Received: 9 February 2021 | Revised: 23 May 2021 | Accepted: 24 June 2021

DOI: 10.1002/cpe.6513

SPECIAL ISSUE PAPER

WILEY

Metaheuristic task scheduling algorithms for cloud computing environments

Merve Nur Aktan[1] | Hasan Bulut[2]

[1]Department of Computer Engineering, Aydın Adnan Menderes, Aydın, Turkey

[2]Department of Computer Engineering, Ege University, Izmir, Turkey

Correspondence
Hasan Bulut, Department of Computer Engineering, Ege University, 35100 Izmir, Turkey
Email: hasan.bulut@ege.edu.tr

Summary

Cloud computing has the advantage of providing flexibility, high-performance, pay-as-you-use, and on-demand service. One of the important research issues in cloud computing is task scheduling. The purpose of scheduling is to assign tasks to available resources while providing optimization on some objectives. Tasks have diversified characteristics, and resources are heterogeneous. These properties make task scheduling an NP-complete problem. In this study, metaheuristic and hybrid metaheuristic algorithms are developed for task scheduling problems in cloud computing environments. We have developed genetic algorithm (GA), differential evolution (DE), and simulated annealing (SA) based metaheuristic algorithms, which are also combined with a greedy approach (GR). In addition to this, we have developed hybrid metaheuristics algorithms, called DE-SA and GA-SA, which are also combined with a greedy approach. The proposed approaches are evaluated in terms of completion time and load balancing of virtual machines. In terms of average completion time, as the number of tasks increases, it has been observed that the DESA algorithm outperforms the solely used DE and SA algorithms. In addition, experiments show that hybrid algorithms improve both the average completion time and the average standard deviation of virtual machine loads for some task groups.

KEYWORDS

cloud computing, differential evolution algorithm, genetic algorithm, metaheuristic algorithms, simulated annealing, task scheduling

1 | INTRODUCTION

Task scheduling problems can be seen as assigning different tasks on the available resource set most appropriately to achieve the goals defined for the tasks.[1] Task scheduling is an essential and the most crucial part of a cloud computing environment. There are many users in addition to a large amount of data. Resource sharing and reuse demands are increasingly becoming necessary. Cloud computing scheduling objectives can be categorized as consumer and service provider objectives. While consumer objectives focus on makespan, budget, and deadline, service provider objectives may focus on resource utilization and load balancing metrics. An effective task scheduling policy should optimize the objectives from both categories.[2,3]

Task scheduling can be identified as a general problem among distributed environments such as cluster computing, grid computing, and cloud computing. As a general definition, distributed environments consist of connected computer resources via network and collaboration to achieve a common goal. However, each type of them has a different mechanism. These differences should be taken into consideration while designing a task scheduling algorithm. For instance, grid computing is a collection of computer resources placed in geographically different locations. Generally, grids

have a decentralized scheduler. Each grid site has its grid scheduler; that is why the grid scheduler has not complete control of resources. On the other hand, the cloud environment has a centralized scheduler called a global broker. Grid systems mainly aim to provide optimal workload between resources; however, cloud systems aim to provide dynamic scheduling via scaling up or down to satisfy varying resource demands. Scalability is an essential part of a cloud. Another difference between grid and cloud is that cloud has service provider companies, and the scheduling process should take notice to optimize service provider objectives. In contrast, the grid has authorized users to adjust processing power, memory, data, and storage.[4-6]

There are various applications, from cloud monitoring software to big data processing, developed for cloud computing and grid systems. Cloud computing monitoring software,[7] which is platform-independent, detects malfunctioning problems, patterns causing cloud system failure. Proposed software processes large volume events in a reasonable time. Nacar et al.[8] proposed a grid and web services system for distributed and collaborative computational chemistry and material science. A semantic research grid (SRG)[9] integrating academic research, scientific databases, and journal-conference content management systems is proposed. SRG framework consists of tools and services which support cyberinfrastructure-based scientific research.

There are diverse big data applications developed on distributed systems. Case studies for face detection,[10] Twister:Net,[11] and Support vector machine (SVM) training optimization[12] are examples. First of all, generally, face detection applications need the processing of a small-sized and large number of images. However, the Hadoop HDFS file system uses the processing of large-size images. Also, a small-sized and large number of images may cause a slowing down in HDFS because of an increasing number of jobs, scheduling overhead, and memory usage. There are two solutions proposed in work proposed by Demir and Sayar.[10] One of them is, merging all files into a single large file, and the other is, combining images without merging. Also, they proposed a novel Hadoop file format to ease the process. Secondly, Twister: Net study[11] aims to develop polymorphic implementation, which accepts fundamental features of big data as independent components rather than integrated communication, task-data management aspects. Lastly, SVM training optimization[12] aims to provide optimization on efficient training. Performance optimization and benchmarks are discussed. The study represents a scientific approach to the classic HPC model, dataflow model, and hybrid systems.

Distributed web service handlers are another research topic in distributed systems. Yildiz and Fox[13] proposed distributed handler to provide efficient, scalable, and modular architecture. The proposed approach is promising according to performance and scalability benchmarks. The study also contains a discussion about promising computing environment fundamentals for web service handlers. Yildiz et al.[14] implemented distributed approach for web service handler execution to increase utilization of handler. They explained an orchestration structure for handlers to obtain improved web services.

There are some studies about Real-time applications on Grids. Fox et al.[15] developed a study to support streaming data services for geographical information system grid services. They examined streaming approaches that can be used while transferring XML messages to attain increased data service performance. In addition, efficient XML representation techniques which may improve the performances of web services are discussed. Another study, is carried out by Fox et al.,[16] aims to build high-level services to produce and consume real-time data streams and proposes a message-based approach to managing real-time data streams.

Thus, task scheduling problems exist in the cloud and noncloud environments. However, because of different characteristics such as scheduler type and objectives, which have to be optimized, the scheduling algorithm shows some differences. In our study, we designed a task scheduling algorithm considering cloud computing, so the central scheduler is used, and an objective function is created considering cloud consumer and cloud service provider objectives. The scheduling algorithm used in this study may also be applied to noncloud environments; however, necessary adjustments must be made.

Cloud computing task scheduling belongs to NP-hard problems. Since there are no known polynomial-time algorithms for NP-hard problems, heuristic and metaheuristic approaches are used frequently to find a near-optimal solution.[3] Task scheduling is an optimization problem. Optimization problems have an objective function that is used for reaching the best state. While heuristics algorithms try to find the best state, they mostly stuck at local minimum/maximum points. On the other hand, most heuristic algorithms are not easily applicable to varying problems. Metaheuristics emerged as an improved version of heuristic algorithms, and they can easily be applied to real-life problems to overcome these difficulties.[17,18]

According to Bittencourt et al.'s[19] work, they provide a review article to researchers to identify scheduling problems in the distributed system. Also, they give essential knowledge about the scheduling model, pre-cloud scheduling taxonomy, and scheduler objectives. Their work gave us basic concepts about task scheduling in cloud computing and shaped our research direction. They state that the developments in distributed systems, especially in cluster, grid, and cloud computing, brought new challenges to scheduling. The scheduler has the role of assigning computational tasks to computational resources based on an objective function. Usually, this objective function minimizes the completion time, or makespan, and other parameters if desired.

For static task scheduling of independent and nonpreemptive tasks, Ebadifard and Babamir[20] used a load balancing technique to improve the basic particle swarm optimization (PSO) method. The simulation results obtained from the Cloudsim simulator environment show that the proposed method is effective with many tasks and converges to the near-optimal solution faster than the basic PSO algorithm. Although we used

different algorithms and experimental configurations, their work gave us insights into using the Cloudsim simulator in our experiments and using metaheuristics to increase resource utilization and decrease makespan.

Ge et al.[21] proposed a task scheduling method based on a differential evolution algorithm, decreasing user cost and execution time of tasks. Our methodology shows some differences from Ge et al.'s methodology. They used dynamic selection strategy and dynamic mutation strategy; however, we used different selection and mutation strategies explained in detail in Section 2.3.2. Also, they used a user cost parameter in the fitness function. They aimed to reduce task execution time and user cost. Their fitness function is designed to minimize a function, whereas our fitness function is designed to maximize a function. Other differences are parameters such as the number of tasks, task instruction length (mi), population size, crossover rate, and the number of generations.

Some hybrid metaheuristic approaches for task scheduling problems in the cloud are used in literature. Tsai et al.[22] proposed a hyper-heuristic scheduling algorithm for cloud (HHSA) that integrates simulated annealing, genetic algorithm, particle swarm optimization, and ant colony optimization. HHSA applies to both sequence-independent and sequence-dependent scheduling problems. It has been tested on both a simulator and a real system. The performance results show that HHSA can reduce the completion time of tasks. There are other hybrid approaches in the literature. Gan et al.[23] proposed a simulated annealing algorithm combined with a genetic algorithm called genetic simulated annealing for task scheduling in cloud computing. A new population is obtained by applying selection, crossover, mutation, and simulated annealing on the existing population. The algorithm provides resource search and allocation in a cloud computing environment efficiently, according to experimental results. Another work presented by Ge et al.[24] demonstrated the DE-ACO approach obtained by combining differential evolution algorithm (DE) and ant colony optimization (ACO), which uses cost, load balancing, and the minimum task completion time as evaluation criteria. The study was performed in a Cloudsim simulator environment, and the results were compared with min–min and ACO algorithms. It has been observed that DE-ACO is superior to these algorithms. Another study proposed by Kamalinia and Ghaffari,[25] a hybrid method in which HEFT, differential evolution, and genetic algorithms are used together. The performance results were obtained on a randomly generated direct acyclic graph dataset. The results reveal that the proposed algorithm outperforms some heuristics in terms of makespan. All these studies show that using the hybrid metaheuristics approach provides some benefits in the solution of the task scheduling problem.

In this study, metaheuristic and hybrid metaheuristic algorithms are developed for task scheduling problems in cloud computing environments. We have developed genetic algorithm (GA), differential evolution (DE), and simulated Annealing (SA) based metaheuristic algorithms, which are also combined with a greedy approach (GR). In addition, we have developed DESA and GASA algorithms, which are hybrid metaheuristics algorithms, to improve the performance of standalone metaheuristic approaches. The hybrid ones are also combined with a greedy approach for a small number of tasks during a scheduling phase.

The rest of the article is organized as follows. Section 2 explains the developed approaches. Then, Section 3 describes evaluation metrics, dataset, parameters, and performance results. Finally, Section 4concludes the article.

2 | METHODOLOGY

A task scheduling model, which incorporates a variable-length sliding window mechanism, a greedy approach, and metaheuristic and hybrid metaheuristic approaches, is proposed in this study. The metaheuristic algorithms are based on genetic algorithm (GA), differential evolution (DE), and simulated annealing (SA). Since SA is faster than other metaheuristic approaches, it is combined with GA and DE to obtain GASA and DESA algorithms. These proposed models use a greedy approach (GR) for a small number of tasks compared to the number of virtual machines.

2.1 | Task scheduling model

Tasks arrive at the system in different time intervals, which continue as long as the system runs. The scheduling process queues incoming tasks into a sliding window and schedules them to machines. The sliding window is a list that contains tasks. The goal is to schedule tasks on the fly as they arrive at the system. There may be too many tasks pending for assignment at a time. Due to this, a variable-length sliding window technique is used to schedule the tasks arriving over a recent period.

The size of the sliding window depends on the number of pending tasks. According to the number of tasks in the sliding window, either metaheuristic or GR is applied. If the number of tasks in the sliding window is equal to or less than the number of virtual machines, GR is applied. The purpose is to take advantage of the improving effect of GR on the completion time to make metaheuristics give results better. Suppose the number of tasks in the sliding window is greater than the number of virtual machines and less than or equal to a predefined number, a multiple of virtual machines. In that case, a related metaheuristic algorithm is applied. If the number of tasks in the sliding window is greater than the predefined number of virtual machines, a related metaheuristic is applied only to the predefined number of tasks. The remaining tasks stay in the sliding window

for the next scheduling time. The order of tasks is preserved in the sliding window queue. Hence, no starvation is observed. Scheduling algorithm results in a mapping of tasks to virtual machines.

2.2 | Greedy approach

In the greedy approach (GR), in this study, the tasks in the sliding window are sorted in descending order based on their sizes for the scheduling stage. The machine with the least load is found, and the task is sent to that machine for scheduling. Then the machine load is updated. This process continues until all tasks in the sliding window are sent to the machines.

2.3 | Metaheuristic algorithms

The metaheuristic algorithms developed in this study are based on genetic algorithm (GA), differential evolution (DE), and simulated annealing (SA). Since all of the metaheuristic approaches utilize chromosome structure and a fitness function, they are explained first.

First of all, chromosome structure will be explained. The chromosome consists of three properties: the list of tasks (cloudlet), the list of the machine number indicating which virtual machine these tasks will be assigned to, and the fitness value of the chromosome. Each gene of the chromosome indicates a task. When a population is created for the first time, the specified number of empty chromosomes is created. Then, tasks in the sliding window are transferred into the chromosome genes. Later, scheduling is done for the chromosome to assign at least one task to each machine. Suppose a chromosome consists of five virtual machines (VM_1, ..., VM_5) and 10 genes (T_1, ..., T_{10}). Cutting points can be 0, 2, 7, 8, and the structure formed accordingly can be shown in Figure 1.

Secondly, the fitness function and parameters which are used in the function will be explained. The fitness function aims to reduce the maxspan and increase resource utilization, as shown in Equation (1).

$$Fitness = \frac{1}{maxspan} \times Avg.\ Resource\ Utilization. \tag{1}$$

As it has been seen in Equation (1), maxspan and average resource utilization are the parameters of the fitness function. Maxspan is the largest task completion time among all the resources in the system when looking at the resources in the system at a particular time. There are existing tasks, pending tasks on the waiting list, and tasks to be assigned based on the scheduling decision in each resource. The completion time of all these tasks is calculated for each resource. The longest completion time is defined as maxspan. The load of each VM is calculated according to Equation (2).

$$VM_j(load) = VM_j(instant\ load) + VM_j(load\ of\ waiting\ list) + VM_j(load\ of\ the\ tasks\ to\ be\ scheduled). \tag{2}$$

Average resource utilization is one of the essential factors for the fitness value of the solution chromosome. High resource utilization indicates that the load is distributed evenly among all resources. The completion time can be reduced by increasing resource usage. The expected usage of each resource based on the given task assignment is obtained by dividing the load of each resource by the maxspan value as given in Equation (3).

$$VM_j(resource\ utilization) = \frac{VM_j(load)}{maxspan}. \tag{3}$$

Average resource utilization is specified in Equation (4), where n is the number of virtual machines in the system.

$$Avg.\ Resource\ Utilization = \frac{\sum_{j=1}^{n} VM_j(resource\ utilization)}{n}. \tag{4}$$

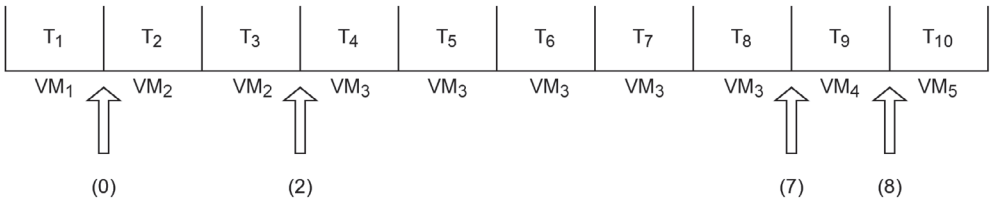

FIGURE 1 Developed chromosome structure

2.3.1 | Genetic algorithm

Genetic algorithms (GA) are created based on inspiration from nature's selection and genetics processes. A new solution (chromosome) set is created using parts from the previous generation in every generation. GA efficiently uses previous information and fitness function for creating the next generation.[26] A simple GA that is used for many problems consists of three operators. These are selection, crossover, and mutation.[27] In this study, the elitism method, tournament selection, ordered crossover, and swap mutation were used. They will be explained respectively.

Elitism property is used to prevent loss of chromosomes with high fitness value because the resulting chromosomes may be produced with lower fitness value after crossover and mutation applications.[28] In the elitism method, the best chromosome of the previous generation is forwarded to the new one without any changes. Thus, the best chromosome is preserved. By transferring the properties of the best chromosome into the next generation, elitism reduces genetic deviation.[29]

The mating pool consists of selected chromosomes from the population in the tournament selection method. Chromosomes placed in the mating pool are used to create a new child. These children shape the next generation. The mating pool should comprise good chromosomes since the next generation inherits them. In the tournament selection, the winner is the chromosome with the highest fitness among the s tournament rivals, where s is the tournament size. Then the winner is inserted into the mating pool.[30] A new chromosome is created from the selected chromosomes by applying tournament selections twice.

As a result of ordered crossover, a single child is created from two parents. Assume that parents are P_1 and P_2, the child is C_1. Two random cut points are determined for P_1, and the genes between these two cut points are copied into C_1 while keeping their positions. After that, starting from the first gene of P_2, genes whose task number does not exist in C_1 are transferred into C_1 in the order they appear in P_2, as exemplified in Figure 2.

After the crossover process, swap mutation is applied. Two genes (tasks) that have different virtual machine numbers are selected on the resulting chromosome. A random number is chosen between 0 and 1. If this chosen number is less than the mutation rate, an exchange is made between the selected genes.

The general flow of GA is as follows: The tasks in the sliding window are transferred into the chromosome. Let n_c (i.e., 10) be the number of chromosomes in the population and n_v (i.e., 5) be the number of virtual machines in the system. For the first population, $pop_{current}$, the tasks in the sliding window are randomly mixed to create the chromosomes. A list of the machine numbers is assigned to these chromosomes. The fitness value of each chromosome in $pop_{current}$ is calculated as in Equation (1). The best chromosome is directly transferred into pop_{next}. For each of the remaining chromosomes in pop_{next}, two chromosomes are selected by the tournament selection method from $pop_{current}$ and then ordered crossover is performed. The resulting child, which is obtained after crossover, is added to pop_{next}. The list of machine numbers is assigned to the chromosome in pop_{next} and swap mutation is applied to it. This process continues until all the chromosomes in pop_{next} are created. In this way, $2n_c$ (i.e., 20) generations are created. The chromosome with the highest fitness value of the last generation is the solution chromosome, and the task-virtual machine assignment specified in the solution chromosome is performed.

2.3.2 | Differential evolution

The differential evolution (DE) algorithm is also population-based as the genetic algorithm and uses similar operators such as crossover, mutation, and selection. The major difference between them is that the DE focuses on mutation while the genetic algorithm focuses on the crossover.[31] Parameters and formulas used in DE have been adapted from Karaboga and Ökdem,[31] Schmidt and Thierauf,[32] and Onan et al.[33] The DE parameters are shown in Table 1.

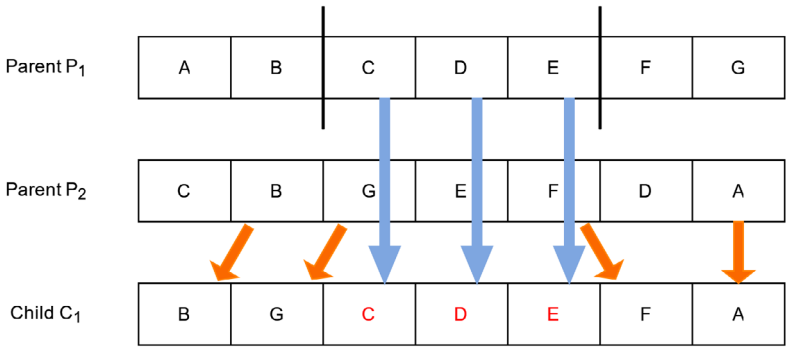

FIGURE 2 Ordered crossover

TABLE 1 DE parameters

Notation	Definition
NP	Population size (number of chromosome), NP ≥ 4
D	Number of variable in chromosome (number of gene) (1, 2, 3, ..., j)
CR	Crossover rate [0.1–1.0] = 0.8
G	Generation $(1, 2, 3, \ldots, (G_{max}))$
F	Scaling factor [0-2] =0.8
$x_{j,i,G}$	j parameter of i chromosome in G generation (gene)
$n_{j,i,G+1}$	Mutation and crossover applied intermediate chromosome
$u_{j,i,G+1}$	A chromosome created for next generation after $x_{j,i,G}$
$r_{1,2,3}$	Randomly chosen chromosomes to be used in the creation of the new chromosome, where $r_{1,2,3} \in \{1, 2, 3, \ldots NP\} r_1 \neq r_2 \neq r_3 \neq i$

The main steps of the DE algorithm used in this study are given in Algorithm 1. In this study, the first population is created by shuffling the tasks in the sliding window. Each task has a normalizedID property, and this is unique, like a taskID. The purpose here is to keep the normalizedID of tasks in the range of [0–1]. Forward transformation and backward transformation processes are adapted for task scheduling to the proposed scheduling method from Onwubolu and Davendra[34] study. The main steps of Algorithm 1 are initialization, evaluation, forward transformation, mutation, crossover, backward transformation. They will be explained below.

Algorithm 1: The Pseudocode of Differential Evolution Algorithm

```
Input: List of Tasks
Initialization
while desired number of populations has not been reached do
    Evaluation
    Forward transformation
    foreach chromosome in the population do
        Mutation
        Crossover
        Backward transformation
        Evaluation
        Selection
```

Initialization is creating the initial population according to the pending tasks in the sliding window. In the evaluation process, the list of the machine numbers is assigned to chromosomes of the population. Then, the fitness value of each chromosome is calculated according to Equation (1). After the evaluation process is completed, the forward transformation process will be applied. In this process, the normalizedID of each task in the sliding window is calculated according to Equation (5).

$$normalizedID(t_i) = \frac{taskID(t_i) - taskID(t_{min})}{taskID(t_{max}) - taskID(t_{min})}, \tag{5}$$

where t_i is the ith task.

The mutation process is applied to create the next generation. Each chromosome of the next generation, which will take the place of the current population's chromosomes, is determined by choosing three different chromosomes from the current population other than itself (target chromosome). The gene values of the first two selected chromosomes are subtracted from each other and multiplied by the F coefficient, and then added with the third chromosome. In this way, the mutated chromosome is obtained, as shown in Equation (6).

$$\forall j \leq D : n_{j,i,G+1} = x_{j,r_3,G} + F.(x_{j,r_1,G} - x_{j,r_2,G}). \tag{6}$$

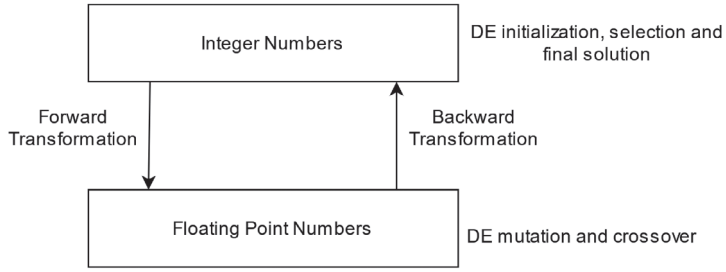

FIGURE 3 DE transformation processes

After that, the crossover between the target chromosome and the mutated chromosome is applied according to Equation (7), and a new chromosome is created.

$$\forall j \le D : x_{j,u,G+1} = \begin{cases} x_{j,n,G+1} & \text{if } (rand[0,1] \le CR) \vee (j = j_{rand}) \\ x_{j,i,G} & \text{otherwise} \end{cases}. \tag{7}$$

Crossover applied chromosome consists of floating-point numbers. Backward transformation is necessary to convert the chromosome into a chromosome consisting of tasks. In the backward transformation process: The tasks in the sliding window are sorted according to the task number in increasing order. Floating-point numbers found in the crossover applied chromosome are sorted in increasing order and matched one-to-one with previously sorted sliding window tasks. Thus, taskID, which corresponds to each floating-point number in the sorted crossover applied chromosome, is determined. Then, the chromosome consisting tasks is formed by preserving the order in the original crossover applied chromosome. DE transformation processes are shown in Figure 3. After the backward transformation, the evaluation process is applied to assign the list of the machine numbers to the resulting chromosome. The fitness value is calculated as described in the previous sections.

Then, the selection process is applied according to Equation (8). The chromosome with the highest fitness value is selected.

$$\forall i \le NP : x_{i,G+1} = \begin{cases} x_{u,G+1} & \text{if } f(x_{u,G+1}) \ge f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases}. \tag{8}$$

DE creates the new population by going through the steps described above, and this process continues until it reaches the specified number of generations. In the last generation, the assignment specified in the chromosome with the highest fitness value is performed.

2.3.3 | Simulated annealing

According to Russell and Norvig,[17] simulated annealing is based on the annealing process as in metallurgy. In the annealing process, metals are heated to a high temperature. Then, they are cooled gradually to provide a low energy state of the material. The simulated annealing (SA) pseudocode used in the study is given in Algorithm 2. The solution refers to a chromosome in the pseudocode. A chromosome is created from pending sliding window tasks. A list of the machine numbers is assigned, and the fitness value is calculated. Initially, the current chromosome is identified as the best solution. The whole process continues while the temperature is greater than 1. First of all, the swap mutation was performed on the next solution. In this process, two genes (tasks) with different virtual machine numbers are selected in the chromosome, and an exchange is made between the selected genes. The current solution refers to the chromosome before mutation. The following solution refers to the chromosome after mutation. At this stage, ΔE is calculated, which is equal to the difference between the fitness value of the next solution and the fitness value of the current solution. If ΔE is greater than 0, the mutation is accepted. Otherwise, the acceptance probability, $e^{-\Delta E/T}$ is calculated, and the mutation is accepted only if the acceptance probability is greater than a predefined threshold, which is a random number between 0 (inclusive) and 1 (exclusive). Then the best solution is updated, and the cooling process is applied.

2.4 | Hybrid metaheuristics

Hybrid metaheuristics use the same chromosome structure and fitness function described in Section 2.3. The hybrid metaheuristic algorithms proposed in this study are developed by combining SA with GA and DE to obtain GASA (genetic algorithm-simulated annealing), and DESA (differential

Algorithm 2: The Pseudocode of Simulated Annealing Algorithm

Input: List of Tasks

```
currentSolution ← makeChromosome(problem.InitialState)
bestSolution ← currentSolution
while T > 1 do
    nextSolution ← currentSolution
    swapMutation(nextSolution)
    ΔE ← nextSolution.fitness - currentSolution.fitness
    if ΔE > 0 then
        currentSolution ← nextSolution
    else
        currentSolution ← nextSolution only random [0 − 1] < e^(−ΔE/T)
    if currentSolution.fitness > bestSolution.fitness then
        bestSolution ← currentSolution
    T ← T ∗ (1 − coolingRate)
```

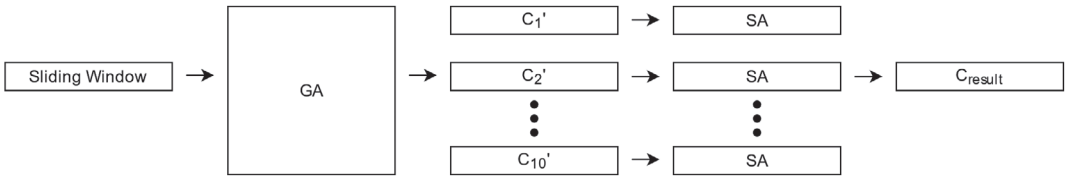

FIGURE 4 GASA process

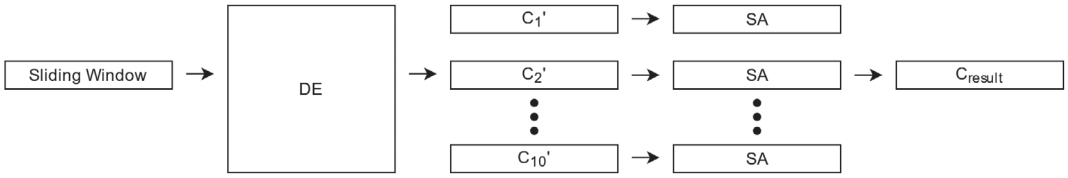

FIGURE 5 DESA process

evolution-simulated annealing) approaches. These proposed models also use a greedy approach (GR) for a small number of tasks with respect to the number of virtual machines.

2.4.1 | Genetic algorithm-simulated annealing

Genetic algorithm-simulated annealing (GASA) is a hybrid algorithm in which GA and SA are applied together. In this approach, GA is executed for 20 generations, and then SA is applied to each chromosome in the last GA population. The initial temperature of SA is 100,000, and the cooling rate is 0.003. Tasks are scheduled to machines according to the best chromosome among the SA applied chromosomes. The GASA process is shown in Figure 4.

2.4.2 | Differential evolution-simulated annealing

Differential evolution-simulated annealing (DESA) is a hybrid algorithm in which DE and SA are applied together. In this approach, DE is executed for 20 generations, and then SA is applied to each chromosome in the last DE population. The initial temperature of SA is 100,000, and the cooling rate is 0.003. Tasks are scheduled to machines according to the best chromosome among the SA applied chromosomes. The DESA process is shown in Figure 5.

3 | EXPERIMENTAL RESULTS

In this section, evaluation metrics, dataset, system, and algorithm parameters will be explained elaborately. Also, experimental results related to completion time, virtual machine loads, and runtime of the scheduling process will be shared.

3.1 | Evaluation metrics

In this study, average completion time and average standard deviation of virtual machine loads are used as evaluation metrics.

3.1.1 | Average completion time

The completion time is defined as the total time required to execute the entire workflow. The completion time of a virtual machine (VM) is defined as the finish time of the last task in that VM. The overall completion time is considered as the maximum completion time of the VMs, as shown in Equation (9). The average completion time is obtained by running a task group n times, recording the completion times, and taking the averages.

$$Completion\ Time = \max_{i \in VM}\{CT_i\}, \tag{9}$$

where CT_i is the completion time of the last task in VM_i.

3.1.2 | Average standard deviation of virtual machine loads

In order to evaluate the load of the virtual machines, the standard deviation in the completion times is measured. A standard deviation value close to zero means that the load is balanced among virtual machines.

3.2 | Dataset

The dataset has been created with uniform distribution. Ten task files for each task group consisting of 50, 100, 200, 300, 400, and 500 tasks are created. The size of the tasks is between 20,000 and 60,000 mi (million instructions), and the time intervals between arrival times of tasks to the system are random values between 5 and 10 s. Each algorithm is executed 30 times on each task file for each task group. The result of the related task group was obtained by taking the average of the results.

3.3 | Parameters

We have used CloudSim 4.0 simulator[35] on a computer with Intel (R) Core (TM) i7-3517U CPU @ 1.90 GHz 2.40 GHz processor and 6 GB memory. CloudSim simulation parameters are given in Table 2. Algorithm parameters for GA, DE, and SA are specified in Table 3. The parameters of the hybrid algorithms are the same as the individual algorithms that compose the hybrid algorithms.

3.4 | Average completion time results

Based on the parameter values given in Section 3.3, we have evaluated the average completion times and the standard deviation of the average completion times, in terms of second, for the proposed algorithms. In addition, we have also provided the results for the greedy approach since it is used with the metaheuristic approaches. The average of the minimum theoretical limit has been calculated for each task group. This value is obtained by summing the execution times of each task in the related task group divided by the number of virtual machines. DE gives the best result for a 50 task scenario, and GA gives the best result for a 100 task scenario, while DESA gives the best results for 200, 300, 400, and 500 task scenarios. In addition, as seen from Table 4, GASA gives the second-best results for 200, 300, 400, and 500 task scenarios. We can conclude that hybrid metaheuristic algorithms give better results for large task sizes compared to single metaheuristic algorithms.

TABLE 2 Simulation parameters

Resource	Parameter	Value
	Number of datacenters	1
	Number of hosts in the datacenter	1
	Number of virtual machines in the host	5
	Number of tasks	50–500
Datacenter	System architecture	x86
	Operating system	Linux
	VM model	Xen
	Time zone	10
Host	Ram	16 GB
	Storage	1 TB
	Bandwidth	10 GB/s
	MIPS of processing element (PE)	5000 MIPS
	Number of PE per Host	1
	Virtual machine scheduler type	Time shared
Virtual machine (VM)	Ram	0.5 GB
	Storage	10 GB
	Bandwidth	1 GB/s
	MIPS of PE	1000 MIPS
	Number of PE per VM	1
	Task scheduler type	Space shared
Task	Length of task	20,000–60,000 mi
	Number of PE requirement	1

3.5 | Average standard deviation of virtual machine loads results

In terms of seconds, the average standard deviation of completion times is given in Table 5 to evaluate the load balance efficiency of the algorithms. The greedy approach gives better results for 50, 200, and 400 task scenarios, while DESA gives better results for 100, 300, and 500 task scenarios. The greedy approach assigns tasks to virtual machines with the minimum completion time and achieves better load balance in three out of six experimental scenarios. Among hybrid metaheuristic approaches, DESA gives better results.

3.6 | Average scheduling time

The average scheduling time shows the runtime of the scheduling process for 5, 10, 15, and 20 tasks in terms of milliseconds, as shown in Table 6. The average scheduling time values for each algorithm are obtained by taking the average of 100 runs in a related task group. Because of the task scheduling model used in this study, scheduling time results for five tasks in each algorithm are obtained by the greedy algorithm. However, scheduling time results for 10, 15, and 20 tasks in each algorithm are obtained by the specified algorithm in Table 6. Scheduling time starts when a sliding window is ready for scheduling and finishes when submission of tasks starts according to the decision of related algorithms about which task will assign to which machine. The maximum number of tasks that can be accepted for scheduling is 20. As we explained in Section 2.1, if the number of tasks in the sliding window is equal to or less than the number of virtual machines, GR is applied. That is why we chose five as a minimum number of tasks in the sliding window for calculating scheduling time. According to the arrival time of tasks, the number of tasks in the sliding window may be less than five or more than 20. Five and 20 are chosen as examples for this experiment. According to parameter values in Table 2, the processing of a task takes between 20,000 and 60,000 ms. It can be taken as 40,000 ms as average. The average scheduling time of 20 tasks takes at most

TABLE 3 Algorithm parameters

Algorithm	Parameter	Value
GA	Population size	10
	Number of generation	20
	Mutation rate	0.015
DE	Population size	10
	Number of generation	20
	Crossover rate	0.8
	F constant	0.8
SA	Initial temperature	100,000
	Final temperature	1
	Cooling rate	0.003

TABLE 4 Average completion time results

	GR	GA-GR	DE-GR	SA-GR	GASA-GR	DESA-GR	Theoretical limit
50 tasks	419.11	418.70	**418.29**	422.00	419.56	420.13	402.00
100 tasks	816.37	**814.76**	815.08	821.44	816.02	815.82	798.57
200 tasks	1612.04	1617.72	1615.59	1650.43	1610.97	**1610.12**	1598.86
300 tasks	2413.63	2415.67	2413.50	2468.50	2407.28	**2405.33**	2392.99
400 tasks	3218.37	3229.22	3226.02	3323.79	3217.34	**3216.27**	3205.40
500 tasks	4004.86	4012.91	4010.35	4105.02	4001.22	**3999.56**	3985.19

TABLE 5 Average standard deviation of virtual machine loads results

	GR	GA-GR	DE-GR	SA-GR	GASA-GR	DESA-GR
50 tasks	**12.71**	13.70	13.68	15.56	14.06	14.64
100 tasks	13.29	13.58	13.57	18.28	13.39	**12.98**
200 tasks	**10.40**	17.90	16.47	43.46	13.03	11.58
300 tasks	15.23	19.88	18.51	60.48	13.92	**11.07**
400 tasks	**10.80**	23.25	20.07	98.83	13.83	11.33
500 tasks	12.85	22.95	21.99	98.75	13.96	**11.27**

TABLE 6 Average scheduling time in milliseconds and their standard deviation

Sliding window size		GR	GA	DE	SA	GASA	DESA
Five tasks	Avg.	20.78	22.18	21.25	19.53	21.87	22.81
	SD	8.61	8.93	9.02	9.25	9.68	10.03
10 tasks	Avg.	41.87	22.34	22.96	26.09	56.71	58.28
	SD	9.64	9.48	8.73	9.43	14.84	11.07
15 tasks	Avg.	64.37	39.53	39.37	41.87	83.75	81.09
	SD	10.00	8.73	8.15	9.12	12.66	11.03
20 tasks	Avg.	84.84	65.31	64.21	64.84	116.71	116.71
	SD	10.23	11.62	11.72	10.73	15.45	15.93

116.71 ms, as shown in Table 6. Thus, the maximum number of tasks to be scheduled, 20, takes at most 116.71 ms to schedule. This scheduling time equals approximately 0.29 % of the completion time of a single task. Proposed algorithms complete the scheduling process in a reasonable time.

4 | CONCLUSION

In this study, we propose task scheduling algorithms based on metaheuristic and hybrid metaheuristic approaches for tasks running in the cloud. A variable-length sliding window mechanism is incorporated to schedule the tasks arriving over a recent period. In addition, a greedy approach is used when the number of tasks to be scheduled is equal to or less than the number of virtual machines during a scheduling phase. Experimental results are performed to obtain the average completion time, the average standard deviation of completion times, and the average scheduling time.

According to experimental results, for average completion time, DESA hybrid algorithm outperforms other algorithms for larger task groups (200, 300, 400, 500) while DE and GA, single metaheuristics, give better results for smaller task groups (50,100). DE performs slightly better than GA for the average completion times for larger task groups (200, 300, 400, 500). Because of this, the DESA hybrid algorithm also outperforms other algorithms for the same task groups. Considering the average standard deviation of virtual machine loads, DESA gives better results for 100, 300, 500 task groups while the Greedy algorithm gives better results for 50, 200, 400 task groups. We conclude that DESA and GASA hybrid algorithms improve the results of single metaheuristics regarding results for average completion time. DESA gives better results for larger tasks in both evaluation metrics. Regarding average scheduling time, the scheduling process for the maximum number of tasks at once is completed in a reasonable time compared with the execution time of a single task. So, proposed algorithms can perform the scheduling process in an acceptable period.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable. The data used in this research are generated randomly as described in the paper.

ORCID

Merve Nur Aktan https://orcid.org/0000-0003-3931-9580
Hasan Bulut https://orcid.org/0000-0002-4872-5698

REFERENCES

1. Singh M, Paul S, Kumar A. Task scheduling in cloud computing: review. *Int J Comput Sci Inf Technol*. 2014;5(6):7940-7944.
2. Wadhonkar A, Theng D. A survey on different scheduling algorithms in cloud computing. Paper presented at: Proceedings of the 2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB); 2016:665-669; Institute of Electrical and Electronics Engineers Inc; Chennai, India. https://doi.org/10.1109/AEEICB.2016.7538374.
3. Kalra M, Singh S. A review of metaheuristic scheduling techniques in cloud computing. *Egypt Inform J*. 2015;16(3):275-295. https://doi.org/10.1016/j.eij.2015.07.001.
4. Elzeki O, Rashad M, Elsoud M. Overview of scheduling tasks in distributed computing systems. *Int J Soft Comput Eng*. 2012;2(3):470-475.
5. Prajapati HB, Shah VA. Scheduling in grid computing environment. Paper presented at: Proceedings of the 2014 4th International Conference on Advanced Computing & Communication Technologies; 2014:315-324; Institute of Electrical and Electronics Engineers Inc; Rohtak, India. https://doi.org/10.1109/ACCT.2014.32.
6. Obali M, Topcu AE. Comparison of cluster, grid and cloud computing using three different approaches. Paper presented at: Proceedings of the 2015 23nd Signal Processing and Communications Applications Conference (SIU); 2015:192-195; Institute of Electrical and Electronics Engineers Inc; Malatya, Turkey. https://doi.org/10.1109/SIU.2015.7130446.
7. Aktas MS. Hybrid cloud computing monitoring software architecture. *Concurr Comput Pract Exper*. 2018;30(21);e4694. https://doi.org/10.1002/cpe.4694.
8. Nacar MA, Aktas MS, Pierce M, et al. VLab: collaborative Grid services and portals to support computational material science. *Concurr Comput Pract Exper*. 2007;19(12):1717-1728. https://doi.org/10.1002/cpe.1199
9. Fox G, Mustacoglu AF, Topcu AE, Cami A. SRG: a digital document-enhanced service oriented research grid. Paper presented at: Proceedings of the 2007 IEEE International Conference on Information Reuse and Integration; 2007:61-66; Las Vegas, NV, USA. https://doi.org/10.1109/IRI.2007.4296598.
10. Demir I, Sayar A. Hadoop optimization for massive image processing: case study face detection. *Int J Comput Commun Control*. 2014;9(6):664-671. https://doi.org/10.15837/ijccc.2014.6.285
11. Kamburugamuve S, Wickramasinghe P, Govindarajan K, et al. Twister:Net - communication library for big data processing in HPC and cloud environments. Paper presented at: IEEE 11th International Conference on Cloud Computing (CLOUD): 2018;1:383-391; San Francisco, CA, USA. https://doi.org/10.1109/CLOUD.2018.00055.
12. Abeykoon V, Fox G, Kim M, et al. Stochastic gradient descent-based support vector machines training optimization on big data and HPC frameworks. *Concurr Comput Pract Exper*. 2021. https://doi.org/10.1002/cpe.6292.
13. Yildiz B, Fox GC. Toward a modular and efficient distribution for web service handlers. *Concurr Comput Pract Exper*. 2013;25(3):410-426. https://doi.org/10.1002/cpe.2854.
14. Yildiz B, Fox G, Pallickara S. An orchestration for distributed web service handlers. Paper presented at: Proceedings of the 2008 3rd International Conference on Internet and Web Applications and Services; 2008:638-643; Athens, Greece. https://doi.org/10.1109/ICIW.2008.55.

15. Fox GC, Aktas MS, Aydin G, et al. Grids for real time data applications. Paper presented at: Proceedings of the International Conference on Parallel Processing and Applied Mathematics; Vol. 3911; 2006:320-332; LNCS, Springer, Berlin/Heidelberg, Germany.
16. Fox G, Aydin G, Bulut H, et al. Management of real-time streaming data Grid services. *Concurr Comput Pract Exper*. 2007;19:983-998.
17. Russel S, Norvig P. *Artificial Intelligence: A Modern Approach*. 3. Harlow, UK; Pearson Education; 2009.
18. Singh P, Dutta M, Aggarwal N. A review of task scheduling based on meta-heuristics approach in cloud computing. *Knowl Inf Syst*. 2017;52(1):1-51. https://doi.org/10.1007/s10115-017-1044-2.
19. Bittencourt LF, Goldman A, Madeira ERM, da Fonseca NLS, Sakellariou R. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*. 2018;30:31–54. http://dx.doi.org/10.1016/j.cosrev.2018.08.002.
20. Ebadifard F, Babamir SM. A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment. *Concurr Comput Pract Exper*. 2018;30(12):e4368. https://doi.org/10.1002/cpe.4368.
21. Ge J, He Q, Fang Y. Cloud computing task scheduling strategy based on improved differential evolution algorithm. Paper presented at: Proceedings of the AIP Conference Proceedings; vol. 1834, 2017:040038; American Institute of Physics Inc; Manchester, UK. https://doi.org/10.1145/3421766.3421785.
22. Tsai CW, Huang WC, Chiang MH, Chiang MC, Yang CS. A hyper-heuristic scheduling algorithm for cloud. *IEEE Trans Cloud Comput*. 2014;2(2):236-250. https://doi.org/10.1109/tcc.2014.2315797.
23. Gan GN, Huang TL, Gao S. Genetic simulated annealing algorithm for task scheduling based on cloud computing environment. Paper presented at: Proceedings of the 2010 International Conference on Intelligent Computing and Integrated Systems; 2010:60-63; Guilin, China. https://doi.org/10.1109/ICISS.2010.5655013.
24. Ge J, Cai Y, Fang Y. Cloud computing task scheduling strategy based on differential evolution and ant colony optimization. Paper presented at: Proceedings of the 2010 International Conference on Intelligent Computing and Integrated Systems; Vol. 1967, 2018:040029; American Institute of Physics Inc; Busan, South Korea. https://doi.org/10.1063/1.5039103.
25. Kamalinia A, Ghaffari A. Hybrid task scheduling method for cloud computing by genetic and de algorithms. *Wirel Pers Commun*. 2017;97(4):6301-6323. https://doi.org/10.1007/s11277-017-4839-2.
26. Goldberg DE. *Genetic Algorithms in Search, Optimization and Machine Learning*. Vol 22. Boston, MA, USA; Addison-Wesley Longman Publishing; 1989.
27. Zomaya AY, Teh YH. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans Parallel Distrib Syst*. 2001;12(9):899-911. https://doi.org/10.1109/71.954620
28. Cheng W, Shi H, Yin X, Li D. An elitism strategy based genetic algorithm for streaming pattern discovery in wireless sensor networks. *IEEE Commun Lett*. 2011;15(4):419-421. https://doi.org/10.1109/LCOMM.2011.022411.101804
29. Ahn CW, Ramakrishna RS. Elitism-based compact genetic algorithms. *IEEE Trans Evol Comput*. 2003;7(4):367-385. https://doi.org/10.1109/TEVC.2003.814633
30. Miller BL, Goldberg DE. Genetic algorithms, tournament selection and the effects of noise. *Evol Comput*. 1996;4(2):113-131. https://doi.org/10.1162/evco.1996.4.2.113
31. Karaboga D, Ökdem S. A simple and global optimization algorithm for engineering problems: differential evolution algorithm. *Turk J Elect Eng Comput Sci*. 2004;12(1):53-60.
32. Schmidt H, Thierauf G. A combined heuristic optimization technique. *Advances in Engineering Software*. 2005;36(1):11–19. http://dx.doi.org/10.1016/j.advengsoft.2003.12.001.
33. Onan A, Korukoğlu S, Bulut H. A multiobjective weighted voting ensemble classifier based on differential evolution algorithm for text sentiment classification. *Expert Syst Appl*. 2016;62:1-16. https://doi.org/10.1016/j.eswa.2016.06.005
34. Onwubolu G, Davendra D. Scheduling flow shops using differential evolution algorithm. *Eur J Oper Res*. 2006;171(2):674-692. https://doi.org/10.1016/j.ejor.2004.08.043
35. The cloud computing and distributed systems lab. CloudSim; 2017.

How to cite this article: Aktan MN, Bulut H. Metaheuristic task scheduling algorithms for cloud computing environments. *Concurrency Computat Pract Exper*. 2021;e6513. https://doi.org/10.1002/cpe.6513