuristic algorithms (greedy based, group-based, and GA algorithms) are followed to schedule the tasks of mobile users on mobile cloud computing. This approach was proposed by Tang et al. [29], which reduces energy consumption by considering task dependency, transmission, task deadline, and cost. Li et al. [30] modeled an energy optimization algorithm considering energy consumption in each layer with nonlinear programming and delay optimization in STML (Scheduling to Minimize Lateness) approach. Wu et al. [9] adopted a partition operator to divide the graph into two non-overlapping sub-graphs. The proposed EDA (Estimation of Distributed Algorithm) has sorted the tasks for the fog and cloud layer according to their precedence. The permutation of tasks reduces the energy consumption and makespan along with a lifetime of IoT. Many such task-scheduling strategies on distributed systems can also be found in [31–45].

From the above works discussed, most of the papers considered the minimum completion of time and energy. In this paper, the consumption of energy for both fog and cloud are considered. The energy consumed in fog node is the energy usage for the processing of the tasks and energy required for an idle node. The energy consumed in the cloud node is the sum of the energy usage for processing and the energy usage for the transmission of the tasks and data to the cloud node. The evolutionary method is applied in the proposed approach for optimal makespan, cost, and energy consumption.

Table 1. Some related works on task scheduling in different systems.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Article</th>
<th>Tar System</th>
<th>Ideas</th>
<th>Improved Criteria</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pham et al. (2017) [1]</td>
<td>Cloud-Fog System</td>
<td>Utility function considering makespan and mandatory cost to prioritized tasks</td>
<td>Execution time and mandatory cost</td>
<td>Small dataset</td>
</tr>
<tr>
<td>2</td>
<td>Nguyen et al. (2019) [3]</td>
<td>Cloud-Fog System</td>
<td>Genetic algorithm</td>
<td>Makespan and total cost</td>
<td>Budget, deadline, and resource limitations are not considered</td>
</tr>
<tr>
<td>5</td>
<td>Liu et al. (2018b) [16]</td>
<td>Fog Computing</td>
<td>Genetic algorithm</td>
<td>Makespan and communication cost</td>
<td>Based on the smart city database</td>
</tr>
<tr>
<td>6</td>
<td>Liu et al. (2018a) [17]</td>
<td>Fog Computing</td>
<td>Classification of mining</td>
<td>Execution time and waiting time</td>
<td>Bandwidth between processors is not considered</td>
</tr>
<tr>
<td>7</td>
<td>Xu et al. (2019) [18]</td>
<td>Cloud-Fog System</td>
<td>Laxity and ant colony optimization</td>
<td>Energy consumption</td>
<td>Small dataset</td>
</tr>
</tbody>
</table>
Table 1. Cont.

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Article</th>
<th>Target System</th>
<th>Ideas</th>
<th>Improved Criteria</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Benblidia et al. (2019) [19]</td>
<td>Cloud-Fog System</td>
<td>Fuzzy logic</td>
<td>Execution delay and energy consumption</td>
<td>Very small dataset</td>
</tr>
<tr>
<td>9</td>
<td>Boveiri et al. (2019) [21]</td>
<td>Cloud System</td>
<td>Max–Min ant system</td>
<td>Scheduling length</td>
<td>Small dataset and in the cloud environ-ment</td>
</tr>
<tr>
<td>10</td>
<td>Aladwani (2019) [22]</td>
<td>Fog Computing</td>
<td>Max–Min system</td>
<td>Total execution time, total waiting time, and total finish time</td>
<td>Small dataset</td>
</tr>
<tr>
<td>11</td>
<td>Zhao et al. (2019) [20]</td>
<td>Cloud Computing</td>
<td>Greedy-based algorithm</td>
<td>Power consumption</td>
<td>Network optimization is not considered</td>
</tr>
<tr>
<td>12</td>
<td>Jena (2017) [23]</td>
<td>Cloud Computing</td>
<td>Clonal selection algorithm</td>
<td>Energy consumption and makespan</td>
<td>Data centers and jobs are dynamic</td>
</tr>
<tr>
<td>13</td>
<td>Alla et al. (2019) [24]</td>
<td>Cloud Computing</td>
<td>Best-worst and TOPSIS method</td>
<td>Makespan and energy consumption</td>
<td>Not used in the large-scale data center</td>
</tr>
<tr>
<td>14</td>
<td>Wang et al. (2019) [25]</td>
<td>Fog Computing</td>
<td>Hybrid heuristic algorithm based on IPSO and IACO</td>
<td>Delay and energy consumption</td>
<td>Only used for tasks generated from the smart production line</td>
</tr>
<tr>
<td>19</td>
<td>Li et al. (2019) [30]</td>
<td>Cloud-Fog System</td>
<td>Nonlinear programming and STML approach</td>
<td>Energy consumption and delay</td>
<td>Fog nodes are homogeneous</td>
</tr>
<tr>
<td>20</td>
<td>Wu et al. (2021) [9]</td>
<td>Cloud-Fog System</td>
<td>Partition of graph with EDA</td>
<td>Energy consumption, makespan, and lifetime of IoT</td>
<td>Small but real-time data</td>
</tr>
<tr>
<td>21</td>
<td>Abdulredha et al. (2021) [15]</td>
<td>Cloud-Fog System</td>
<td>Evolutionary algorithm</td>
<td>Energy and makespan</td>
<td>Energy consumption is not considered</td>
</tr>
</tbody>
</table>

3. System Model

In this section, the network model, process flow model, problem formulation, earliest finish time model, cost model, and energy model of the system are clearly discussed.

3.1. Network Model

We assume a three-layer system. The first layer is the collection of IoT devices \( \{i_1, i_2, \ldots, i_n\} \) that gather the data and send it to the immediate upper layer for further processing. The middle fog layer contains fog nodes \( \{f_1, f_2, \ldots, f_m\} \) that can compute, store and communicate. The topmost cloud layer consists of high-performance servers \( \{S_1, S_2, \ldots, S_n\} \) with enough storage and processing capability, as shown in Figure 1. Each Server \( S_i \) consists of a set of virtual machines such as \( \{VM_1, VM_2, \ldots, VM_k\} \). Each VM is characterized by a processing rate and memory.

The data can transfer from cloud to fog and also from fog to cloud according to the request at the time of processing. The fog layer contains a \( CF_{manager} \) (cloud fog manager) which gathers all the tasks and resources, and it runs a task scheduler. The task-scheduling scheme makes a decision regarding whether the task will be executed in the fog or cloud node; accordingly, the task is offloaded. The outcome of the task execution is sent back to the \( CF_{manager} \). The \( CF_{manages} \) integrates all the outcomes and sends it to the IoT devices. The EMCS task-scheduling algorithm is installed in \( CF_{manager} \) for performing an optimal task execution schedule. Table 2 show notations used in this paper.
**Figure 1.** System architecture.

**Table 2.** Notations and description.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$i_n$</td>
<td>Represents IoT devices</td>
</tr>
<tr>
<td>2</td>
<td>$f_m$</td>
<td>Represents fog nodes</td>
</tr>
<tr>
<td>3</td>
<td>$S_n$</td>
<td>Represents cloud nodes</td>
</tr>
<tr>
<td>4</td>
<td>$\text{CF}_{\text{manager}}$</td>
<td>Cloud fog manager</td>
</tr>
<tr>
<td>5</td>
<td>${T_1, T_2, T_3, \ldots, T_n}$</td>
<td>Set of tasks</td>
</tr>
<tr>
<td>6</td>
<td>$T_i$</td>
<td>Individual task where $i \in [1, n]$</td>
</tr>
<tr>
<td>7</td>
<td>$T_{i\text{-length}}$</td>
<td>Number of instructions of task $T_i$</td>
</tr>
<tr>
<td>8</td>
<td>${PN_1, PN_2, PN_3, \ldots, PN_m}$</td>
<td>Set of processors</td>
</tr>
<tr>
<td>9</td>
<td>$PN_i$</td>
<td>Individual processor where $i \in [1, m]$</td>
</tr>
<tr>
<td>10</td>
<td>$PN_c$</td>
<td>Numerous cloud nodes</td>
</tr>
<tr>
<td>11</td>
<td>$PN_f$</td>
<td>Numerous fog nodes</td>
</tr>
<tr>
<td>12</td>
<td>$T_{\text{entry}}$</td>
<td>Entry task that does not have predecessors</td>
</tr>
<tr>
<td>13</td>
<td>$T_{\text{exit}}$</td>
<td>Exit task that does not have successors</td>
</tr>
<tr>
<td>14</td>
<td>$PN_{bw}[k]$</td>
<td>Bandwidth of processor $k$</td>
</tr>
<tr>
<td>15</td>
<td>$\text{pred_time}[T_i]$</td>
<td>Time for completion of preceding tasks of $T_i$</td>
</tr>
<tr>
<td>16</td>
<td>$\text{net_In}[T_i, PN_p]$</td>
<td>Total input to processor $PN_p$ for task $T_i$</td>
</tr>
<tr>
<td>17</td>
<td>$\text{EST}_{PN_p}[T_i]$</td>
<td>Task $T_i$’s earliest start time on processor $PN_p$</td>
</tr>
<tr>
<td>18</td>
<td>$\text{EFT}_{PN_p}[T_i]$</td>
<td>Task $T_i$’s earliest finish time on processor $PN_p$</td>
</tr>
</tbody>
</table>
Table 2. Cont.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>( \text{avail}(PN_p) )</td>
<td>Available of processor ( PN_p )</td>
</tr>
<tr>
<td>20</td>
<td>( W_{PN_p}[T_i] )</td>
<td>Task ( T_i )’s execution time on processor ( PN_p )</td>
</tr>
<tr>
<td>21</td>
<td>( E_{PN_p}[T_i] )</td>
<td>Energy required to process task ( T_i ) on processor ( PN_p )</td>
</tr>
<tr>
<td>22</td>
<td>( E_{\text{comm}}[T_i] )</td>
<td>Energy for communication of a task ( T_i ) on processor ( PN_p )</td>
</tr>
<tr>
<td>23</td>
<td>( c_1 )</td>
<td>Processing cost per unit time for fog and cloud</td>
</tr>
<tr>
<td>24</td>
<td>( c_2 )</td>
<td>Communication cost per unit time for fog and cloud</td>
</tr>
<tr>
<td>25</td>
<td>( e_f )</td>
<td>Energy required per unit for a execution of a task</td>
</tr>
<tr>
<td>26</td>
<td>( e_{\text{idle}} )</td>
<td>Energy used when fog is idle</td>
</tr>
<tr>
<td>27</td>
<td>( e_{\text{comm}} )</td>
<td>Energy per unit for transmission of data</td>
</tr>
<tr>
<td>28</td>
<td>( e_c )</td>
<td>Energy per unit for execution of the task in the cloud</td>
</tr>
<tr>
<td>29</td>
<td>( \alpha, \beta, \gamma )</td>
<td>Balance coefficient</td>
</tr>
<tr>
<td>30</td>
<td>( \varepsilon )</td>
<td>Probability of crossover</td>
</tr>
<tr>
<td>31</td>
<td>( \mu )</td>
<td>Probability of mutation</td>
</tr>
</tbody>
</table>

3.2. Communication Model

The communication model followed in our system architecture is similar to [46]. It is a hybrid communication model which used both hierarchical and peer-to-peer (P2P) communication. The communication between the layers is hierarchical, and the communication between the nodes in the fog layer is P2P. The details of the communication mechanism in each layer are discussed below:

- **IoT Device Layer**: In this layer, the devices communicate with the fog nodes which are within the communication range using different wireless communicate protocols.
- **Fog Layer**: In this layer, the fog nodes communicate with each other using device-to-device (D2D) communication. All the fog nodes follow a uniform wireless communication protocol.
- **Cloud layer**: This layer consists of multiple cloud servers which are kept in a centralized data center. The servers are connected through wired links for communication. The communication pattern followed between the cloud and fog layer is wireless in nature.

The process flow model below shows the whole process of the task-scheduling algorithm.

3.3. Process Flow Model

This section discusses the task-scheduling process that will occur in the three-layered architecture. The steps are presented below and shown in Figure 2.

1. **Step 1**: The IoT devices collect the data and send a request to the closest fog node.
2. **Step 2**: The fog node immediately transfers the request to the \( CF_{\text{manager}} \).
3. **Step 3**: To process a request, the request is decomposed into numerous dependent tasks which can be executed in the processors.
4. **Step 4**: Then, the resource usages and the number of instructions of a task are estimated.
5. **Step 5**: The task scheduler runs a scheduling algorithm and finds an optimal scheduling scheme.
6. **Step 6**: As a result of the scheduling algorithm, the task are offloaded to the corresponding cloud and fog nodes.
7. **Step 7**: The task collects the data from preceding tasks which completed their processes either in cloud node or fog node.
8. **Step 8**: Each task is executed in the corresponding node.
9. **Step 9**: After task execution was completed, the results are returned to the \( CF_{\text{manager}} \).
10. Step 10: After the completion of all tasks, the results of a request are combined by the CF manager.
11. Step 11: The aggregated result is bundled as a response and sent to the IoT device through the fog node.

Figure 2. The process flow model on Cloud_fog system.

Using the above models, the assumptions considered in this proposed work are discussed as follows. (a) The IoT devices send the job requests to the nearest fog node. (b) A job request consists of a set of dependent tasks. (c) The dependent tasks of a single job request are scheduled to multiple fog/cloud nodes. (d) The tasks are both memory and CPU intensive. (e) Due to task dependency, whenever there is a requirement of data transfer from one fog node to another, the sender fog node transfers the data to the nearest fog node. (f) The fog nodes have the same communication bandwidth due to there being a uniform communication model as described above. (g) The heterogeneity lies in fog only in terms of processing capacity and energy consumption per unit time for the execution of a task. (h) The servers have different communication bandwidths in the data center.

3.4. Problem Formulation

Consider the request which is decomposed with numerous dependent tasks in CF_manager and represented on a DAG. The dependent task set is $T=\{T_1, T_2, T_3, \ldots, T_n\}$, and each task $T_i$ contains $T_i = \{Tid_i, dsize_i, T_{i-prev}\}$, where $Tid_i$, $dsize_i$, $T_{i-prev}$ represents the task identification number, the required data from other processors, and number of instructions. These tasks are assigned to the DAG, which is represented as $G = (T, C)$ where $T = \{T_1, T_2, \ldots, T_n\}$ $(\forall T_i \in T, i \in [1, n])$, and $C$ represents the edge where $C_{ij} \in C(T_i, T_j \in T)$ shows that after the completion of task $T_j$, task $T_i$ will start. The weight of $C_{ij}$ shows the data size transferred from task $T_j$ to $T_i$ and $dsize_i = \sum_{j} C_{ij}$. The predecessor and the successor of $T_i$ are denoted as $pred(T_i)$ and $succ(T_i)$, respectively. The entry task does not have a predecessor, and the exit task does not have a successor. Thus, the entry task and exit task can be represented as $T_{entry}$ and $T_{exit}$, respectively.

The processors of the cloud-fog network can also be represented with a complete graph $G = (PN, E)$ where $PN = \{PN_1, PN_2, \ldots, PN_m\}$ $(\forall PN_i \in PN, i \in [1, m])$ denotes the processor set and $E_{ij} \in E$ is the link between $PN_i$ and $PN_j$.

Let $PN_c$ represent the numerous cloud nodes and $PN_f$ represent the numerous fog nodes. Then, the total nodes can be represented as $PN = PN_c \cup PN_f$. The process-
ing rate and bandwidth of a processor can be denoted as $PN_{pr}[p]$ and $PN_{bw}[p]$, where $p = \{1, 2, \ldots, m\}$, respectively. All the cloud and fog processors are heterogenous and have different processing rates. The cloud nodes have different bandwidths, but as the fog nodes are far from the cloud and placed in one layer, the bandwidth of the fog node is considered as same. It is assumed that when a task is offloaded to the cloud, it starts its execution immediately without waiting in a queue.

Figure 3 shows both the task and processor graph, where we consider eight dependent tasks $\{T_1, T_2, \ldots, T_8\}$ represented in DAG and four processors $\{PN_1, PN_2, \ldots, PN_4\}$ represented in a complete graph. The tasks $T_2$, $T_3$ and $T_4$ cannot be processed until task $T_1$ is completed its execution. Similarly, task $T_5$ will start execution after the completion of tasks $T_2$ and $T_3$, task $T_6$ will start after the completion of tasks $T_3$ and $T_4$, task $T_7$ will start after the completion of tasks $T_3$, $T_4$ and $T_5$ and finally, $T_8$ will start execution after the completion of tasks $T_6$ and $T_7$. The outcomes of predecessor tasks will be the input of the consequent task. The tasks have priority that the task will start its execution only if the input resources of the task are available. The input resources are not only the outcomes of predecessor tasks but also the data from either cloud or fog nodes. The major objective is the assignment of tasks to the processors in such a way that the system has optimum completion time, reduced cost, and energy consumption. Here, our task scheduler schedules the tasks on the processors considering all these criteria.

**Theorem 1.** The proposed EMCS algorithm is intractable.

**Proof.** The proposed EMCS algorithm considers a task set $T = \{T_1, T_2, T_3, \ldots, T_n\}$. Each $T_i$ has a number of instructions $T_i$-length. $PN = \{PN_1, PN_2, \ldots, PN_m\}$ are the number of processors of cloud-fog network. The processing rate and bandwidth of processor $PN_i$ are $PN_{pr}[i]$ and $PN_{bw}[i]$, respectively. The problem is to assign task $T_i$ to processor $N_j$ after the completion of the execution of a task at processor $N_j$. As the tasks are dependent, after the completion of a task, the successor task will be started. There are $m^n$ possible solutions to map tasks to processors, which is a challenging task. In real time, $m$ and $n$ are very large in terms of thousands. This problem can be reduced to a bin-packing problem where processors $N = \{N_1, N_2, \ldots, N_m\}$ can be mapped to bins $k = \{k_1, k_2, \ldots, k_m\}$ and tasks $T = \{T_1, T_2, T_3, \ldots, T_n\}$ can be mapped to objects $S = \{s_1, s_2, s_3, \ldots, s_n\}$. Here, $T_i$-length is mapped to volume $V_i$ [47]. The bin-packing problem is an NP-hard problem; hence, our proposed EMCS algorithm is also NP-hard. $\square$

![Task Graph](image1.png)

![Processor Graph](image2.png)

**Figure 3.** Task and Processor Graph.
3.5. Earliest Finish Time Model

In this section, we have shown the makespan model of the proposed task-scheduling algorithm. Following Pham et al. [1], the task $T_i$ can start its execution after the completion of all preceding tasks of $T_i$ on processor $PN_p$ and also after obtaining all the input data from the other nodes. The time for completion of preceding tasks can be defined as:

$$\text{pred\_time}[T_i] = \max_{T_j, \text{pred}(T_i), PN_q \in \text{PN}} [\text{EFT}(T_j, PN_q)]$$  \hspace{1cm} (1)

$\text{EFT}(T_j, PN_q)$ is the earliest finish time of task $T_j$ on processor $N_q$. For the entry task, $\text{pred\_time}[T_{\text{entry}}] = 0$. When $T_i$ executes in $PN_p$, the input data of $T_i$ are the outcomes of the predecessor task, and the data from the cloud or fog nodes must be available as well. If $\text{net\_In}[T_i, PN_p]$ is the total input data of task $T_i$ required for processing the task in processor $PN_p$, then it can be represented as:

$$\text{net\_In}[T_i, PN_p] = \begin{cases} \left( dt[T_i, PN_q] + \sum_{T_j, \text{pred}(T_i)} W_{PN_p}[T_j] \times ( \frac{1}{\text{PN}_p[I]} + \frac{1}{\text{PN}_p[I]} ) \right) & \text{if } PN_p \neq PN_q \\ 0 & \text{if } PN_p = PN_q \end{cases}$$  \hspace{1cm} (2)

where $dt[T_i, PN_q]$ is the size of the data, which is required to transfer from processor $PN_q$ for executing task $T_i$. The earliest start time $\text{EST}_{PN_p}[T_i]$ and earliest finish time $\text{EFT}_{PN_p}[T_i]$ of task $T_i$ on processor $PN_p$ are computed as:

$$\text{EST}_{PN_p}[T_i] = \max\{ \text{avail}(PN_p), \text{pred\_time}[T_i] + \text{net\_In} \}$$  \hspace{1cm} (3)

$$\text{EFT}_{PN_p}[T_i] = W_{PN_p}[T_i] + \text{EST}_{PN_p}[T_i]$$  \hspace{1cm} (4)

where $\text{avail}(PN_p)$ is the availability of processor $PN_p$ for executing task $T_i$ and $W_{PN_p}[T_i]$ is the execution duration of task $T_i$ on node $PN_p$, and these are represented as:

$$\text{avail}(PN_p) = \max_{T_j, PN_p} [\text{EFT}_{PN_p}[T_j]]$$  \hspace{1cm} (5)

$$W_{PN_p}[T_i] = \frac{T_i - \text{length}}{PN_p[P]}$$  \hspace{1cm} (6)

The time of completion of all the tasks (makespan) is the earliest finish time of the last task $T_{\text{exit}}$, which can be calculated as the $\text{EFT}_{PN_p}[T_{\text{exit}}]$. So,

$$\text{makespan} = \text{EFT}_{PN_p}[T_{\text{exit}}]$$  \hspace{1cm} (7)

3.6. Cost Model

The cloud charges a monetary cost for processing a task which includes processing cost along with communication cost, while a fog node demands only communication cost [1]. The communication cost for both cloud and fog node is the total cost for transferring data from the predecessor node as well as the cost for transferring data from other cloud or fog nodes. Hence, the monetary cost $\text{Cost}_{PN_p}[T_i]$ for executing task $T_i$ on $PN_p$ can be computed as:

$$\text{Cost}_{PN_p}[T_i] = \begin{cases} c_1 \times W_{PN_p}[T_i] + c_2 \times \sum_{PN_q \in PN_c} \left( dt[T_i, PN_q] + \sum_{T_j, \text{pred}(T_i)} C_{QP} \right) & \text{if } PN_p \in PN_c \\ c_2 \times \sum_{PN_q \in PN_c} \left( dt[T_i, PN_q] + \sum_{T_j, \text{pred}(T_i)} C_{QP} \right) & \text{if } PN_p \in PN_f \end{cases}$$  \hspace{1cm} (8)

where $c_1$ is the processing cost per unit time and $c_2$ is the cost for transferring the data to processor $PN_p$ for executing the task $T_i$.

The total cost can be calculated as:
Total cost = \[ \sum_{T_i \in T, PN_p \in PN} \text{Cost}_{PN_p}[T_i] \] (9)

3.7. Energy Model

If \(T_i\) is assigned to a fog node \(PN_p\), the energy usage in the fog node can be estimated as energy required for executing the task and energy usage when the fog node is being idle [48]. The IoT devices communicate with the fog nodes which are within the communication range of them. The communication among fog nodes follows a D2D pattern where a sender fog node transfers data to the nearest fog node. Due to the close proximity transmission between the fog nodes, the communication energy is negligible. When \(PN_p\) is the cloud node, the energy consumption is considered as the sum of energy usage for execution of a task and the transmission of both task and data. The energy for the transmission of task and data can be represented as communication energy consumption \(E_{\text{comm}}[T_i]\). The energy required to execute a task in a node is as follows:

\[
E_{PN_p}[T_i] = \begin{cases} 
  e_f \times W_{PN_p}[T_i] + e_{idle} & \text{if } PN_p \neq PN_f \\
  e_f \times W_{PN_p}[T_i] + E_{\text{comm}}[T_i] & \text{if } PN_p = PN_c \text{ and } p \neq q
\end{cases}
\] (10)

where

\[
E_{\text{comm}}[T_i] = e_{\text{comm}} \times \left\{ \left( dt[T_i, PN_p] + \sum_{T_j \in PN_q} T_{j\text{pred}(T_i)} C_{qp} \right) + T_i - \text{length} \right\}
\] (11)

where \(e_f\) is the energy per unit for executing a task in a fog node, \(e_{idle}\) is the energy used when a fog node is being idle, \(e_{\text{comm}}\) is the energy per unit for the transmission of data from other nodes to the cloud node, and \(e_c\) is the energy per unit used to process a task in the cloud.

The total energy consumption is calculated as follows:

\[
E = \sum_{PN_p \in PN, T_i \in T} E_{PN_p}[T_i]
\] (12)

**Theorem 2.** Energy consumption of a processor in cloud-fog network is \(E_{PN_p}[T_i]\).

**Proof.** The fog as well as the cloud node energy consumption are considered to prove the lemma. The fog node is used for executing a task or it will be in idle mode. The fog nodes are closed to the IoT devices and the data one fog node is transmitted to the nearest fog node; thus, the transmission energy consumption is negligible. Hence, the amount of energy consumed to process a task \(T_i\) in fog node \(PN_f\) is represented as:

\[
E_{PN_f}[T_i] = e_f \times W_{PN_f}[T_i] + e_{idle}
\] (13)

where \(e_f\) is the energy per unit time for executing a task in the fog node, and \(e_{idle}\) is the energy used when the fog node is idle. In case of a cloud node, the energy consumption for the execution and transmission of data is considered where the idle time is ignored. So, the amount of energy consumed to process a task \(T_i\) in cloud node \(N_c\) is

\[
E_{PN_c}[T_i] = e_c \times W_{PN_c}[T_i] + e_{\text{comm}}^{PN_c}[T_i]
\] (14)

where \(e_c\) is the energy per unit used to process a task in the cloud. If \(N_c\) is the processor to process a task \(T_i\), the energy consumption formulation can be calculated as shown in Equation (10), where \(PN_p = PN_c\).

**Theorem 3.** Transmission energy consumption for a task \(T_i\) to execute in a cloud node \(PN_c\) is \(e_{\text{comm}}[T_i]\).
Proof. If the task $T_i$ will be executed in a cloud node, then the amount of data transferred from IoT device to cloud node is $T_i \cdot \text{length}$. When a task is executed in a cloud node, it collects the link data from the other processors, which is represented in Equation (2) as:

$$dt[T_i, PN_p] + \sum_{T_{ij} \in \text{pred}(T_i)} C_{qp}$$

(15)

where $PN_p$ is the cloud node and $PN_q$ is another processor. Thus, the energy required for the transmission of a task $T_i$ to a cloud node is calculated and represented in Equation (11). □

4. Proposed EMCS Task Scheduling Algorithm

In this work, the EMCS algorithm is proposed which uses the GA evolutionary strategy. The proposed method optimizes the task completion time, cost and energy consumption. EMCS learns from biological reproduction such as selection, crossover, mutation, and finding the best individual. The steps are shown below and the algorithms for these processes are presented in Algorithms 1 and 2.

Algorithm 1 Algorithm for fitness value computation

Input: DAG of task, chromosome, location
Output: fitness value

1: sorted_task $\leftarrow$ topological_sort(task)
2: for $T_i$ in sorted_task do
3: $PN_p \leftarrow$ chromosome[$T_i$]
4: if $T_i == T_{entry}$ then
5: \hspace{1em} $\text{pred\_time}[T_i] \leftarrow 0$
6: else
7: \hspace{1em} Compute $\text{pred\_time}[T_i]$
8: end if
9: for $T_j$ in predecessors($T_i$) do
10: $PN_q \leftarrow$ chromosome[$T_j$]
11: if $T_j == \text{Null}$ then
12: \hspace{1em} $\text{net\_In}[T_i,PN_p] \leftarrow 0$
13: else if $PN_p \neq PN_q$ then
14: \hspace{1em} Compute $\text{net\_In}[T_i,PN_p]$
15: end if
16: end for
17: Compute $\text{EST}[T_i], \text{EFT}[T_i], \text{Cost}[T_i], E[T_i]$
18: end for
19: Set $\alpha, \beta, \gamma$;
20: Compute utility function $F$;
21: Return $F$;

Algorithm 2 Algorithm for best solution selection

Input: Fitness value, chromosome, location
Output: best_solution (chromosome, fitness, pop_location)

1: Set $\text{fitness\_function} \leftarrow \text{Fitness value}$
2: Set population size and num_generation
3: for $n \leftarrow 1$ to num_generation do
4: \hspace{1em} use roulette wheel selection to select chromosomes
5: \hspace{1em} use two-point crossover with probability $\epsilon$
6: \hspace{1em} use mutation with probability $\mu$
7: end for
8: Return $\text{best\_solution}$
4.1. Chromosome Encoding

The chromosome is encoded by assigning a processor to the task. Each individual chromosome is the outcome of the task scheduling. The chromosome can be encoded as n-bits with values ranging from [1:m], where $m$ is the total processors and $n$ is the total tasks. The processors are both cloud and fog nodes. If 1 to $k$ are the cloud nodes, then $k + 1$ to $m$ are the fog nodes. The task is represented as the index of the chromosomes, and the chromosome represents the assigning of the processors (i.e., both cloud and fog nodes) to the tasks. As shown in Figure 4, the number of tasks and processors are 8 and 4. The processors 1 and 2 are the cloud nodes and processors 3 and 4 are the fog nodes, respectively. The number of tasks decides the length of the chromosome, which is 8 bits in this scenario. Thus, the chromosome can be represented as follows:

The chromosome <3, 4, 1, 3, 1, 3, 4, 2> means processor 3 executes tasks 1, 4 and 6, processor 4 executes tasks 2 and 7, processor 1 executes tasks 3 and 5, and similarly, processor 2 executes task 8.

This type of chromosome encoding was chosen for a better performance of genetic operations (crossover and mutation) to create new offspring by inheriting quality genes from parents in the solution search space. The order of execution of the tasks is to be considered because the task can be processed after the completion of the dependent task.

4.2. Population Initialization

All possible chromosomes belong to the solution search space. The population can be initialized by choosing a set of chromosomes randomly. Let $N$ be the size of the population, then $N$ number of chromosomes are randomly selected from the solution search space, ensuring the diversity of the first-generation population. Some operations are applied to the selected individuals of the initial population to create offsprings.

4.3. Fitness Function

The quality of the individual can be determined by the fitness function. The individuals with maximum fitness value are considered as quality outcomes. The fitness value can be determined for each individual by maximizing the utility function $F$ calculated below:

$$F = \alpha \times \frac{\min(EFT_{PN})}{\text{makespan}} + \beta \times \frac{\min(Cost_{PN})}{\text{Total_Cost}} + \gamma \times \frac{\min(E_{PN})}{E}$$

(16)

where $\alpha$, $\beta$, and $\gamma$ are the balanced coefficients such that the sum of them is 1. The values of $\alpha$, $\beta$, and $\gamma$ are considered according to the priority of the makespan, cost or energy consumption. If we give high priority to minimize the makespan, then $\alpha$ is considered with value in range 0.5–1 and other two $\beta$, and $\gamma$ are considered in between 0.5–1. Similarly, if minimization of cost is high priority then $\beta$ value is set in the range of 0.5–1 and $\alpha$ and $\gamma$ are considered in the range of 0.5–1. This way $\gamma$ is also considered in the range of 0.5–1, while
the minimization of energy consumption is considered for higher priority, and the other two are in the range of 0.5–1. Algorithm 1 presents the whole steps for fitness value calculation.

### 4.4. Genetic Operators

Here, the genetic operators are selection, crossover, and mutation. The selection operator selects the populations with higher fitness values. The crossover recombines the genes to inherit the best gene to the next generation. The mutation maintains the diversity from one generation to the next generation.

- **Crossover:** The crossover generates new individuals by crossing the genes of two individuals of the population. In this work, the traditional two-point crossover is applied to the parents for inheriting the quality genes in offspring. The process is shown in Figure 5, where two points are selected randomly and the genes between these two points of the first and second parents are exchanged with each other, while the remaining genes are unchanged to generate new offspring.

  The selection of parents in the crossover process can influence the algorithm. The crossover rate of every individual is $\epsilon$. The parents are selected for crossover by applying a roulette wheel process. From this technique, the best individual with a maximum fitness value has a chance to be the parent, ensuring the offspring have good genes.

- **Selection Process:** The selection process selects a set of individuals on the basis of Darwin’s law of survival and forms a population for the next generation. After the crossover operation, the fitness value of the offspring is calculated. Here, 40% of parents are selected for the next generation, and the remaining population will be formed with offspring of the highest fitness value. Other individuals are discarded in this process. In this selection process, individuals with lesser fitness values are not discarded rapidly, and they are used to explore the search area, so that the population deviance is maintained in each generation.

- **Mutation Operator:** $N$ chromosomes are formed with a new population after the completion of crossover and selection operations. Each individual with mutation rate $\mu$ has participated in a one-point mutation process. A random gene of each individual as shown in Figure 6 is selected, and the value of that gene is changed to a different value ranging from [1:m], which assigns the chosen task to be executed in another processor.

  The mutation overcomes the limitations of crossover by exploring the other areas in the solution space and avoiding local solutions. A set of modifications in GA was introduced in the proposed EMCS algorithm. Those are as follows:

  - **Parent Selection:** 75% of the population are selected as parents for crossover operations. This selection process adopts the roulette wheel technique of selecting parents for mating.
  - **Selection Strategy:** After crossover, 40% of parents are preserved for the next generation, and the remaining population is selected with offspring of larger fitness.

Algorithm 1 presents the fitness value calculation, which is discussed as follows. In this process, the task graph is considered as DAG, so that the topological sort function can be used to find dependencies, which is assigned to the *sorted_task* in step 1. Afterwards, we compute the following steps for each node $T_i$ of the *sorted_task*:

- **In step 3,** assign processor $N_p$ with $\text{chromosome}[T_i]$ (i.e. selected processor for processing task $T_i$);
- **In steps 4–7,** check whether $T_i$ is the entry task or not. If $T_i$ is an entry task, then the set time for completion of the preceding tasks $\text{pred_time}[T_i]$ is 0; otherwise, compute $\text{pred_time}[T_i]$ using Equation (2). For each node $T_i$ of $\text{predecessors}(T_i)$, compute steps 9–15.
- **In step 9,** set $PN_q$ with $\text{chromosome}[T_i]$. 

• In steps 10–15, check $PN_p \neq PN_q$; if true, then compute data transferring time from node $PN_q$ to node $PN_p$ using Equation (2); otherwise, set $net_{In}[T_i, PN_p]$ as 0.

• In step 17, compute $EST[T_i], EFT[T_i], cost[T_i]$, and $E[T_i]$ using Equations (3)–(6), (8), (10) and (11). In step 19, set $a, b, c$ such that $a + b + c = 1$. In step 20, compute the utility function $F$ using Equations (7), (9), (12) and (13). Finally, return $F$ in step 21.

The time complexity of Algorithm 1 is the total time required to process the algorithm. The DAG $G = (T, C)$ is the set of task as the input. The total time complexity is calculated as follows.

- Step 1 requires $O(T + C)$ times for topological sort.
- The time required for executing step 2 to step 18 is $O(T * \text{indeg}(T))$, where \text{indeg}() is the number of incoming edges to a node $T$.
- The time complexity of step 19 is 1.
- Similarly, the complexity of each step 20 and step 21 is 1.

The overall time complexity of the algorithm is $O(T + C) + O(T * \text{indeg}(T)) + 1 + 1 + 1 = O(C + T * \text{indeg}(T))$.

Algorithm 2 presents the best solution selection. In step 1, a fitness function is assigned with a fitness value computed in Algorithm 1. Then, we set the population size and number of generations in step 2. Afterwards, repeat steps 4–6 for each generation. In these steps, perform roulette wheel selection, two-point crossover with probability, and mutation with probability for each generation. At last, return the best solution. The time complexity analysis of Algorithm 2 is shown as follows.

- Step 1 and step 2 has a time complexity of 1.
- Step 3 to step 7 has a time complexity of $O(\text{num\_generation}(\text{population\_size} + \text{population\_size} * \text{chromosome\_size} + \text{population\_size} * \text{chromosome\_size}))$.
- Step 8 has a time complexity of 1.

So, the overall time complexity of the algorithm is $1 + 1 + O(\text{num\_generation}(\text{population\_size} + \text{population\_size} * \text{chromosome\_size} + \text{population\_size} * \text{chromosome\_size}) + 1 = O(\text{num\_generation} * \text{population\_size} * \text{chromosome\_size})$. 
The whole process of Algorithms 1 and 2 for EMCS is shown in the flowchart in Figure 7.

Figure 7. Flowchart for EMCS.
5. Performance Evaluation

This section presents the experimental setup and evaluation of our proposed EMCS algorithm by comparing it with some other recent algorithms.

5.1. Experimental Setup

The experiments are conducted using the Python environment. The software or hardware taken for this simulation is shown in Table 3. The assumptions considered for implementing the algorithm are explained below.

Both cloud and fog nodes with different processing rates, monetary costs, and energy rates are considered. Each node has its own processing rate (represented by MIPS (million instructions per second)) as well as communication cost. The fog nodes have a minimal processing speed, while the cloud nodes have servers and virtual machines with high processing speed, which demands processing cost on the usage of the processors. The bandwidth of the fog nodes is higher than the cloud nodes. As all the fog nodes are placed in a middle layer, the bandwidth is the same for all fog nodes, and as per the heterogeneous behavior of the cloud, the bandwidth is varied. Similarly, the energy cost is more in cloud nodes compared to the fog nodes. The idle fog nodes have also consumed energy. Both cloud and fog nodes consume energy at the time of computation, and additional energy for communication is also consumed by the cloud. The cost is represented as Grid Dollar (G$), which is a currency unit used in this simulation.

Table 3. Hardware/Software Specification.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Hardware/Software</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System</td>
<td>Intel® Core ™ i5-4590 CPU @ 3.30 GHz</td>
</tr>
<tr>
<td>2</td>
<td>Memory (RAM)</td>
<td>4 GB</td>
</tr>
<tr>
<td>3</td>
<td>Operating System</td>
<td>Windows 8.1 Pro</td>
</tr>
</tbody>
</table>

In our model, we are considering a DAG for task graphs with varying sizes in MIPS. The task graph is randomly generated with different sizes from 20 to 100 with a complete graph of processors having a different set of fog and cloud nodes. The input and output files of a task have size from 10 to 50 MB. The performance of EMCS is evaluated by taking the parameters shown in Table 4.

Table 4. Simulation parameters and values setup.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tasks</td>
<td>[20, 100]</td>
</tr>
<tr>
<td>2</td>
<td>Cloud nodes (PN_c)</td>
<td>[5, 45]</td>
</tr>
<tr>
<td>3</td>
<td>Fog nodes (PN_f)</td>
<td>[5, 45]</td>
</tr>
<tr>
<td>4</td>
<td>Processing rate of cloud (PN_pr)</td>
<td>[250, 500] MIPS</td>
</tr>
<tr>
<td>5</td>
<td>Processing rate of fog (PN_pr)</td>
<td>[10, 500] MIPS</td>
</tr>
<tr>
<td>6</td>
<td>Bandwidth of cloud (PN_bw)</td>
<td>10, 100, 512, 1024 Mbps</td>
</tr>
<tr>
<td>7</td>
<td>Bandwidth of fog (PN_fbw)</td>
<td>1024 Mbps</td>
</tr>
<tr>
<td>8</td>
<td>Amount of communication data (dt/T, PN_p)</td>
<td>[10, 50] MB</td>
</tr>
<tr>
<td>9</td>
<td>Processing cost per time unit for cloud (c_{1})</td>
<td>0.5 G$/s</td>
</tr>
<tr>
<td>10</td>
<td>Processing cost per time unit for fog (c_{2})</td>
<td>[0.1, 0.5] G$/s</td>
</tr>
<tr>
<td>11</td>
<td>Communication cost per time unit for cloud (c_{3})</td>
<td>0.7 G$/s</td>
</tr>
<tr>
<td>12</td>
<td>Communication cost per time unit for fog (c_{2})</td>
<td>[0.3, 0.7] G$/s</td>
</tr>
<tr>
<td>13</td>
<td>Number of instructions (T_{(n-length)})</td>
<td>[100, 500] 10^9 Instructions</td>
</tr>
<tr>
<td>14</td>
<td>Energy per unit time for execution of the task in fog (c_{1})</td>
<td>[1, 5] w</td>
</tr>
<tr>
<td>15</td>
<td>Energy used when fog node is idle (c_{idle})</td>
<td>0.05 w</td>
</tr>
<tr>
<td>16</td>
<td>Energy per unit for transmission of data (E_{comm})</td>
<td>[0.1, 2] w</td>
</tr>
<tr>
<td>17</td>
<td>Energy per unit for execution of task cloud (c_{2})</td>
<td>[5, 10] w</td>
</tr>
<tr>
<td>18</td>
<td>Balance coefficient (α, β, γ)</td>
<td>[0, 1] and α + β + γ = 1</td>
</tr>
</tbody>
</table>
The evolutionary model is used in our algorithm. The values considered for the parameters of the GA are shown in Table 5. The two-point crossover and single-point random mutation are considered.

For evaluating the performance of our model EMCS, the model is compared with other models such as CMaS [1], HEFT [13], BLA [14] and ETS [15]. CMaS is used to check the makespan and monetary cost, whereas HEFT is used only for makespan check. BLA is another evolutionary algorithm that checks the cost for execution and use of memory for allocating the tasks in the fog nodes. ETS is also an evolutionary algorithm that checks the makespan and cost for execution. For comparing the makespan, cost and energy consumption of all the models, the models are implemented with given parameter values. The values for parameters of BLA and ETS are shown in Table 5.

Table 5. Parameters of GA, BLA and ETS.

<table>
<thead>
<tr>
<th>Name</th>
<th>GA</th>
<th>ETS</th>
<th>BLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of chromosome</td>
<td>[20, 100]</td>
<td>[20, 100]</td>
<td>[20, 100]</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Probability of crossover ($\epsilon$)</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Probability of mutation ($\mu$)</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
<td>100</td>
<td>Q = 1, D = 30, W = 69</td>
</tr>
</tbody>
</table>

5.2. Results and Discussion

Several experiments were conducted with different scenarios. When 20 fog nodes and 5 cloud nodes were considered with tasks varied from 20 to 100, Figure 8 shows the computation of makespan for different numbers of tasks. The EMCS model performed 67.1% better than CMaS, 58.79% better than HEFT, 54.68% better than BLA and 47.81% better than ETS.

Figure 8. Comparing makespan of EMCS with other methods.

Regarding the cost of usage of cloud and fog (shown in Figure 9), it has been observed that HEFT has the highest cost, while EMCS has the lowest cost and CMaS, BLA and ETS have the average. HEFT focuses on the processing time of the task without considering other costs and energy. So, it has the highest cost. While compared with CMaS, EMCS can save 62.4% of the average cost; EMCS can save 26.41% of the average cost of BLA and 6.5% of average cost of ETS.
It is observed that the energy consumption is higher with an increase of task numbers in Figure 10. It is also observed that HEFT has the highest energy consumption compared to the others. EMCS saves 11.55% of the average energy of CMaS, 4.75% of the average energy of BLA and 3.19% of the average energy of ETS.

Next, the impact of cloud nodes in different algorithms (EMCS, CMaS, HEFT, BLA and ETS) with an increase in the nodes from 5 to 25 is evaluated. In this experiment, the number of processing nodes is fixed at 50 with varying cloud nodes from 5 to 25 generating different simulation results. Each cloud node has different parameter values. The results are shown in Figures 11–13 which indicate that the makespan is decreased with the increase of cloud nodes in the network. As the cloud demands more processing cost and communication cost, the monetary cost is increased with the increasing of cloud nodes, and similarly, energy consumption is also increased with the increasing of cloud nodes. It is found that with the increase of clouds, the makespan of our algorithm EMCS is 64.21% better than that of CMaS, 30.41% better than HEFT, 11.62% better than BLA, and 7% better than ETS. It is
observed in Figure 11 that there is an increase of makespan from 20 to 25 clouds in our proposed model, while the makespan in other algorithms is decreased.

In the case of cost comparison on increasing clouds from 5 to 25, the cost is increased in all the algorithms, while our model EMCS saves more cost than CMaS, HEFT, BLA, and ETS. It is observed in Figure 12 that our model is slightly better than BLA and ETS with increasing clouds from 5 to 15, and it is also better in the change of clouds from 15 to 25. Overall, the EMCS algorithm can save 21.81% of the average cost compared with BLA and 7.25% of the average cost compared with ETS.

Similarly, energy consumption is also increased with increasing the clouds from 5 to 25 in all the algorithms. Our model saves more energy than CMaS, HEFT, BLA and ETS. It is noticeable in Figure 13 that our model can save 8.24% of the average compared with BLA and 3.78% of the average compared with ETS, but the energy consumption of our model is almost the same with BLA and ETS when the number of clouds is set to 15.

Figure 11. Impact of cloud nodes on makespan.

Figure 12. Impact of cloud nodes on cost.
Similarly, when the fog nodes are changed from 5 to 25 with a fixed number of processors at 50, the impact on makespan, cost, and energy consumption is shown in Figures 14–16. The makespan is reduced with an increase in fog nodes which is noticeable in Figure 14. It is also found that our model is 6.94% better than BLA and 3.43% better than ETS.

The cost is varied when the fog nodes are changed from 5 to 25, as shown in Figure 15. It is noticeable that the cost is increased when the fog nodes are 15 and the cost of our model is almost the same as the cost of BLA and ETS at 20 fog nodes. The average cost of our model is 14.09% less than the average cost of BLA and 7.64% less than the average cost of ETS.

Similarly, following Figure 16, it is found that the energy is reduced with the increase of fog nodes from 5 to 20, but it is increased when there are 25 fog nodes. In addition,
the energy consumption is similar at fog node 20 for our model, BLA and ETS. When the average energy consumption is considered, then our model consumes 31.75% less energy than BLA and 15.97% less energy than ETS.

So, it is observed from all the above results that a balance between cloud and fog nodes is required to give a better performance in terms of makespan, cost, and energy consumption. In addition, as the EMCS algorithm gives optimum makespan, cost, and energy consumption, it is helpful for reducing global warming.
6. Conclusions

In this work, we adopt the combined cloud-fog architecture to schedule the set of dependent tasks. The evolutionary strategy-based EMCS algorithm is proposed, and its performance is evaluated with other methods such as CMaS [1], HEFT [13], BLA [14] and ETS [15]. By observing all the results, it is found that EMCS is better than CMaS by 67.1%, HEFT by 58.79%, BLA by 54.68% and ETS by 47.81% in terms of the average makespan with high performance on cost and energy consumption. EMCS is better than CMaS by 62.4%, BLA by 26.41% and ETS by 6.5% on average cost, and EMCS has reduced energy consumption compared with CMaS by 11.55%, BLA by 4.75% and ETS by 3.19%. The increase of cloud and fog nodes has a heavy impact on the network. Overall, our model is better than other algorithms with the change of cloud nodes from 5 to 25. It is observed that the makespan of EMCS is 11.62% better than the makespan of BLA and 7% better than the makespan of ETS. Similarly, the cost of EMCS is also reduced to 21.81% of the cost of BLA and 7.25% of the cost of ETS, and the energy consumption of EMCS is 8.24% less than BLA and 3.78% less than ETS. However, the makespan, cost, and energy consumption are almost the same in EMCS, BLA and ETS when the cloud node is set to 15. Similarly, with varying fog nodes from 5 to 25, comparing EMCS, BLA and ETS, the average estimation of EMCS is 6.94% better than BLA and 3.43% better than ETS in the case of makespan. Meanwhile, the average cost of EMCS is 14.09% less than BLA and 7.64% less than ETS, and the average energy consumption of EMCS is 32.75% less than BLA and 15.97% less than ETS. While considering the fog nodes at 20, the cost and energy consumption are almost the same in both EMCS, BLA and ETS.

So, from the overall result of the algorithm, it can be concluded that this algorithm will be a better solution for industry to use as it shows less energy, cost, and makespan that can reduce the carbon emissions and decrease the global warming. In the future, we will try to implement more meta-heuristic algorithms to solve the scheduling problems. We will also expand our research by focusing on the priority, deadline constraint of the task and use of virtual machines in task scheduling.

**Author Contributions:** Conceptualization, R.S., S.K.B. and N.P.; methodology, R.S., S.K.B., N.P. and K.S.S.; software, R.S., K.S.S.; validation and formal analysis, R.S., S.K.B., N.P. and M.B.; supervision and project administration: S.K.B., N.P., K.S.S., M.B. and S.C.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT Korea (No.2021R1F1A1049393) and in part by Institute for Information & Communication Technology Promotion (IITP) Korea (No. 2019-0-01816).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data and materials are available on request.

**Acknowledgments:** The author thanks BPUT, Rourkela, India for providing facilities to conduct the research work. The author also thanks the Kempe Foundation of Sweden and Wallenberg AI, Autonomous Systems and Software Program (WASP).

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

**References**


