Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 4290382, 11 pages
https://doi.org/10.1155/2022/4290382

Research Article

Genetic Algorithm-Based Task Scheduling in Cloud Computing Using MapReduce Framework

Zhihao Peng,[1] **Poria Pirozmand**,[2] **Masoumeh Motevalli**,[3] **and Ali Esmaeili**[4]

[1]*EIT Data Science and Communication College, Zhejiang Yuexiu University, Shaoxing, China*
[2]*Hebei Key Laboratory of Machine Learning and Computational Intelligence, Hebei University, Baoding, China*
[3]*Department of Computer Engineering, Karaj Branch Islamic Azad University, Karaj, Iran*
[4]*Faculty of Technical Engineering, Islamic Azad University, North Tehran Branch, Tehran, Iran*

Correspondence should be addressed to Masoumeh Motevalli; masoumehmotevalli@yahoo.com

Received 25 August 2022; Accepted 16 September 2022; Published 30 September 2022

Academic Editor: Ali Asghar Rahmani Hosseinabadi

Task scheduling is an essential component of any distributed system because it routes tasks to appropriate resources for execution, such as grids, clouds, and peer-to-peer networks. Common scheduling algorithms include downsides, such as high temporal complexity, non-simultaneous processing of input tasks, and longer program execution times. Exploration-based scheduling algorithms prioritize tasks using a variety of methods, resulting in long execution times on heterogeneous distributed computing systems. As a result, task prioritization becomes a bottleneck in such systems. It is appropriate to prioritize tasks with the shortest execution time using faster algorithms. The genetic algorithm (GA) is one of the evolutionary approaches used to solve complex problems quickly. This paper proposes a parallel GA with a MapReduce architecture for scheduling jobs on cloud computing with various priority queues. The fundamental aim of this study is to employ a MapReduce architecture to minimize the total execution time of the task scheduling process in the cloud computing environment. The proposed method accomplishes task scheduling in two stages: first, the GA was used in conjunction with heuristic techniques to assign tasks to processors, and then the GA was used in conjunction with the MapReduce framework to assign jobs to processors. In our experiments, we consider heterogeneous resources that differ in their ability to execute various tasks, as well as running a job on different resources with varying execution durations. The results show that the proposed method outperforms other algorithms such as particle swarm optimization, whale optimization algorithm, moth-flame optimization, and intelligent water drops.

1. Introduction

One kind ofdistributed computing system is heterogeneous, in which several processors are used to do the same work [1]. In cloud computing, task scheduling is separated into a series of lower priority tasks for processing [2]. These sub-tasks exhibit precedence constraints in the sense that the outcome of previous tasks is required before executing the current sub-task [3]. It may lower the task completion time by breaking a computer work into sub-tasks and executing them on numerous processors. As a result, the goal of this study is to arrange sub-tasks on a variety of available processors in order to minimize task completion time without breaching precedence requirements [4].

The development of task scheduling algorithms that distribute sub-tasks of a program to processors is a challenge in cloud systems. Despite recent improvements in the area of task scheduling, this problem remains a significant challenge in heterogeneous computing settings[5, 6].

Reduced execution time of the scheduling algorithm remains an essential problem due to the rise in the quantity of data in the cloud environment. As a result, several techniques for reducing task completion time by parallelizing sub-tasks and honoring their precedence connections have been proposed. A directed acyclic graph, which consists of vertices that represent tasks and directed edges that indicate task dependency, is often used to depict precedence relationships. It is acceptable and expressive to deliver

programs with a vast and varied volume using directed acyclic graphs [7, 8]. On the other hand, the Hadoop system enables task execution in a shared data center [9]. MapReduce is an excellent Hadoop technique for processing massive data in cloud computing that runs instructions and programs in parallel utilizing processors or computers [10]. In a distributed setting, this programming paradigm enables the creation of parallel and distributed processing on a huge number of data [11]. This framework is provided with several methodologies and applications in various disciplines, such as algorithms with high time complexity in huge data, which decreases algorithm execution time by enhancing parallel processing [12]. As a result, this article discusses how to schedule priority jobs on directed acyclic networks using the genetic algorithm (GA) and the MapReduce framework. The proposed method accomplishes task scheduling in two stages: the GA was used in conjunction with heuristic techniques to allocate tasks to processors in the first stage, and the GA was used in conjunction with the MapReduce framework to assign jobs to processors in the second stage.

In essence, the following are the article's objectives:

(i) Creating a new combination of GA with MapReduce framework to assign jobs to processors.

(ii) Using the MapReduce architecture for scheduling in heterogeneous cloud environments.

(iii) Reducing the overall program execution time.

(iv) Accelerating the convergence of solutions and avoiding premature convergence.

The remainder of the paper is structured as follows. Section 2 examines various essential strategies for work scheduling in cloud computing. The overall structure and recommended approach are presented in Section 3 utilizing the MapReduce framework. Section 4 analyzes the acquired findings and compares the proposed approach to other algorithms. Section 5 concludes with the conclusion and future works.

2. Literature Review

This section discusses approaches for task scheduling as well as the benefits and drawbacks of different methods. This study also describes the fundamental concept of the suggested scheduling approach after examining the current methods. The main issue of this paper is the multiobjective aspect of job scheduling in cloud computing. Among the different scheduling concerns, one of the NP-hard optimization problems is scheduling the machine with parallel processors.

Two list-based scheduling methods, heterogeneous earliest-finish-time (HEFT) and critical-path-on-a-processor (CPOP), have been introduced in [13] in order to concurrently meet the two objectives of high efficiency and quick scheduling. To lower the quickest start time, the HEFT algorithm picks a task with the upward rank at each stage and assigns it to processors using the insertion-based technique. The CPOP algorithm, on the other hand, prioritizes by aggregating upward and downward rank into tasks. Another difference between the two algorithms is in the assignment of the processor to the tasks, with the CPOP method executing the critical path tasks on the processors, hence minimizing the execution time of all critical path tasks. Shabestari et al. [14] proposed an ant colony method (ACO)-based scheduling algorithm. The goal of this scheduler is to reduce the workflow time of the group of tasks while also reducing the maximum completion time of the complete job. This technique enables more agile work while decreasing completion time. The results of the evaluation also revealed that the ACO algorithm outperforms the random and best-effort methods. Liu [15] created an ant colony algorithm-based adaptive work scheduling system for cloud computing. Pheromones may now be updated dynamically in response to changes in the environment to the polymorphic ant colony algorithm to increase the algorithm's convergence speed and successfully prevent the formation of local optimum solutions. Based on the tasks supplied by users, the enhanced algorithm tries to develop a distribution plan with a quicker execution time, reduced cost, and balanced load rate. Their experiment results demonstrate that the modified adaptive ant colony algorithm can rapidly discover the ideal solution to the cloud computing resource scheduling issue, shorten job completion time, minimize execution cost, and keep the overall cloud system center load balanced.

For handling multi-objective task scheduling in cloud computing, Manikandan et al. [16] offered a unique hybrid whale optimization algorithm-based MBA algorithm. The multi-objective behavior of the hybrid WOA-based MBA algorithm reduces the makespan by optimizing resource consumption. By using the mutation operator from the bees algorithm, the output of the random double adaptive whale optimization algorithm (RDWOA) may be made to be of higher quality. The performance of the algorithm is analyzed and compared to that of other algorithms that make use of the CloudSim tool kit platform for various criteria like the amount of time it takes to finish, the amount of money it costs to compute, and so on. The study reveals that the suggested method is superior to other algorithms in terms of performance metrics such as computational cost, execution time, makespan, and resource consumption. These metrics were evaluated in relation to the findings obtained. To schedule a set of user tasks on a set of VMs, Imenea et al. [17] presented a third-generation multi-objective optimization method for the first time to our knowledge called non-dominated sorting genetic algorithm (NSGA-III). The method is used in the cloud to minimize the runtime (TE), power consumption (CE), and cost. Manikandan et al. [18] suggested a system that uses fuzzy C-means clustering hybrid algorithms for job scheduling and fish swarm optimization for optimal resource allocation to minimize cost, energy, and resource use. Shukri et al. [19] suggested an enhanced version of the multi-verse optimizer (EMVO) as a better job scheduler in this field. In terms of minimizing makespan time and boosting resource usage, the findings reveal that EMVO outperforms both MVO and PSO algorithms.

The task scheduling and resource provisioning challenges have become an appealing paradigm in the cloud sector, owing to the rising demand for services offered by VMs that are organized by actual servers owned by cloud service providers' data centers (CSPs). In [20], the authors suggested a novel model based on a multi-agent system for task scheduling and resource provisioning that uses deep reinforcement learning to reduce energy costs. A quantile regression deep Q network method delivers the best policy and long-term choices. A series of tests demonstrate the efficacy of the suggested scheduling strategy as well as the performance of the proposed job allocation mechanism. In [21], the ant particle swarm genetic algorithm is presented as a mix of PSO-ACO-GA for work scheduling on cloud computing VMs. Here, PSO and GA will iterate to determine the job based on fitness value, and ACO will distribute the work to particular VMs. This article improves on characteristics such as CPU usage, execution time, and makespan.

Mangalampalli et al. [22] used the cat swarm optimization (CSO) algorithm to solve task scheduling in cloud computing. The presented algorithm considers lifetime, migration time, energy consumption, and the total cost of power in data centers. Tasks are scheduled in the proposed algorithm by calculating task priorities at the task level and calculating VM priorities at the VM level to plan the appropriate mapping of tasks on virtual machines.

Amer et al. [23] used the modified Harris hawks optimizer (HHO) algorithm to solve the multi-objective scheduling problem. To improve the quality of the standard HHO algorithm's discovery phase, a scientific smart method called elite opposition-based learning is used. Furthermore, the minimum completion time algorithm is used as the first phase to obtain a determined initial solution rather than a random solution at each run time, avoiding local optimization and improving the quality of service in terms of program length minimization. Furthermore, the minimum completion time algorithm is used as the first phase to obtain a determined initial solution rather than a random solution at each run time, avoiding local optimization and improving service quality by minimizing program length to cover implementation costs while maintaining service quality.

3. The Proposed Algorithm

As mentioned earlier, the suggested approach accomplishes task scheduling in two stages: in the first stage, the GA was used in conjunction with heuristic techniques to allocate tasks to processors, and in the second stage, the GA was used in conjunction with the MapReduce framework to assign jobs to processors. The problem variables are described initially in this part, followed by procedures to construct GA for combining cloud services.

3.1. Initial Definitions. The suggested approach employs a collection of heterogeneous P processors linked by a high-speed network. The issue of task scheduling is mapped on a directed graph $G(V, E)$ in this method. So, V represents the program's sub-tasks, and E represents the graph edges that specify the dependencies between the sub-tasks. Each sub-task in the graph can only be executed on a single processor. The amount of communication cost between the two sub-tasks is represented by edge values, and the average computational cost of each sub-task in the processors is measured on each node. The example described in this study is shown in Figure 1.

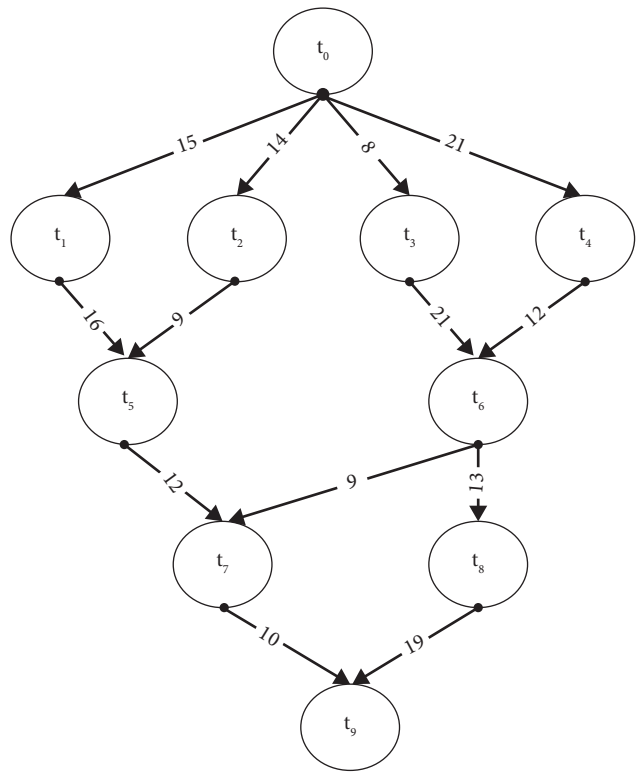

Figure 1: A directed graph that does not rotate and has ten sub-tasks [5].

Table 1 lists the symbols used in this paper's equations and algorithms.

The communication cost between two linked sub-tasks is zero if they are scheduled on the same processor. The start and end nodes in the graph represent the beginning and end of the program, respectively. The utilization model of the proposed method is unrelated, which implies that one processor may perform some activities in less time and others in more time, as shown in Table 2. Furthermore, all previous sub-tasks must be scheduled and completed before commencing to execute a sub-task.

Following the creation of the initial population, each chromosome is evaluated and ranked based on total execution time; this ranking is referred to as the suitability value of each chromosome. In the ranking, the HEFT processor allocation algorithm is employed. The fusion process is then initiated by choosing and merging several chromosomes. The parallelism procedure is performed on the chromosomes after the fusion step. The proposed method is executed until the termination condition is satisfied. In the next portion of this research, the details of each step of the suggested algorithm and accompanying pseudocodes will be discussed.

Table 1: Symbols used in the proposed method.

Symbol	Description
t_i	i^{th} sub-task in a graph
p_k	K^{ih} processor in the system
E	Set of edges in a graph
T	The following set of tasks in the graph
P	Set of processors in the system
Tentry	Sub-task input in a graph
Texit	Sub-output task in a graph
Succ(t_i)	The set of delays below the task t_i
Pred(t_i)	A set of sub-tasks t_i
$\overline{W(t_i)}$	The average computational cost below task t_i
$W(t_i, P_k)$	Computational cost below task t_i on the processor
$C(t_i, t_j)$	The amount of communication cost between the tasks of t_i and t_j
$\text{rank}_b(t_i)$	Upward rank of sub-task t_i
$\text{rank}_t(t_i)$	Downward ranking of sub-task t_i
$\text{Rank}_{b+t}(t_i)$	Add a task to the sub-task t_i in the rank
$\text{EST}(t_i, P_k)$	The earliest start time sub-task t_i on a processor
$\text{EFT}(t_i, P_k)$	The earliest end time sub-task t_i on a processor
$\text{AST}(t_i, Pk)$	Actual start time sub-task t_i on a processor
$\text{AFT}(t_i, Pk)$	Actual end time sub-task t_i on a processor
Avail {k}	Idle time and availability of pk processor
Pop	Population
PopSize	Population size
ChSize	Chromosome size

Table 2: An example of sub-task computational expenses on processors.

Task	P_0	P_1	P_2	$\overline{\omega}$
t_0	8	9	10	9
t_1	9	10	11	10
t_2	10	6	11	9
t_3	12	8	16	12
t_4	25	18	17	20
t_5	13	10	16	13
t_6	7	12	17	12
t_7	17	10	12	13
t_8	12	8	13	11
t_9	13	10	13	12

3.2. Initial Population Quantification. Making a chromosome through encoding is the first stage in populating the population. The suggested algorithm's chromosomal structure contains a gene that provides a solution to the task scheduling issue. A chromosome's structure is a permutation of natural numbers from *0* to *n1*, which shows the order of priority of jobs in a non-rotating directed graph. Furthermore, the topological arrangement of tasks on the chromosome must be valid. The proposed method places the starting job at the very first position on the chromosome and the job at the very end of the chromosome, known as the end node. The additional jobs are located on the chromosome in locations that do not contradict task precedence. All subtasks in the graph must be planned for scheduling to be feasible.

The *PopSize* symbol represents the proposed algorithm's starting population size, which is four times the number of jobs in the network ($4n$). The algorithm's population size remains fixed until the conclusion of the algorithm. This means the population size does not fluctuate throughout the creation of new generations. In the proposed algorithm, three exploratory ranking policies called upward rank (equation (1)), downward rank (equation (2)), and a combination of these two methods with level ranking (equation (3)) have been used to quantify the first three chromosomes of a population in order to have good seeding in the initial population quantification. It is shown in Table 3 using three policies, with random permutations determining the priority of the remaining chromosomes. Because their priorities are invalid, randomly generated chromosomes are arranged from left to right to ensure that priority constraints are not broken. The first population creation process is depicted in Algorithm 1.

Table 3: An example of task priorities in a cloud environment.

Sub-task (t_i)	Rankb	Rankt	Rankb + t	Level
T0	129	0	123	0
T1	81	24	105	1
T2	78	23	101	1
T3	100	17	117	1
T4	99	30	129	1
T5	60	50	110	2
T6	67	52	119	2
T7	35	75	110	3
T8	42	77	119	3
T9	12	107	119	4

$$\text{rank}_b(t_i) = \overline{W(t_i)} + \max_{t_j \in \text{SUCC}(t_i)} \left(C\left(t_i, t_j\right) + \text{rank}_b\left(t_j\right) \right). \quad (1)$$

In (1), $W(t_i)$ the average computational cost of task t_i, $C(t_i, t_j)$ the amount of communication cost between tasks t_i and t_j and $\text{rank}_b(t_j)$ the upward rank is the sub-task t_i.

$$\text{rank}_t(t_i) = \max_{t_j \in \text{SUCC}(t_i)} \left(\text{rank}_t(t_j) + W(t_j) + C(t_i, t_j)\right). \quad (2)$$

In (2), $\text{rank}_t(t_j)$ is the downward rank is the sub-task t_i.

$$\text{Level}(t_i) = \begin{cases} 0, & \text{if } t_i = t_{\text{entry}}; \\ \max\limits_{t_j \in \text{pred}(t_j)} \left(\text{Level}(t_j) + 1\right), & \text{otherwise.} \end{cases} \quad (3)$$

In equation (3), $\text{Level}(t_j)$ is the precedence sub-task of t_i.

3.3. Assigning Sub-Tasks to Processors. Each chromosome in the initial population must have a legitimate order of precedence, i.e., the precedence restrictions on the chromosome must not be violated. The suggested approach for work assignment processors has been subjected to the HEFT processor allocation policy [10]. In this procedure, the sub-task with the greatest priority is chosen from the chromosome's sub-tasks and given to processors with the shortest execution time compared to other processors. An insert-based scheduling strategy is used in this processor allocation approach. This approach is discussed in the next section. The earliest start time is the t_i task on the p_k processor. EST (t_i, p_k) is symbolized and obtained from

$$\text{EST}(t_i, p_k) = \max\left\{\max_{t_j \in \text{pred}(t_i)} \left(AFT(t_j) + C(t_i, t_j)\right)\right\}. \quad (4)$$

The moment when the actual task t_i began running on processor p_k with AST (t_i, p_k) is symbolized and obtained from

$$\text{AST}(t_i, p_k) = \max(\text{AST}(t_i, p_k)), \text{Avail}(p_k)), \quad (5)$$

where Avail (p_k) is the time when the p_k processor is idle and ready to execute tasks. Earliest finish time of sub-task t_i on processor p_k with EFT (t_i, p_k) is symbolized and calculated from

$$\text{EFT}(t_i, p_k) = W(t_i, p_k) + \text{EST}(t_i, p_k), \quad (6)$$

here, $W(t_i, p_k)$ is defined as the amount of computing effort required to complete task t_i on processor p_k which is actually the real-time finish of sub-task t_i on processor p_k with AFT (t_i, p_k).

It is shown and obtained from

$$\text{AFT}(t_i, p_k) = \min_{1 \le l \le p} \text{EFT}(t_i, p_l), \quad (7)$$

which p_k in the above equation is the fittest processor for the sub-task t_i. The pseudocode for assigning tasks to processors is explained in Algorithm 2.

3.4. Fitness Function. The degree of fitness of chromosomes is critical in determining which chromosomes to carry on to the next generation. The real finish time of the final sub-task in the directed graph is without rounds; or in other words, the graph's output node equals the program's execution time. (8) yields the execution time of the whole program, which is the same as the appropriateness of chromosome *i*.

$$\text{makespan } i = \max\{\text{AFT}(t_{\text{exit}})\}. \quad (8)$$

A low fitness signifies a short overall execution time. As a result, the poorest and best chromosomes in the population have the greatest and lowest fitness rates, respectively.

3.5. Crossover. Crossover is a key operator in GAs for modifying population chromosomes [8]. In GAs, the crossover operator also contributes to population evolution. To create a new generation of chromosomes, the operator unites more than one chromosome. Some qualities are inherited from the first parent, while others are inherited from the second parent. Algorithm 3 describes the crossover operation in details.

The single-point operator is simplified to a MapReduce framework in Algorithm 4, with each combined action allocated to a mapping.

3.6. Mutation. GA mutations are employed to preserve population variability by altering chromosomes. Following the use of the combination operator, several chromosomes are modified using the mutation operator to prevent local optimum convergence and to produce variety in the population. Two more genes are designated for mutation procedures. Algorithm 5 describes the suggested algorithm's mutation operation.

In Algorithm 6, mutation operators are condensed to combine with the MapReduce architecture.

3.7. Selection. In the suggested approach, the technique of picking a roulette wheel with random acceptance is utilized to choose several chromosomes for the local search algorithm (LSA) [16]. It has a temporal complexity of O in this manner (1). This method consists of two major steps. In the first step, a chromosome is selected at random from the population. The probability $(1/n)$ of selection is used. The next step is to analyze the chromosome fitness value and to find if the locally selected chromosome is one of the elites of the population. Otherwise, the selected chromosome is not accepted and the process is repeated. Algorithm 7 provides the pseudocode for selecting a roulette wheel with random acceptance.

3.8. Termination Conditions. The primary distinction between natural evolution and the issue in natural evolution is that species in nature do not tend to end, but in order to solve a problem within the stipulated budget, the process must be halted [17]. There are popular ways for testing the termination condition of algorithms, such as limiting the cost of fitness evaluation function or computer clock time, or monitoring diversity and terminating it when population diversity diminishes [14]. The termination condition in the proposed method happens when all of the chromosomes, or solutions, converge to the same degree of fit.

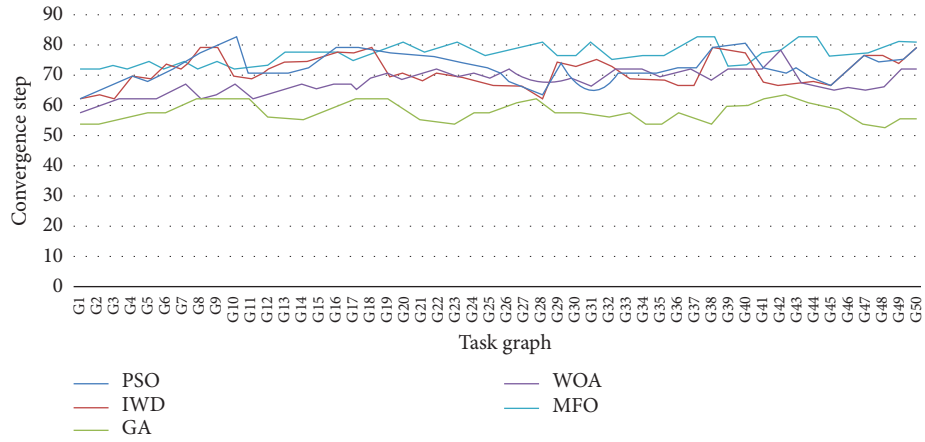

Figure 2: The algorithms' convergence speed is determined by running them on 50 graphs with 20 tasks.

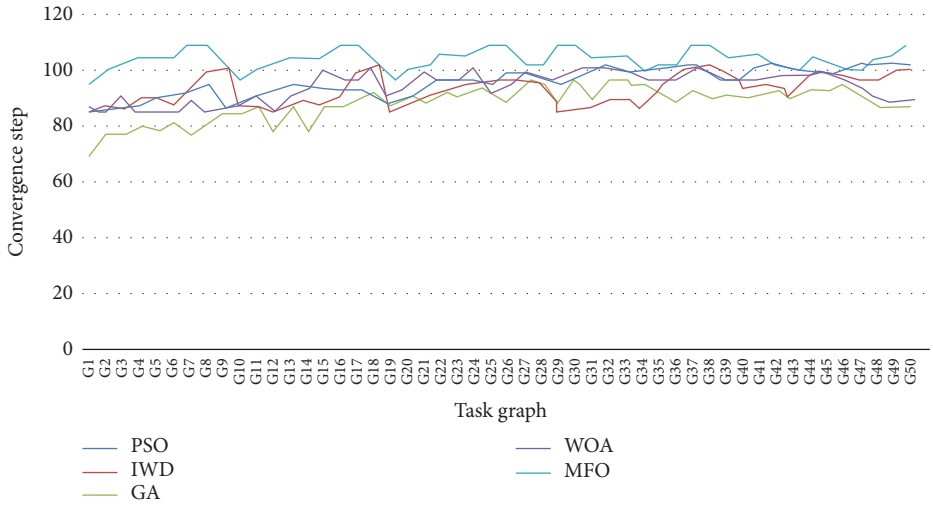

Figure 3: The convergence speed achieved from the methods' implementation on 50 graphs with 40 tasks.

4. Simulation Results

The suggested technique of the study, including job scheduling cloud computing utilizing GA, is evaluated in this paper using the Matlab software R2016b v9.1 × .64 and executed on a computer system with an Intel® Core™2 Duo E4500 processor and 2 GB RAM.

This study considers heterogeneous resources, which vary in their capacity to execute various tasks, and executing a job on different resources might result in varied execution durations. In the equivalent graph, communication cost or edge cost refers to the time spent transferring the outcome of one job to another.

Two graphs, $G1_{n\times n}$ for the communication cost between tasks and $G2_{n\times m}$ for the computing cost of each task, are required to construct each random DAG, where n is the number of tasks and m is the number of available resources. The computation cost is created at random between 5 and 20, while the communication cost is generated at random between 1 and 10 every millisecond unit. In the experiments, the following algorithm parameters are set. The starting population size is 50, the maximum number of iterations is 120, the inertia weight is 0.7, and $C1$ and $C2$ are equal to 2. The test phase was divided into three phases, each with 50 graphs and no different activities (20, 40, 60).

In addition to the PSO and IWD algorithms, the results of the proposed method are compared against the results of the well-known WOA and MFO algorithms. The identical method was utilized for all five of the abovementioned test conditions.

Figures 2–4 depict graphs with 20 tasks, 50 graphs with 40 tasks, and 50 graphs with 60 tasks. All algorithms have the same set of parameters. In addition, the maximum number of iterations in each of the five algorithms is set at 120.

In these graphs, the number of iterations required for the algorithm to reach the optimum solution is referred to as the convergence step. When the number of tasks equals 20, the PSO algorithm's convergence step is changed in the interval (63-81), the IWD algorithm's in the interval (63-80), the WOA algorithm's in the interval (58-72), the MFO

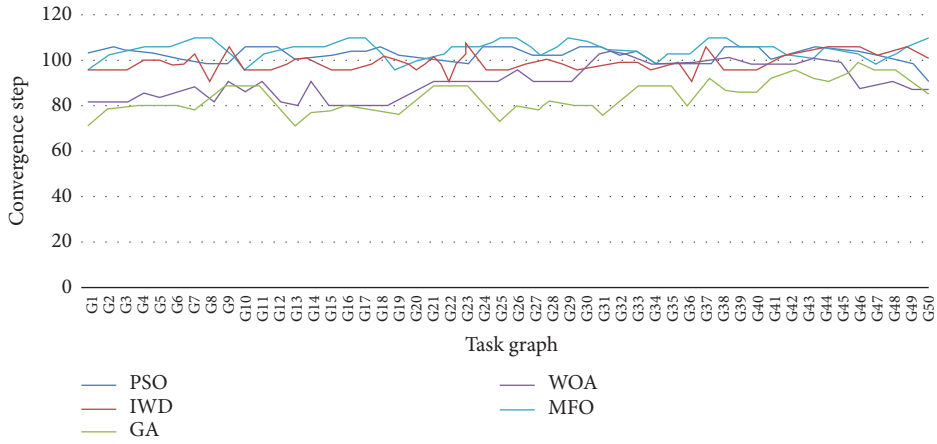

Figure 4: The algorithms' convergence speed is determined by running them on 50 graphs with 60 tasks.

Applying three different heuristic ranking strategies on the first three of the chromosomes.
for *i from* 3 to *PopSize*-1 **do**
for *j from* 0 to *ChSize*- 1 **do**
produce at random a gene with the parameters $j \in (0, \text{ChSize} - 1)$ that has not been generated in previous genes.
In order to maintain a topological order that is reliable, transfer chromosome *i* from its current position on the left to its new position on the right in the queue.
end for
end for

Algorithm 1: Creation of the initial population.

Push sub-task **to** priority queue
while not empty (priority queue)
Pop t_i (first sub-task) **From** priority queue
for each processor p_k**do**
insertion-based HEFT scheduling policy (AFT (t_i, p_k))
assign t_i to the processor p_k
end for
end while
Return makespan = AFT (t_{exit})

Algorithm 2: Sub-task allocation to processors.

algorithm's in the interval (72-82), and the GA algorithm's in the interval (53-63). Furthermore, MFO has had the poorest average outcomes. The GA's advantage is also evident in all iterations.

When the number of tasks reaches 40, the convergence step in the PSO algorithm shifts to the interval (85-103), the IWD method shifts to the interval (85-100), the WOA algorithm shifts to the interval (85-100), the MFO algorithm shifts to the interval (96-108) and the GA algorithm shifts to the interval (77-97). Again, as seen in the graph, MFO has produced the poorest outcomes in this circumstance. The results of the algorithms in the second scenario are closer than the results of the algorithms in the other two cases. IWD and WOA outcomes are closely related, and in 16 instances, they are identical. The IWD algorithm produced the best results in 21% of instances, the GA method in 72%, and the WOA algorithm in 7% of cases.

Furthermore, when the number of tasks reaches 60, the PSO algorithm's convergence step shifts to (90-105), the IWD algorithm shifts to (95-105), the WOA method shifts to (81-101), the MFO algorithm shifts to (96-109), and the GA algorithm shifts to (71-102). As the number of tasks rises, WOA outcomes approach GA, and PSO results approach IWD. MFO had the poorest average performance in the third scenario, as in the previous situations. WOA performed better in 26% of the instances, whereas GA performed best in the remaining situations. Table 4 displays the best, average, worst, and standard deviation of convergence

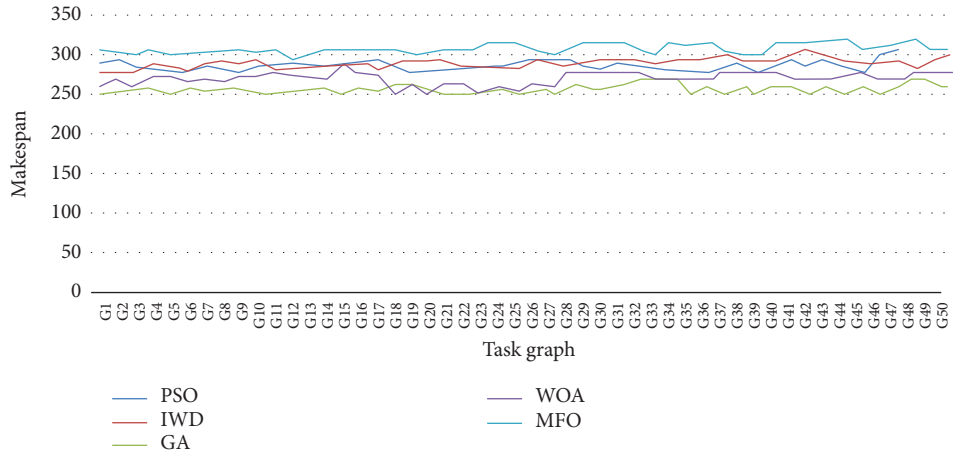

Figure 5: Makespan results from running the algorithms on 50 graphs with 20 tasks.

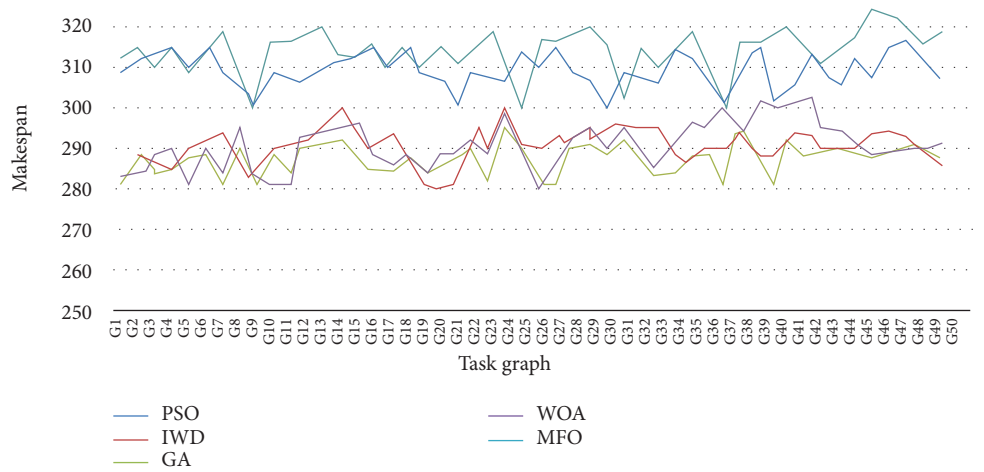

Figure 6: Makespan is determined by running the algorithms on 50 graphs with 40 tasks.

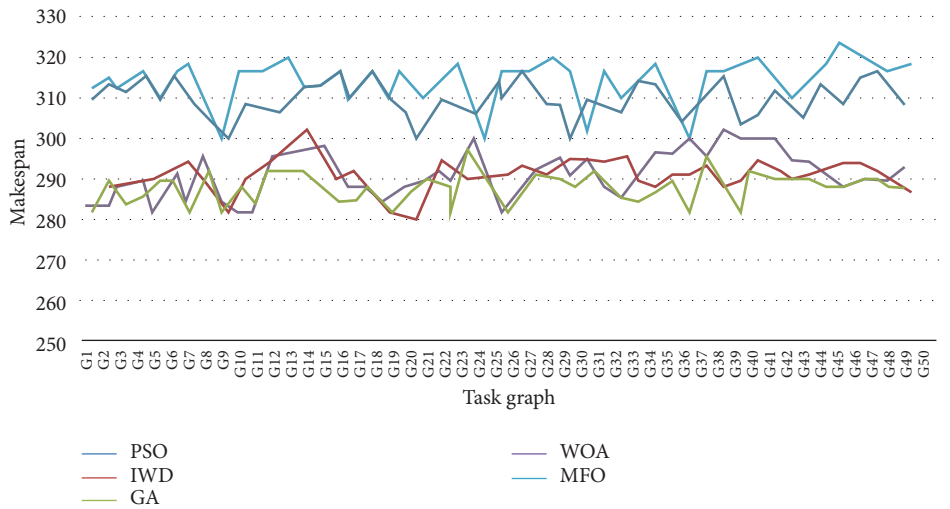

Figure 7: Makespan results from running the algorithms on 50 graphs with 60 tasks.

Input: Two parents from the existing population
Output: Two offspring
Select at random an appropriate crossing point *i*
Separate the chromosomes of the father and mother into left and right parts.
Create a new offspring, specifically a son
Transfer the father's left chromosomal segment to the son's left chromosome segment.
Copy genes from the mother's chromosome that do not occur in the father's left chromosome segment to the right chromosome segment of the son.
Produce a new offspring, specifically a daughter
Transfer the mother's left chromosomal segment to the daughter's left chromosome segment.
Copy genes from the father's chromosome that do not occur on the mother's left chromosome to the daughter's right chromosome.
Return (Two selected offspring)

Algorithm 3: Crossover operation.

Input: Mappers
Output: New Mappers
For each Mapper *do* Algorithm 5 (A chromosome Ti that has been picked at random)
Execute the MapReduce task manager
Record the result as *M* blocks
Transfer the outcome to the shuffle step.
The best person should be written to the global file in DFS if all individuals have been processed successfully
Return (New Mappers)

Algorithm 4: Crossover mapper.

Input: A chromosome *Ti* that has been picked at random.
Output: A new chromosome is being created.
Line 1: from *Succ (i)* **select** the first successor T_j
In the interval [*i*+1, j-1] Choose a gene T_k randomly
for all T_l member *Pred (k)*
if $l < i$ then
swapped (T_i,T_k) to create a new generation.
Return the new offspring
else
Go to Line 1
end if
end for

Algorithm 5: Mutation operation.

(1) Give each Reducer the result of Algorithm 6.
(2) Activate the MapReduce task manager.
(3) Begin the process of calculating.
(4) Incorporate the findings into M-sized blocks.
(5) Verify that the termination condition has been met.
(6) **if** all individuals are properly processed **then** Write (best individual, DFS).
(7) The result should be sent to the cluster.

Algorithm 6: Mutation reduce.

Input: A chromosome.
Output: A new chromosome
Generate $R \in [0, \text{PopSize} - 1]$ a random number
  if (make span R/make $\text{span}_{\min}$) == 1 then
    returnchoromosome$_R$;
  end if
while (founding one of the fitness solutions)

Algorithm 7: Stochastic acceptance for roulette wheel selection.

Table 4: The best, average, worst, and standard deviation of convergence speed results.

Test scenario	Algorithm	Best	Medium	Worst	Standard deviation
First scenario	IWD	63	74	80	7.23
	PSO	63	78	81	6.58
	GA	53	57	63	4.08
	WOA	58	68	72	6.21
	MFO	72	80	82	4.76
Second scenario	IWD	85	94	100	6.24
	PSO	85	98	103	8.04
	GA	70	88	97	11.61
	WOA	85	94	100	6.24
	MFO	96	104	108	5.16
Third scenario	IWD	95	103	105	4.76
	PSO	90	96	105	6.24
	GA	71	84	102	11.90
	WOA	81	99	101	10.64
	MFO	96	104	109	5.44

Table 5: The best, average, worst, and standard deviation results of makespan.

Test scenario	Algorithm	Best	Medium	Worst	Standard deviation
First scenario	IWD	280	296	301	9.86
	PSO	280	298	303	10.78
	GA	251	**256**	270	8.59
	WOA	260	281	290	8.22
	MFO	299	310	316	7.25
Second scenario	IWD	282	302	306	8.96
	PSO	299	309	316	7.04
	GA	273	280	288	6.13
	WOA	**272**	**278**	291	3.87
	MFO	306	313	316	4.39
Third scenario	IWD	282	296	300	8.40
	PSO	300	312	316	7.30
	GA	**281**	**288**	295	5.71
	WOA	**281**	289	301	41.12
	MFO	300	319	324	11.34

Bold numbers represent the best results achieved.

speed values. Figures 5–7 illustrate the results of implementing the proposed method and comparing algorithms on 50 graphs with a varying number of jobs. All algorithms have the same set of parameters. In addition, the maximum number of repeats in all three situations is 120. After 120 repetitions, the values of makespan are recorded and reported in Table 4.

As illustrated in the diagram, the PSO and IWD outcomes in the first scenario are quite near to each other. The MFO algorithm had the poorest performance, whereas GA had the highest performance in 86% of situations.

In the second scenario, PSO degrades the most and is worse than MFO in 13 occurrences. Similar to the previous example, GA had the greatest performance; however, it performed lower than WOA on 10 occasions.

In the third scenario, the IWD outcomes of two GA and WOA algorithms became closer and even outperformed each other in five instances. However, GA continues to outperform in 67% of situations. As illustrated in Figures 5–7, the overall tendency is that as the number of tasks rises, makespan increases, PSO performance declines, and IWD performance improves. In addition, Table 5 displays the best, average, worst, and standard deviation outcomes of makespan.

As demonstrated in this section, the GA algorithm solved the problem better than the other four algorithms and was able to compete with them. In the first and third scenarios, the GA algorithm outperformed the compared algorithms, and only in the second scenario did the WOA algorithm outperform the GA algorithm by one unit. This proves the GA algorithm's stability and convergence.

5. Conclusion and Future Works

Scheduling is a critical topic in distributed computing systems, and several scheduling techniques have been suggested. This research presents a parallel GA for static tasks in cloud computing systems utilizing a MapReduce framework. This method was created by merging genetic algorithms with the HEFT algorithm, which is used to allocate sub-tasks to processors. The proposed method accomplishes task scheduling in two stages: first, the GA was used in conjunction with heuristic techniques to assign tasks to processors, and then the GA was used in conjunction with the MapReduce framework to assign jobs to processors. In our experiments, we consider heterogeneous resources that differ in their ability to execute various tasks, as well as running a job on different resources with varying execution durations. The results show that the proposed method outperforms other algorithms such as PSO, WOA, MFO, and IWD.

One disadvantage of this strategy is that the graphs' computational and communication costs are chosen at

random from a range. MapReduce, on the other hand, is low-level programming that necessitates a significant amount of code in order to obtain the mapping and reduction functions. As a result, future research on the proposed technique could be based on real-world graphs like the fast Fourier transform (FFT), molecular dynamics code, and so on. In the mapping and communication stage segment of Google's MapReduce approach, there is a brief overlap. Unfortunately, this overlap in MapReduce software, which has a longer communication phase than the mapping phase, does not affect reaction time. If it is possible to provide solutions that can overlap the massive stages of communication and reduction in such software, the necessary time of program execution may be reduced while simultaneously maximizing the available resources.

Data Availability

Test data are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of electrical and electronics engineering*, vol. 8, pp. 679–690, 1999.

[2] P. Pirozmand, A. Javadpour, H. Nazarian, P. Pinto, S. Mirkamali, and F. Ja'fari, "GSAGA: a hybrid algorithm for task scheduling in cloud infrastructure," *The Journal of Supercomputing*, vol. 78, no. 8, pp. 1–27, 2022.

[3] Ibrahim, "Task scheduling algorithms in cloud computing: a review," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 4, pp. 1041–1053, 2021.

[4] I. Attiya, M. Abd Elaziz, and S. Xiong, "Job Scheduling in Cloud Computing Using a Modified harris Hawks Optimization and Simulated Annealing Algorithm," *Computational intelligence and neuroscience*, vol. 2020, Article ID 3504642, 2020.

[5] T. Pencheva, K. Atanassov, and A. Shannon, "Modelling of a roulette wheel selection operator in genetic algorithms using generalized nets," *International Journal Bioautomation*, vol. 13, no. 4, p. 257, 2009.

[6] I. Seyedi, M. Hamedi, and R. Tavakkoli-Moghadaam, "Developing a mathematical model for a multi-door cross-dock scheduling problem with human factors: a modified imperialist competitive algorithm," *Journal of Industrial Engineering and Management Studies*, vol. 8, no. 1, pp. 180–201, 2021.

[7] A. J. Crispin and V. Rankov, "Automated inspection of PCB components using a genetic algorithm template-matching approach," *International Journal of Advanced Manufacturing Technology*, vol. 35, no. 3-4, pp. 293–300, 2007.

[8] C. Seyedi, M. Hamedi, and R. Tavakkoli-Moghaddam, "Optimization for a truck scheduling problem in multi-door cross dokcing with learning effect and deteriorating jobs," *Journal of Transportation Research*, vol. 19, no. 71, pp. 183–206, 2022.

[9] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," *BMC Bioinformatics*, vol. 11, no. S12, pp. 1–6, 2010.

[10] S. N. Khezr and N. J. Navimipour, "MapReduce and its application in optimization algorithms: a comprehensive study," *Majlesi Journal of Multimedia Processing*, vol. 4, no. 3, 2015.

[11] T. Gunarathne, T. L. Wu, J. Qiu, and G. Fox, "MapReduce in the clouds for science," in *Proceedings of the 2010 IEEE second international conference on cloud computing technology and science*, pp. 565–572, IEEE, Indianapolis, IN, USA, November 2010.

[12] L. Golshanara, S. M. T. Rouhani Rankoohi, and H. Shah-Hosseini, "A multi-colony ant algorithm for optimizing join queries in distributed database systems," *Knowledge and Information Systems*, vol. 39, no. 1, pp. 175–206, 2014.

[13] M. Malik, S. Rafatirad, and H. Homayoun, "System and architecture level characterization of big data applications on big and little core server architectures," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 3, no. 3, pp. 1–32, 2018.

[14] W. Wu, W. Lin, C. H. Hsu, and L. He, "Energy-efficient hadoop for big data analytics and computing: a systematic review and research insights," *Future Generation Computer Systems*, vol. 86, pp. 1351–1367, 2018.

[15] H. Liu, "Research on cloud computing adaptive task scheduling based on ant colony algorithm," *Optik*, vol. 258, Article ID 168677, 2022.

[16] N. Manikandan, N. Gobalakrishnan, and K. Pradeep, "Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment," *Computer Communications*, vol. 187, pp. 35–44, 2022.

[17] L. Imene, S. Sihem, K. Okba, and B. Mohamed, "A Third Generation Genetic Algorithm NSGAIII for Task Scheduling in Cloud Computing," *Journal of King Saud University-Computer and Information Sciences*, 2022.

[18] N. Manikandan, P. Divya, and S. Janani, "BWFSO: hybrid Black-widow and Fish swarm optimization Algorithm for resource allocation and task scheduling in cloud computing," *Materials Today Proceedings*, vol. 62, pp. 4903–4908, 2022.

[19] S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced multi-verse optimizer for task scheduling in cloud computing environments," *Expert Systems with Applications*, vol. 168, Article ID 114230, 2021.

[20] T. Oudaa, H. Gharsellaoui, and S. Ben Ahmed, "An agent-based model for resource provisioning and task scheduling in cloud computing using DRL," *Procedia Computer Science*, vol. 192, pp. 3795–3804, 2021.

[21] A. K. Arzoo and A. Kumar, "Hybrid ant particle swarm genetic algorithm (APSGA) for task scheduling in cloud computing," *Information and Communication Technology for Competitive Strategies (ICTCS 2021)*, vol. 9-20, pp. 9–20, 2023.

[22] S. Mangalampalli, S. K. Swain, and V. K. Mangalampalli, "Multi objective task scheduling in cloud computing using Cat swarm optimization algorithm," *Arabian Journal for Science and Engineering*, vol. 47, no. 2, pp. 1821–1830, 2022.

[23] G. Attiya, I. Zeidan, and A. Aida, "Nasr, "Elite learning Harris hawks optimizer for multi-objective task scheduling in cloud computing"," *The Journal of Supercomputing*, vol. 78, pp. 2793–2818, 2022.