GA-IRACE: Genetic Algorithm-Based Improved Resource Aware Cost-Efficient Scheduler for Cloud Fog Computing Environment

Jawad Usman Arshed, Masroor Ahmed, Tufail Muhammad, Mehtab Afzal, Muhammad Arif, and Banchigize Bazezew

1Department of Computer Science, Capital University of Science & Technology, Islamabad, Pakistan
2Department of Computer Science, Air University, Islamabad, Islamabad, Islamabad, Islamabad, Pakistan
3Department of Computer Science and Information Technology, University of Lahore, Lahore, Pakistan
4Department of Information Technology at Wollo University, Kombolcha Institute of Technology (KIOT), Ethiopia

Correspondence should be addressed to Banchigize Bazezew; banchigizeb@kiot.edu.et

Received 28 April 2022; Accepted 17 June 2022; Published 8 July 2022

Copyright © 2022 Jawad Usman Arshed et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The ever-growing number of Internet of Things (IoT) devices increases the amount of data produced on daily basis. To handle such a massive amount of data, cloud computing provides storage, processing, and analytical services. Besides this, real-time applications, i.e., online gaming, smart traffic management, and smart healthcare, cannot tolerate the high latency and bandwidth consumption. The fog computing paradigm brings the cloud services closer to the network edge to provide quality of service (QoS) to such applications. However, efficient task scheduling becomes critical for improving the performance due to the heterogeneous nature, resource-constrained, and distributed environment of fog devices. With an efficient task scheduling algorithm, the response time to application requests can be reduced along with bandwidth and cloud resource costs. This paper presents a genetic algorithm-based solution to find an efficient scheduling approach for mapping application modules in a cloud fog computing environment. Our proposed solution is based on the execution time as a fitness function to determine an efficient module scheduling on the available fog devices. The proposed approach has been evaluated and compared against baseline algorithms in terms of execution time, monetary cost, and bandwidth. Comprehensive simulation results show that the proposed approach offers a better scheduling strategy than the existing scheduler.

1. Introduction

The broad concept of the Internet of Things (IoT) has allowed the creation of new communities that links humans and machines. IoT is the connection of daily life objects such as computers, farms, animals, factories, and vehicles via heterogeneous networks [1–3]. These objects have become “smart” due to the IoT, allowing them to sense, process, and communicate effectively over a network to perform useful tasks without requiring end-user interaction [4, 5]. With the rapid development of IoT, the number of devices connecting to the network is rapidly increasing. According to a report from Manyika et al. [6], by 2025, the IoT is expected to have a theoretical impact of $11 trillion per year, reflecting 11 percent of the global economy. According to a press release in 2015 [7], 13.4 billion connected devices to the Internet already outnumbered humans on the earth. A new study from Juniper research predicts 38.5 billion connected IoT devices by 2020, a 285% increase from 2015 [8], which is a staggering figure. In Cisco’s annual Internet report, the number of devices and connections connected to the Internet is expected to increase from 18.4 billion in 2018 to 30 billion by 2023 [9]. As IoT devices grow, a massive amount of data will generate, also known as big data. Several data centers have been configured as a cloud to accommodate the enormous amount of big data generated by IoT devices.

In the tradition of cloud computing, data service subscribers are projected to be able to benefit from efficient
and flexible services [10]. There are two main actors in traditional cloud computing, i.e., cloud service providers (CSP) and cloud users or customers [11]. Users lease the CSP’s resources on a pay-per-use basis. The CSP acquires cloud resources, such as storage and processing, and makes those resources available to cloud users [12]. Customer tasks are received by the cloud task manager in the form of cloudlets and are passed to the task scheduler [13]. These traditional clouds are expected to provide data service lets and are passed to the task scheduler [13]. The transferring of data between the end-user and the cloud requires high bandwidth for processing, causing delays [14]. Moreover, if some geospatially distributed and time-sensitive applications, such as smart healthcare monitoring, virtual reality, and smart traffic surveillance, send their requests for processing to the remote cloud may impact the quality of service due to the delay caused by the resource bottlenecks and the bandwidth constraints [15–17]. Many approaches, such as mobile computing, edge computing, and fog computing, have been proposed to overcome this limitation by bringing computation and storage services closer to the end-user application [18, 19].

Recently, fog computing has received the most attention among the various proposed approaches. Fog computing is a new concept introduced by CISCO [20]. It is an extension of cloud computing which provides the services like storage and processing to the IoT users at the edge of the network to reduce network congestion and latency [21]. In contrast to specialized computing facilities such as data centers, fog computing uses a vast number of locally distributed fog servers, which can be smartphones, intelligent gateways, switches, routers, access points, and cellular base stations with limited aptitudes in terms of storage and processing power [22–24]. The sense-process-actuate model (SPAM) is commonly used in fog computing to detect and collect data through sensors and then send the collected data to fog devices for processing. Fog nodes can send data from the fog layer to the cloud, where it is stored and processed for long-term analytics. The data transferred from the fog layer to the cloud for processing reduces the task’s execution time, but it may increase the bandwidth and the monetary cost when using the cloud resources [13]. On the other hand, real-time applications require a faster response than delay-tolerant applications. As a result, these applications must compete for limited resources. Because of the resource-constrained and latency-sensitive nature of the fog applications, resource management in fog computing is one of the challenging tasks [25]. When jobs are scheduled efficiently, they can produce timely and accurate responses, which are crucial for smart systems. For example, in a smart healthcare system, the medical condition of a patient needs to be reported as soon as possible to save the patient’s life. Similarly, in an innovative home application, the security surveillance application must report immediately if any suspicious activity happens [16]. Therefore, some efficient job scheduling algorithm is needed to make optimal use of these resource-constrained and heterogeneous fog devices [26]. In this work, we propose a GA-based optimized policy for placing application modules in a cloud fog environment, which minimizes the execution time of applications with optimized usage of bandwidth and the application’s monetary cost. The contributions of the paper are summarized as follows:

This paper presents an efficient scheduling technique for mapping application modules in a cloud fog computing environment. The proposed solution is leveraged by a genetic algorithm to evolve the solution using the applications’ execution time. Initially, the proposed approach takes into account application and fog device properties to encode the chromosome-based representation of the problem. Then, genetic algorithm-based mutation and crossover operators were employed to evolve the solutions using multiple variations of the parameters. The evolved solution has been utilized to schedule the application modules, using the iFogSim simulator to validate the results. Experimental results are also compared with baseline and state-of-the-art algorithms for three application corpus.

1.1. Architectural Model. Fog computing is a new computing paradigm that enables storage, processing, and communication at the edge of the network. Fog computing can use cloud resources to process and store large-scale applications [27–29]. Figure 1 shows the fog architecture, in collaboration with end-users and cloud, developing a three-layer hierarchical, bidirectional design [30]. In the cloud fog architecture, the top layer is made up of the cloud layer, with fog devices acting as an intermediate layer in the middle. The bottommost layer or end layer comprises end-users with sensors or actuators.

1.2. End Devices/End Layer/User Layer. The user layer is the closest to the end-user, with many heterogeneous and mobile devices. These devices consist of IoT devices like sensors (temperature sensors, heartbeat sensors, GPS sensors, etc.), smart cards, mobile phones, cameras, connected cars, and laptops.

Data from these external devices is converted into signals and sent to fog nodes for processing. End devices are frequently battery-powered and relatively restricted in CPU and memory, resulting in limited battery life. End devices often generate data that must be processed and stored elsewhere because the end device’s processing or storage capacity is insufficient. In some cases, end devices require data from a different source. As a result, end devices are frequently connected to a network. However, uninterrupted connectivity may not always be possible.

1.3. Fog Layer. The computational resources present at the fog layer near the network’s edge are referred to as fog nodes. Typically, the fog layer is comprised of heterogeneous fog devices with a limited amount of computing, storing, and networking capabilities, such as switches, proxy servers, routers, and cellular base stations. Fog devices can be resource-poor (little memory, computation power, etc.) like access points and routers [31], or they can be resource-rich like cloudlets [32]. Existing network equipment, such as gateways, routers, and switches, can operate as fog nodes if they have adequate free resources.
1.4. Cloud Layer. The cloud layer is the topmost layer of the cloud fog architecture with multiple high-speed servers having massive storage capacity, which provides various application and storage services, as shown in Figure 1. The cloud layer of the cloud fog environment resources is used for applications that need a lot of processing power and storage space.

1.5. Scheduling in Fog Computing. Scheduling refers to arranging application modules onto computing resources in order to maximize utilization, for example, of CPU, memory, and bandwidth. The structure of the cloud fog environment allows the scheduling of application modules according to module requirements with the following combinations [33–35].

(1) The modules are offloaded from end devices to the fog and cloud layer when all three layers are involved according to the modules’ requirements

(2) Only the fog layer is involved by offloading the modules from end devices to the fog nodes

Figure 2 shows an example of a scheduling problem in cloud fog computing. We consider five users (U₁, U₂, U₃, U₄, and U₅) at the user layer who want to execute their applications at the fog layer. Each user has two applications, either A₁ or A₂, comprising three modules (M₁, M₂, and M₄). Based on the defined scheduling policies, the scheduler decides the mapping of modules in the cloud fog environment. As shown in Figure 2, modules M₁ and M₂ of A₁ for U₁ and M₁ of A₂ for U₂ are selected for execution at the fog layer, while M₃ of A₁ for U₁ and M₃ of A₂ for U₄ are mapped at the cloud layer for execution. Similarly, a fog device in the range of U₃ has successfully mapped all application modules (M₁, M₂, and M₃). For U₃ and U₅, modules M₂ of A₁ and M₃ of A₁ for U₄ and M₁ and M₃ of A₂ for U₅ are mapped at the fog layer, while M₁ of A₁ for U₄ and M₃ of A₂ for U₅ are placed at cloud. The abbreviations used in the paper are listed in Table 1.

The rest of the paper is organized as follows: Section 2 covers the related work. The problem formulation is described in Section 3. The design and implementation of the problem are covered in Section 4. Section 5 discusses simulation results, and Section 6 concludes the paper with future work.

2. Related Work

This section covers the existing resource allocation techniques for improving performance in the cloud fog environment. Most of the resource optimizations deal with the job allocation from fog to the cloud, whereas some deal with the scheduling of jobs on the resource-constrained fog devices. The purpose of these allocations and scheduling is to provide a better quality of service by reducing the execution time, minimizing network usage, and minimizing the monetary cost, etc. An overview of some of these job allocation approaches is as follows.

In [22], Pham and Huh proposed a task scheduling technique for dependent tasks on each other. The authors used the directed acyclic graph (DAG) to represent tasks. These dependent tasks need data communication with each other. A priority is created by traversing the DAG and assigning the prioritized jobs to the fog nodes. The assignment of modules is based on the fog devices’ resource limitations. The proposed technique has a low resource utilization rate and increases the application’s execution time. In [36], Zeng et al. proposed another scheduling and job image placement algorithm in which the job images are placed on a storage server. The computations are performed by embedded clients and fog nodes with shared storage servers. Jobs are scheduled to be completed as quickly as possible while providing a better user experience. The jobs are scheduled to achieve a better user experience with minimum completion time. In [37], Ni et al. proposed a dynamic resource allocation strategy to improve the user’s quality of service with better resource utilization. The resource allocation is based on the credibility of fog nodes by using priced timed Petri nets (PTPNs) and the job’s completion time. To optimize energy consumption, a heuristic-based resource allocation algorithm based on a “penalty and reward policy” was proposed by Pooranian et al. [38]. The resource allocation problem was considered a “bin packing penalty-aware” problem in which fog servers are considered bins. In contrast, the VMs are packs that must be served according to time and frequency constraints. In the event of a penalty, the server remains unallocated for some iteration. DT Hoang [39] designed a hierarchically based job scheduling algorithm for cloud fog architecture to improve performance and achieve low latency. The author proposed a fog-based region architecture for job assignments in fog and cloud regions. In [40], Sun et al. proposed a two-level resource scheduling algorithm in which the fog layer is divided into clusters. Each resource is assigned to a different fog cluster or fog node within a given cluster. Among fog nodes in the same cluster, the authors used the theory of nondominated sorting genetic algorithms II for resource scheduling for multiobjective optimization. The authors claim that they were able to
reduce delays and improve job execution solidity. In [41] Taneja and Davy proposed a resource-aware scheduler for the effective utilization of resources at fog layer. For the device selection, the authors used the recursive binary search algorithm. The proposed approach does not consider the cost of cloud services in scheduling the modules from fog to the cloud layer.

The goal of fog computing is to serve a wide range of applications, either latency-sensitive or delay-tolerant. For these latency-sensitive applications, Liu et al. [26] combine the minimum completion time of a job with the mining association rules to reduce the execution time and average waiting time. These rules generated from the priori algorithm are used to schedule jobs for the fog devices. Gupta et al. [21] proposed another job scheduling algorithm for latency-critical applications and showed that the modules placed on the edge are more effective than those placed in the cloud. The authors used the EEG tractor game (latency-critical application) and computed energy consumption, network usage, and loop delay using a first-come, first-serve (FCFS) algorithm in the cloud fog computing environment. Bittencourt et al. [42] analyzed different types of latency-tolerant and latency-sensitive applications to discuss resource allocation in fog computing. The authors used the concurrent, FCFS, and delay-priority strategy for application module placement. The concurrent strategy has a resource contention issue when the number of modules to be scheduled increases. The resource contention was

**Figure 2:** Example on scheduling of application modules in cloud fog computing environment.
eliminated by the FCFS strategy by offloading the modules from fog to the cloud layer, causing a delay for latency-sensitive applications and increasing network usage. To deal with latency-sensitive applications, the authors used a delay priority strategy, in which applications are moved from the fog layer to the cloud layer based on their delay sensitivity. However, when the number of application modules grows, the proposed approach cannot keep up, resulting in increased network usage, execution delay, and increased monetary costs of using cloud resources. Choudhari et al. [43] proposed a prioritized job scheduling algorithm to reduce overall response time and cost. When a job arrives, its priority is determined by its deadline. A job’s computed priority is used to place it in the fog layer. Each fog layer has multiple micro data centers and fog nodes that can communicate with one another. The job is moved to the cloud layer if all of the data centers in a fog layer are saturated. In the proposed approach, the primary goal is to balance execution time and memory allocation. Other important goals, such as network usage and energy consumption, are overlooked. Bitam et al. [44] used bioinspired bees swarm optimization algorithm to schedule tasks in a cloud fog environment. The prime objective of the proposed algorithm was to find the best trade-off between memory allocation and CPU execution time. Gai and Qiu [45] used reinforcement learning to improve the quality of experience (QoE) and optimize resource allocation. For cost mapping tables and resource allocation optimization, the authors proposed two reinforcement algorithms. A cluster of fog nodes was set up for each user request to achieve a specific goal, such as delay. Zhang et al. [46] proposed a double deep Q-learning model (DDQ) in edge computing to reduce energy consumption. The DDQ method was used to calculate the Q value of each dynamic voltage and frequency scaling algorithm (DVFS). Instead of using the sigmoid function as the activation function, rectified linear units were used to avoid gradient vanishing. Rahbari and Nickray [47] used the greedy knapsack-based scheduling (GKS) algorithm to implement the job scheduling in the cloud fog environment. Comparing the proposed algorithm to concurrent, FCFS, and delay priority scheduling, the authors claim it is more energy-efficient and lowers execution costs. However, the authors ignored network usage, which is an important parameter. For real-time job scheduling, Mai et al. [48] used deep reinforcement learning in their work. The authors used a reinforcement learning approach and evolutionary strategies to train a neural network. Because fog computing is still in its early stages, only a few algorithms with efficient resource utilization have been considered. Compared to the proposed approach, the presented methods do not preserve execution time, network usage, and cloud resources, thereby increasing the monetary cost. In the cloud fog environment, we use the GA to optimize the scheduling of application modules for efficient resource utilization at the fog layer and reduce the execution time of applications along with the bandwidth consumption and the monetary cost.

### 3. Problem Formulation

Fog computing is the concept of utilizing fog node services for end devices that help them to overcome capacity constraints. When the device’s generated data exceeds its capacity, it can connect to a fog node, pass data to the fog node, and offload data to the fog node for processing. Since the network transfers between the fog node and the end devices have minimal latency and the fog nodes have the relatively high processing power, it allows the entire process to be quick, enabling real-time interactions. However, the offloading of tasks from the fog to the cloud layer and the efficient scheduling at the fog layer becomes critical for latency-sensitive applications like traffic control, augmented reality, and gaming. In case when all application modules are accommodated at the fog layer, it may reduce the monetary cost and bandwidth of the application modules. Still, it may
increase the application’s execution time due to the limited power of the fog devices not being affordable for time-sensitive applications.

On the other hand, when data is transferred from the fog to the cloud layer for processing, it may reduce the task’s execution time, increasing the bandwidth and the monetary cost for applications [13, 49]. Therefore, we are essentially concerned with formulating the goal of scheduling in a cloud fog environment that can reduce the applications execution time while optimizing the bandwidth and monetary cost. The entire set of resources consists of fog devices (FDs) and cloud resources (CLOUD). The resource capacity of a network node can be defined with three fundamental parameters: CPU, RAM, and bandwidth. Similarly, the application modules ($M_i$) may be computationally intensive or bandwidth-intensive, depending on their computations and bandwidth requirements [21, 50–52]. Therefore, if in a module set $M$, the application module is represented with $M_i$ and its computation requirements with CR, it can be defined as follows:

Let $C = \{CR_{M1}, CR_{M2}, CR_{M3}, \ldots, CR_{Mn}\}$ be the computation requirement of each module in the module set $M$. 

---

**Figure 3: Flowchart of GA-IRACE.**
**Algorithm 1: GA-IRACE scheduler.**

```plaintext
Input:  
i) List of Modules (LM) with their computation requirement (MI) and bandwidth requirement (BW)  
ii) List of Fog devices (LFd) with their Processing Power (MIPS)  
Output: modulemap[[][]]: Mapping of modules LM on LF_d  

1 MODULEMAP (LF_d, LM)  
   begin  
2     Find Computation Threshold (CT) for CR  
3     Find Bandwidth Threshold (BT) for BR  
4     for i=0 to LM. size do  
5       if LM_i (cpu) > CT and LM_i (network) > BT  
6         then  
7           MAP (LM_i → CLOUD)  
8         else  
9           LM(R) ← LM_i  
10        end  
11      end  
12      modulemap[[][]] = RACE-GA (LM(R), LF_d)  
13      return modulemap  
14   end
```

**Algorithm 2: GA-IRACE.**

```plaintext
Input:  
i) List of Modules (LM) with their computation requirement (MI) and bandwidth requirement (MB)  
ii) List of Fog devices (LF_d) with their Processing Power (MIPS)  
iii) Size of population (P)  
iv) Maximum Generation (MaxG)  
Output: modulemap[[][]]: Mapping of modules LM on LF_d  

1 GA (LM, LF_d, P, MaxG)  
   begin  
2     POP = InitialPopulation (LM, LF_d, P)  
3     for each Chromosomes in POP  
4       Calculate fitness  
5     end  
6     for generation=1 to Max_G do  
7       new_Elite_fitness = Calculate_Fitness (POP, P)  
8       if (elite_fitness( ) > new_elite.fitness( ) ) then  
9         Elite_POP = Elite_ind  
10       else  
11         new_POP = Elite_ind  
12       end  
13       for i=1 to Elite_POP  
14         POP.insert (Elite_POP)  
15       end  
16       for j=1 to new_POP  
17         POP.insert (new_POP)  
18       end  
19       Elite_POP = Top 10% of POP  
20       New_POP = POP - Elite_POP  
21       Parents = Selection (New_POP, P)  
22       Children = Crossover (Parents, P)  
23       Mutated_Pop = Mutation (Children, P)  
24       POP = Mutated_POP + Elite_POP  
25     end  
26     return Elite_POP  
27   end
```
Let $B_w = \{B_{w1}, B_{w2}, B_{w3}, \ldots, B_{wn}\}$ be the bandwidth requirement of each module in the module set $M$.

The computation threshold $CT$ and a bandwidth threshold $BT$ are represented in Equations (1) and (2).

$$CT = \text{median}(c),$$  

$$BT = \text{median}(BW).$$  

$$\forall (M_i \in M) \left\{ \begin{array}{l} \frac{M_{i(CR)}}{M_{i(BR)}} > CT \\ \frac{M_{i(CR)}}{M_{i(BR)}} > BT \end{array} \right\} \Rightarrow M_i \rightarrow \text{CLOUD}.$$

$$\text{otherwise} \quad \text{REM}(M) \leftarrow M_i.$$  

Because the modules may be computationally or bandwidth-intensive, we begin our module mapping by categorizing them, as shown in Equation (3). The modules with more extensive computation requirements than the computation threshold are directly sent towards the cloud for required operation, whereas the remaining modules are stored in the list for optimized scheduling. Equation (4) shows the actual start time of the module $M_j$ on the fog device $FD_i$.

$$\text{ST}_{ij} = \left\{ \begin{array}{l} 0 \left( \text{pred}(M_j = 0) \right), \\ \max_{m \in \text{previous}(m_j)} \max_{k \in \text{pred}(m_j)} \left\{ ET_{jk}(k \rightarrow i) \right\} \\ \text{if} \left( \text{pred}(M_j) \neq 0 \right). \end{array} \right.$$  

In Equation (4) $ST_{ij}$ shows the start time of the module. The modules executed before $M_j$ on the fog device $FD_i$ are denoted by previous($M_j$). The module $M_j$ cannot start its execution until its predecessor completes its execution. If more than one module needs to be executed simultaneously on the same device, i.e., $FD_i$, we select a task according to the order in which modules are scheduled. The execution time of a module $M_i$ can be calculated as shown in the following equation.

$$ET_{ij} = \frac{M_{i(J)}}{\text{MIPS}(FD_i)} \forall M_i \in F, \forall FD_i \in F.$$  

The estimated finish time $EFT$ of a module $M_j$ scheduled on a fog device $FD_i$ can be calculated in the following...
Let $P_n$ be the processing node, either fog or the cloud, where the modules are scheduled for processing. The cost $\text{Cost}(M_i, P_n)$ is the monetary cost of executing the application modules $M_i$ when scheduled on $P_n$ is computed as shown in the following equation.

$$\text{Cost}(M_i, P_n) = \sum_{p_n \in N_{\text{cloud}}} C_{\text{comm}}^{(M_i, p_n)} \text{if } p_n \in N_{\text{cloud}},$$

$$\sum_{p_n \in N_{\text{fog}}} C_{\text{comm}}^{(M_i, p_n)} \text{if } p_n \in N_{\text{fog}},$$

The processing cost in Equation (7) is calculated as given below in the following equation:

$$C_{\text{proc}}^{(M_i, p_n)} = c_t \cdot A(M_i, P_n),$$
where $c_1$ is the processing cost per time unit of the application module on the node $p_n$. The communication cost for sending outgoing data from the cloud node $p_n$ can be calculated using Equation (9) in which $c_2$ shows the price per unit of data.

$$C_{\text{comm}}^{(M,p_n)} = c_2 \times \left( d_1^n + \sum_{M \in \text{exec}(p_n)} c_i \right). \quad (9)$$

The overall cost of an application can be computed as given below in the following equation:

$$\text{Cost}(A) = \sum_{i=1}^{n} c(M_i). \quad (10)$$

Similarly, the bandwidth requirement will also be the sum of all the application modules placed on cloud and fog devices and is computed as shown in the following equation.

$$\text{Bw}_{\forall M_i \in A}^{(M,p_n)} = \left\{ \begin{array}{ll}
\sum_{k=1}^{n} p_k \times \text{BR}_{M_k} & \text{if } M_i \rightarrow \text{cloud}, \\
\text{BR}_{M_i} & \text{if } M_i \rightarrow \text{fog}.
\end{array} \right. \quad (11)$$

4. Design and Implementation

The genetic algorithm (GA) was created by Holland in 1975 [53] based on the theory of natural selection, according to which the fittest individuals are chosen to reproduce the next generation. The GAs begins with a primary population of chromosomes. A new population is formed in each generation using genetic operators such as mutation, crossover, and selection.

The mutation operator searches the entire search space to recover lost genetic information and maintain population diversity. The crossover is the process of selecting two individuals and producing a new child from them. The crossover operator’s main task is to collect and combine both parents’ positive characteristics to make more suitable offspring. In evolutionary algorithms, the selection operator is analogous to natural selection, which results in the survival of the fittest individuals. During evolution, the best solutions’ genetic information has a higher chance of being passed down to future generations through reproduction.

The main goal of the selection operators is to survive the fittest individuals by increasing the number of appropriate solutions in the next generation’s population. It increases the chances of choosing the best current-generation solutions as parents and producing offspring. As a result, a new population emerges, establishing the next generation. The process is repeated until all of the specified stopping criteria have been met. The basic flow of the genetic algorithm for our proposed scheduling model is shown in Figure 3. The scheduling is based on the computation and bandwidth requirements of application modules. After finding the threshold for application modules w.r.t. the compute and bandwidth intensity, the more dense modules are sent directly to the cloud for processing. The remaining application modules are sent to the GA for the optimized placement over the fog layer. In the next step, the parameters are initialized as the number of populations and the chromosomes in each population. We have used populations from 50 to 100 for our experiments, each having 50 chromosomes. The parameter setting is defined in Table 1. For the initial population, we have used random numbers. The fitness function is applied to the initial population to find the best chromosomes for the next population. After applying the fitness function to the randomly generated initial population, the fitness value is checked in the next phase to select the best chromosomes. We used the 20% ratio to choose the best chromosomes from the initial population to become part of the new population. The remaining chromosomes are sent to the next phase to apply crossover and mutation operator, as shown in Figure 3 (selection of chromosomes for the new population). The 20% selected the best chromosomes from the initial population, and the remaining 80% chromosomes, after applying crossover and mutation, created the new population. The process is repeated until found the converged solution. Algorithm 1 describes the main scheduling algorithm. After finding the threshold on lines 2 and 3, the application modules with more than computation and bandwidth threshold are directly passed to the cloud for the processing from lines 4 to 7. The remaining modules are then passed to the genetic algorithm to define the optimized scheduling policy lines 9 to 12. The performance of meta-heuristics is greatly influenced by the initial solutions; i.e., if the GA starts with a good initial population, it will almost certainly result in better final solutions.

The diversity of an algorithm loses when a rule generates all the initial solutions as the produced solutions may be close to a particular solution space. In literature, most initial populations are randomly generated; therefore, we also decided to start the initial population randomly.

The elitism technique is used to preserve the best individual generated during evolution (steps 7-10). After a certain number of iterations ($\text{Max}_{t}$), step 27 returns the final
best individuals as the application module scheduling solution.

### 4.1. Chromosome Encoding

The first step in implementing GA is to create a chromosome-like scheme for the problem information. In our scheduling problem, we have to reduce the execution time while optimizing the monetary cost and the use of network bandwidth. The entire set of resources consists of fog devices (FDs) and cloud resources (CLOUD). The resource capacity of a network node can be defined with three fundamental parameters: CPU, RAM, and bandwidth. Similarly, the application modules $A_iM_j$ may be intensive w.r.t. computation and bandwidth requirements [21, 50–52]. The application modules and the available fog devices are arranged in the form of two-dimensional matrices. Each column in the matrix shows a resource, and the modules are assigned to these resources row-wise, as shown in Figure 4.

The assignments of modules are under the scheme generated through random numbers. One random number generated is associated with one application module showing the application module’s position under the resource; e.g., the application module having random number 1 will be assigned first. Similarly, the module associated with random number 2 will be given to the next resource and in a similar fashion; all the modules are set to the available resources according to the assigned random numbers. One complete assignment of application modules to the fog devices according to the produced random numbers shows the completion of one chromosome as illustrated in Figure 4.

### 4.2. Parent Selection Strategy

A parent selection is a method of selecting the chromosomes for crossing from among the current population. In our scenario, the fitness value for all the chromosomes in the population is calculated after the initial population is created by randomly assigning resources to application modules, as shown in Figure 3. Based on the calculated fitness value, the top 10 chromosomes are selected for the following population, while the remaining chromosomes are forwarded to the next phase as parents for the process of crossover and mutation.

### 4.3. Fitness Function

In the evolutionary algorithm, the fitness function determines the quality of the solution during the evolution process. In this paper, the fitness of a schedule is related to minimizing the application’s execution time, as shown in Equation (6). The fitness function is shown in the

---

**Table 2: Experimental setup.**

<table>
<thead>
<tr>
<th>Application type</th>
<th>Compute intensive (application-1)</th>
<th>Bandwidth intensive (application-2)</th>
<th>Mixed (compute+bandwidth intensive) (application-3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of modules</td>
<td>70</td>
<td>80</td>
<td>100</td>
</tr>
<tr>
<td>MI in modules</td>
<td>100-900</td>
<td>20-200</td>
<td>40-500</td>
</tr>
<tr>
<td>No. of fog devices</td>
<td>4, 8, 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation power of fog devices (MIPS)</td>
<td>200 to 3000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation power of cloud (MIPS)</td>
<td>42800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3: GA-IRACE parameter settings.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of populations</td>
<td>100</td>
</tr>
<tr>
<td>Population size</td>
<td>50</td>
</tr>
<tr>
<td>Reproduction operator</td>
<td>Crossover and mutation</td>
</tr>
<tr>
<td>Crossover type</td>
<td>Single point</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>1%, 3%, 5%</td>
</tr>
<tr>
<td>Stopping criteria</td>
<td>100 iterations</td>
</tr>
</tbody>
</table>

---
following equation given below.

\[
\text{Fitness} = \bigvee_{M_{i} \in A} \sum_{i=1}^{M} \text{EFT}(M_{i}), \tag{12}
\]

where \(M_{i}\) shows the application modules to be scheduled on the available fog devices. The EFT in Equation (12) shows the estimated finish time of the application module and is computed using Equation (6). The main objective of this optimization is to minimize the fitness value.

4.4. Crossover Operator. By swapping the information contained in the existing parents, the crossover operator is aimed at developing more compatible chromosomes, also known as offspring. In our case, we have the chromosome comprising of the application modules and their placement on the available resources. In our chromosome arrangements, we have two options for the crossover operation.

1. Replace the fog resources of parent-1 \((P_{1})\) with the fog resources of parent-2 \((P_{2})\) without changing the sequence of application modules.

2. Arrange the application modules and resources in the chromosome in a 2D array format so that application modules are placed column-wise. Each subsequent row shows the assignment of fog devices for these application modules.

We have selected option 2 for our crossover method in which each row shows one chromosome, as shown in Figure 5. With crossover points \(CP_{1}\) and \(CP_{2}\), the two chromosomes are designated as parent-1 \((P_{1})\) and parent-2 \((P_{2})\). The crosspoints are selected as the central point of the \(P_{1}\)
Scenario-1

Figure 9: Bandwidth consumption with 4 FDs.

Scenario-1

Figure 10: Monetary cost with 4 FDs.

Scenario-2

Figure 11: Execution time with 8 FDs.
Scenario-2

**Figure 12**: Bandwidth consumption with 8 FDs.

**Scenario-2**

**Figure 13**: Monetary cost with 8 FDs.

**Scenario-3**

**Figure 14**: Execution time with 10 FDs.
and $P_2$. All the genes between CP$_1$ from $P_1$ are copied at CP$_2$ of $P_2$. Similarly, the genes from CP$_2$ of $P_2$ are copied to the CP$_1$ of $P_1$. The two new chromosomes, each carrying some genetic information from both parents, are created and are added to the new population. The process is repeated to create new chromosomes for the subsequent population. Algorithm 3 describes the crossover operation for our proposed methodology in which, in step 1, the two chromosomes are passed to the algorithm for crossover operation.

We have used the one-point crossover method in which the central point is designated as the "crossover point" at line 2. From lines 4 to 7, the genes to the left of the crossover point of $C_1$ are copied to the new chromosome (new$_C_1$) along with the resource configuration, i.e., the allocated fog devices for processing. Similarly, from lines 9 to 13, the genes to the right of the $C_2$ crossover point, as well as the resource configuration, i.e., the assigned fog devices for processing, are copied to the new chromosome (new$_C_2$).

4.5. Mutation. As a genetic operator, mutation provides genetic diversity in a population of genetic algorithm chromosomes from generation to generation. We have used the single-point mutation method, one of the common methods for implementing the mutation operator. Algorithm 4 shows the mutation method for our proposed approach in which, at line 4, the random numbers are generated for each gene in the chromosome. In line 5, the generated random number is checked with the defined mutation rate. From lines 6-8, the chromosomes are mutated if the generated random number and the mutation rate match with each other. Figure 6 also reflects the mutation process in which the random number generated for $g_2$ in $C_1$ is selected for swapping with $g_4$ in $C_1$. Similarly, the following gene selected for swapping is $A_1M_2$ and $A_4M_4$. The process is repeated for all genes in each chromosome.

5. Simulation and Result Discussion

For experiments, we have used the system with an Intel Core i7-8550U Quad-core processor (clocked at 1.9 GHz) and 8 GB of the main memory. The GA is implemented in C# to generate the optimized scheduling policy for the
application modules. The scheduled policy obtained from GA is then passed to the fog system simulated in iFogSim [21].

We have used the iFogSim, an open-source framework for testing the efficiency of the proposed policy by modeling and analyzing the cloud fog environment. The iFogSim is based on CloudSim [54], a widely-used cloud simulator used to simulate many different computing paradigms [55]. The configuration details for the simulation environment are shown in Table 2. The population size used for the genetic algorithm was 100, with chromosome size remaining 50 in each population. Similarly, the crossover rate was set to 5 percent, with the mutation rate at 3 percent [56], as shown in Table 3. To verify the performance of GA-IRACE, we have used three different scenarios with network topologies of 4, 8, and 10 in a cloud fog computing environment.

Figure 7 shows a graphical representation of one of these topologies generated using iFogSim.

We have used the workload of three different applications, A1, A2, and A3, with varying scales as bandwidth-intensive, compute-intensive, and mixed (bandwidth and compute-intensive). The number of modules in the application varies from 70 to 100, with the modules’ MI ranges from 100 to 900 MI, as shown in Table 2.

5.1. Discussion of Experimental Results. In the experiments, the proposed scheduling algorithm GA-IRACE was compared with the conventional cloud placement, RACE [13], and the other baseline algorithms, i.e., Taneja and Davy [41] and the random solution in terms of execution time, monetary cost, and bandwidth. The simulation findings (Figures 8–16) show that the proposed placement technique has a tremendously positive effect on execution time, the bandwidth, and the monetary cost of applications in all three network topologies tested.

Figure 8 shows that the execution time of A1 for GA-IRACE is significantly improved compared to using the conventional method for cloud placement. It can be seen in Figure 8 that the execution time of A1 improves for the GA-IRACE as compared to the traditional cloud placement, RACE [13], Taneja and Davy [41], and the random placement. For A2, the bandwidth-intensive application RACE [13] has a better execution time than Taneja and Davy [41], traditional cloud placement, and the random solution; however, the GA-IRACE has a better execution time than all baseline algorithms. Similarly, for A3, the mixed application for computation and bandwidth-intensive, the GA-IRACE is faster than the baseline algorithm of Taneja and Davy [41], the RACE [13], traditional cloud placement, and the random solution.

Figure 9 shows the bandwidth consumption of three applications, i.e., A1, A2, and A3. It can be seen that all three applications have maximum bandwidth consumption when considering only the traditional cloud placement and the random solution. Considering the fog devices with the cloud resources reduces the bandwidth consumption for three applications by RACE [13] and Taneja and Davy [41]; however, the GA-IRACE has the better results from all three base algorithms. Figure 9 depicts that the bandwidth consumption reduces for all three applications when using the GA-IRACE scheduling algorithm compared to the other baseline algorithms, i.e., traditional cloud placement, RACE [13], and Taneja and Davy [41].

The increase in usage of cloud resources increases the monetary cost of application, as shown in Figure 10, when all the application modules are scheduled on the cloud. However, the efficient scheduling of the proposed algorithm GA-IRACE of application modules reduces the monetary cost compared to the random solution and the baseline algorithms.

Increasing the number of fog devices improves the execution time of the applications along with changes in bandwidth consumption and the monetary cost, as seen from Figures 11–13. Figure 11 shows the efficient scheduling of our proposed algorithm for the three applications, i.e., A1, A2, and A3, respectively. It can be seen in Figure 11 that the GA-IRACE improves the execution time from the RACE algorithm for all three applications.

Although the traditional cloud placement has the same execution time as scenario-1, the RACE [13], Taneja and Davy [41], and the random solution improve the execution time as compared to scenario-1.

Similarly, our proposed scheduling improves the RACE algorithm from execution time and has better execution time as compared to the baseline algorithms. Figure 12 illustrates the positive impact of GA-IRACE on the bandwidth consumption when we have eight fog devices. By adding more fog nodes, bandwidth consumption is improved compared to traditional cloud placement that uses the cloud as a platform for modules, requiring more bandwidth from the end-user to the cloud.

Although the performance of RACE [13] is better than Taneja and Davy [41] and the random solution, however, GA-IRACE outperforms for all three applications with lower bandwidth consumption. Figure 13 shows the monetary cost for the three applications with eight fog devices. Although Taneja and Davy [41] improve the monetary cost, however, the RACE [13] has improved the result with the better utilization of resources at the fog layer. TCP has a higher monetary cost for applications since it relies on cloud resources and excludes fog resources. From all the three approaches, our proposed approach, i.e., GA-IRACE, better absorbs the increasing number of fog devices and has a lower monetary cost for all three applications. Figures 14–16 illustrate how the underlying network topology affects the execution time of three applications along with the bandwidth and monetary costs. Figure 14 depicts that in the case of A1, GA-IRACE improves the execution time compared to the traditional cloud placement, random solution, RACE [13], and the baseline algorithm, i.e., Taneja and Davy [41]. The execution time of A2 (bandwidth-intensive application) for traditional cloud placement is comparatively less than RACE [13] and Taneja and Davy [41] due to the placement of application modules in the cloud, high in processing power.

Despite this, the GA-IRACE performs better than traditional cloud placement, RACE [13], and Taneja and Davy [41] for all three applications, i.e., A1, A2, and A3. A graph of the bandwidth consumption is presented in Figure 15.
for the three applications with sixteen fog devices. It can be seen that for the three applications \((A_1, A_2,\) and \(A_3)\), the traditional cloud placement consumes the most bandwidth, as the modules are located within the cloud. For RACE \([13]\) and Taneja and Davy \([41]\), the shifting of modules from the fog to the cloud layer under the defined scheduling policy reduces the bandwidth consumption than traditional cloud placement; however, scheduling with the optimized values of GA, the GA-IRACE has better results as less bandwidth consumption for all three applications \(A_1, A_2,\) and \(A_3\), respectively.

In Figure 16, the cost per application for using cloud resources in a network topology of 10 fog devices is shown. As shown in the figure, by adding more resources to the fog layer, cloud resources are used with less monetary cost, as there are adequate resources at the fog layer. However, the efficient scheduling with maximum resource utilization also affects the performance as it can be seen that the optimized scheduling with GA-IRACE has better results than traditional cloud placement, random solution, RACE \([13]\), and Taneja and Davy \([41]\).

6. Conclusion and Future Work

Advances in communication technologies have anticipated the ubiquitous use of Internet of Things (IoT) devices in the future. Consequently, proper utilization of cloud computing for real-time applications regarding latency, cost, and bandwidth consumption remains inevitable. The fog computing paradigm mitigates this limitation because it provides a convenient and efficient way to provide a processing facility close to the edge of the user network. However, scheduling jobs on fog computing becomes a challenging task due to the resource constraint of edge devices. This paper presents a genetic algorithm-based solution to optimize the scheduling of modules between the fog and cloud layers in the three-tier cloud fog computing architecture. The results show that the proposed GA-based solution improves the performance by 15–40% in terms of execution time, cost, and bandwidth consumption. In addition, compared to basic algorithms, the optimized scheduling is faster, consumes less bandwidth, and has less financial costs for using the cloud resources. This work can be extended in several directions in the future. We utilize only the execution time metric as a fitness function to evolve the scheduling of the application modules. However, the monetary cost, bandwidth requirements, and/or latency can be studied as a fitness function. Another direction for the future is developing a multiobjective fitness function to find the trade-offs between these factors in the context of cloud fog computing scheduling. Another exciting aspect of the study would be to evaluate the proposed approach for large corpus applications.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References


