Neural Computing and Applications
https://doi.org/10.1007/s00521-021-06002-w

ORIGINAL ARTICLE

Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing

Poria Pirozmand[1] · Ali Asghar Rahmani Hosseinabadi[2] · Maedeh Farrokhzad[3] · Mehdi Sadeghilalimi[2] · Seyedsaeid Mirkamali[4] · Adam Slowik[5]

Received: 5 September 2020 / Accepted: 31 March 2021

Abstract
The cloud computing systems are sorts of shared collateral structure which has been in demand from its inception. In these systems, clients are able to access existing services based on their needs and without knowing where the service is located and how it is delivered, and only pay for the service used. Like other systems, there are challenges in the cloud computing system. Because of a wide array of clients and the variety of services available in this system, it can be said that the issue of scheduling and, of course, energy consumption is essential challenge of this system. Therefore, it should be properly provided to users, which minimizes both the cost of the provider and consumer and the energy consumption, and this requires the use of an optimal scheduling algorithm. In this paper, we present a two-step hybrid method for scheduling tasks aware of energy and time called Genetic Algorithm and Energy-Conscious Scheduling Heuristic based on the Genetic Algorithm. The first step involves prioritizing tasks, and the second step consists of assigning tasks to the processor. We prioritized tasks and generated primary chromosomes, and used the Energy-Conscious Scheduling Heuristic model, which is an energy-conscious model, to assign tasks to the processor. As the simulation results show, these results demonstrate that the proposed algorithm has been able to outperform other methods.

Keywords Cloud computing · Genetic algorithm · Scheduling duration · Task · Resource · Energy consumption

✉ Adam Slowik
aslowik@ie.tu.koszalin.pl

Poria Pirozmand
poria@neusoft.edu.cn

Ali Asghar Rahmani Hosseinabadi
ark838@uregina.ca

Maedeh Farrokhzad
farrokhzad@gmail.com

Mehdi Sadeghilalimi
msv368@uregina.ca

Seyedsaeid Mirkamali
S.Mirkamali@pnu.ac.ir

[1] School of Computer and software, Dalian Neusoft University of Information, Dalian 116023, China

[2] Department of Computer Science, University of Regina, Regina, Canada

[3] Department of Computer Science, University of Science and Technology of Mazandaran, Behshahr, Iran

[4] Department of Computer Engineering and IT, Payame Noor University (PNU), Tehran, Iran

[5] Department of Electronics & Computer Science, Koszalin University of Technology, Koszalin, Poland

1 Introduction

Cloud computing is explained as "a structure for providing comfort, necessary network permission to use a common pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be quickly supplied and released with the smallest amount of supervision attempt or service contributor operation." [1, 2]. Cloud computing presents diverse services to the clients; however, there are some problems in these networks, and task scheduling is one of the significant issue of them [3, 4]. Through task scheduling, we can increase the throughput of the system and manage our tasks to be processed [5].

Task scheduling is one of the most significant issues in many pieces of research works, and the main goal of scheduling is to map several tasks to proper processors so that it could optimize one or more objectives at an acceptable time [6, 7]. Owing to large solution space, scheduling is categorized as an NP-hard problem, and consequently, it needs time for finding an optimal answer.

Published online: 18 April 2021

In general, the task scheduling problem could be regarded and modeled as a version of the Traveling Salesman Problem (TSP) [8, 9] or Vehicle Routing Problem (VRP) [10, 11]. In the TSP, a salesman must travel through all the cities and return to the city of origin, provided that it passes through the city once and travels the shortest distance [8]. In the VRP, there is also a vehicle that must meet all customers or cities and meet their demand, provided that customer demand should not exceed the capacity of the vehicle and should also travel the shortest possible distance [10]. While both TSP and VRP are NP-hard problems, based on the proves given in [12], task scheduling is a strongly NP-hard problem that could be solved using a metaheuristic algorithm such as the proposed GA-based algorithm.

Task scheduling causes prioritization of given tasks in a specific time, and it tries to achieve higher Quality of Service (QoS) [13, 14]. It also inclines to satisfy some constraints in the problem and optimizes one or more objective functions [15, 16]. The primary aim is to build schedules that could process tasks and allocate them to the existing processors. Since we have limited resources, we must schedule tasks so as we can meet all of the requirements of the system [17].

Recently, task scheduling in Cloud Computing Systems has been researched in a large-scale [15]. These kinds of systems are widely used to process the tasks very fast, and they are designed to meet the diverse computational user needs [18]. In task scheduling systems, the set of tasks can be divided into smaller subtasks so that they could process in parallel. These smaller subtasks almost always have their constraints and dependencies, in which some subtasks must be run before a specific subtask [18].

This method has several advantages, namely reduction of makespan and total run time. By dividing the computation process into smaller subtasks, these two parameters can likely be lessened. Therefore, the primary objective of task scheduling is to allocate subtasks to free processors to diminish makespan and satisfy our preference constraints [19]. This allocation is a challenging task in the cloud computing systems, and assigning tasks to processors in the best optimal state is essential in these systems. Several scheduling algorithms have been proposed to cut down makespan and energy consumption for parallelizing the subtasks with precedence relationships. These links are demonstrated as a Directed Acyclic Graph (DAG), and it is comprised of vertices that represent computations and directed edges that represent the dependencies between those vertices. The DAG graph is a directional graph and does not have a path whose beginning and end are the same and one of its important applications is in routing algorithms [20].

Classical research about task scheduling has concentrated on heuristic methods, such as Heterogeneous Earliest Finish Time (HEFT) [21] and Critical Path on A Processor (CPOP) [21]. The main basis of the listed schedule is the maintenance of an ordered list of subtasks by assigning priority to each subtask in accordance with greedy heuristics. The subtasks are chosen and sorted according to the priority and subtask with the highest priority assigns to a processor, which permits the earliest start time, and then it removes from the list [21]. As it is obviously understandable, the efficiency of these algorithms highly relies on the performance of the heuristics. They may not create trustable answers for various problems, particularly when complexity increases, their performance will reduce [21].

Also, there are several metaheuristic methods for solving task scheduling problems, and they attain near-optimal solutions within a rational time. In general, mapping tasks onto limitless computing resources in cloud computing is for a classification of problems recognized as NP-hard problems [22]. There are no algorithms which may make optimal solution within the polynomial time for these issues. Metaheuristic techniques such as Ant Colony Optimization (ACO) [23], Genetic Algorithm (GA) [24], and Particle Swarm Optimization (PSO) [25] could create answers which improve some criteria like energy consumption, run time, etc. Metaheuristics have been prevalence recently, principally by reason of its efficiency and efficacy in solving large and intricate issues [15]. On the other hand, there is a guided-random-search-based algorithm that incorporates a combinatoric process in the search for solutions, and it is less effective and has a higher computational cost in comparison with the heuristic-based algorithms. Hence, a proper scheduling algorithm should balance makespan against the speed of convergence.

The task scheduling problem can be expressed as the search for an optimal allocation of a set of subtasks onto a set of processors. It has been proven that this problem is an NP-hard problem [22, 26]. Therefore, heuristic methods could be used to find reasonable solutions concerning problem constraints, and it can find a near-optimal solution in polynomial time [2, 27].

In this paper, we solve the problem of multi-objective task scheduling in cloud computing using a Genetic Algorithm and Energy-Conscious Scheduling Heuristic (GAECS). The purpose of solving the above problem is to provide an effective and optimal way to schedule tasks with the aim of reducing makespan and energy consumption.

In the following, the structure of the paper is organized as follows: In Sect. 2, the related works about task scheduling in cloud computing are discussed. In Sect. 3, the proposed algorithm for solving the above problem is

fully explained. The simulation and conclusion results are shown in Sects. 4 and 5, respectively.

2 Related works

In recent years, several types of research have been done to schedule tasks in cloud computing systems, some of which we will discuss as follows. Each of these methods has its benefits and drawbacks.

Awada et al. in 2015 [28] proposed a model with a PSO in the opinion of schedule and allocation for cloud computing that takes into account trustworthiness, run time, makespan, round trip time, transmission cost and load stabilizing between tasks and Virtual Machine (VM). They reduced execution time and increased environment reliability by contemplating the resources unoccupied and reschedule tasks that failed to assign.

A multiple objective task scheduling technique for a VM was proposed by Lakraa and Yadav [29] and considered different measures such as run time, cost, the bandwidth of client, and so forth despite single criteria.

In [30], a Dynamic Tasks Scheduling algorithm based on the Weighted Bi-graph model (DTSWB) is introduced. This method consists of four parts and, compared to other scheduling schemes, has better efficiency.

Kashikolaei et al. [31] presented a new hybrid algorithm based on the Imperialist Competitive Algorithm (ICA) and the Firefly Algorithm (FA) to solve the problem of scheduling tasks in cloud computing. The authors introduced test data to evaluate the proposed algorithm (ICAFA) and compared it with five examples of other algorithms. The simulation results of the ICAFA and the compared algorithm show that the algorithm presented by the authors has been able to solve the problem well and compete with the compared algorithms. It is noteworthy that the authors simulated various parameters such as Makespan, load balancing, stability, CPU time, and efficiency to show the performance of the ICAFA and compared their simulation results with other algorithms and as mentioned, the ICAFA was able to compete with comparable algorithms and solve the problem properly. It is noteworthy that the ICAFA has been able to reach the optimal answer in less time than the compared algorithms.

Zhu et al. [32] focused on computation energy in cloud computing systems and tried to balance computation energy minimization and user-defined deadlines. They proposed a heuristic method that included a task sequencing method and a virtual machine searching strategy. Results show that it outperforms the other algorithms.

Alworafi [33] considered the load on resources and proposed a Hybrid-SJF-LJF (HSLJF) algorithm, which synthesizes the Shortest Job First (SJF) and Longest Job First (LJF) algorithms. First, this algorithm sorts the tasks in ascending order, and then, it chooses one task. Eventually, it chooses a VM that has the least accomplishment time to run the selected task. The outcomes confirmed the advantage of this method in diminishing the makespan and response time.

In [34], a discrete edition of the PSO algorithm called Integer-PSO is proposed, and it can order tasks in the cloud computing environment and can be utilized for optimizing a single target function and multiple target functions. Practical findings show that this strategy has better convergence and load balancing. To show the superiority of the Integer-PSO algorithm presented by the authors, the above algorithm has been compared with the two algorithms rounding-off (RND-PSO) and smallest position value (SPV-PSO). Based on the simulation results, the integer-PSO algorithm has been able to excel in terms of cost-optimal scheduling and makespan in comparison with two other algorithms and solve the problem well. To show this superiority, the authors tested the algorithms with different tasks 512, 1024, and 2048.

Reddy and Kumar [35] presented a modified ACO algorithm to improve task scheduling. Their main aim was to diminish makespan and design a multi-objective task scheduling algorithm with high functioning. They could reduce the makespan and balance workload in the network without any effect on other fundamental parameters. In this algorithm, all ants are presented on processors by chance. Next, ant creates pheromone, and according to the pheromone values, ants are permitted to move.

Abd Elaziz et al. [36] proposed a metaheuristic algorithm based on the boost of the Moth Search Algorithm (MSA) with the Differential Evolution (DE). MSA is a nature-inspired method for the behavior of moths to fly to the light source and uses the phototaxis for exploration and levy flights for exploitation. Since exploitation ability had some drawbacks, DE dealt with a Local Search (LS) method. The results of the experiments showed that it performed better than other ways on grounds of more acceptable performance measures.

Sanaj and Prathap [37] suggested a Chaotic Squirrel Search Algorithm (CSSA) optimize multi-objective task scheduling in a cloud atmosphere. The methods continuously work to be more profitable, and to support greater global convergence, the early ecosystem was produced with untidy optimization for an competent ecosystem. The recommended CSSA was eventually synthesized with the messy LS to enable the exploring authority to complement Squirrel Search (SS) algorithm.

Another nature-inspired metaheuristic algorithm for task scheduling is Crow Search Algorithm (CSA) [19]. As it is obvious, it is inspired by the behavior of crow and the food-collecting habits of it. In nature, the crow usually follows

other crows to discover a better food source. Similarly, the CSA algorithm tries to find a proper VM to cut down makespan. In comparison with other methods, this method could reduce the makespan value.

Sobhanayak et al. [20] presented a combined method using GA and the Bacterial Foraging (BF) algorithms in the computing cloud. This algorithm has two main objectives. First, it minimizes the makespan, and then, it decreases the use of energy. The simulation results show the primacy of this algorithm over other algorithms. There is a Hybrid Artificial Bee Colony and Ant Colony Optimization (HABCACO) load balancing algorithm, which has some advantages of these algorithms. For instance, it used the speed of the ACO algorithm in finding good solutions swiftly and shared interaction of bees and sharing information by waggle dancing from Artificial Bee Colony (ABC). Simulation results showed a good improvement in makespan, execution time, and so on [38].

Amalarethinam and Kavitha [39] proposed a Rescheduling Enhanced Min-Min (REMM) algorithm for meta-task scheduling in cloud computing. The speed of resources in the cloud system is the most significant measure in this method. The resources are sorted according to their velocity, and a task with a smaller amount of run time is to be designated and allocated to the fastest resources. Then, it removes from task set.

Huang et al. [40] solved the problem of scheduling tasks in cloud computing. They used the PSO algorithm to solve the above problem. Their proposed algorithm has been compared with three other optimization algorithms, and they have shown in the simulation results that the algorithm provided by them has a relative advantage over the compared algorithms.

In [41], the authors present a new optimization method based on the Whale Optimization Algorithm (WOA) to solve the problem of assigning tasks in Mobile Cloud Computing (MCC). They considered several criteria to solve this problem, one of which is the reduction of energy in mobile cloud computing. The simulation results of their proposed method show that the makespan and energy consumption in the mobile cloud computing environment have been optimal and have performed very well compared to the compared methods and have been able to solve the problem well.

Basu et al. [42] presented a hybrid algorithm called GAACO to solve the problem of scheduling tasks in cloud computing for IoT applications. The algorithm presented at each step uses the best combination of tasks and avoids getting stuck in the local optimal. The algorithm provided is a combination of GA and ACO and is compatible in environments with multiple heterogeneous processors.

Alwonafi et al. [43] developed a Deadline Budget Scheduling (DBS) model for scheduling tasks in cloud computing that they also considered virtual machines. They suggested that tasks be assigned to virtual machines and that the limitations offered in the proposed method were ultimately considered at the request of users until user satisfaction was met. In the method provided by the authors, makespan, and the cost to evaluate the above model are calculated. The results show that the proposed DBS model has been able to be more cost-effective than other algorithms and reduce costs.

Raju and Devarakonda [44] used a fuzzy clustering method along with the ACO algorithm to solve the task of scheduling tasks in cloud computing. In their paper, they were able to optimize the load balance for the problem and also schedule tasks well and provide them to users.

2.1 Problem statement

The cloud environment is a complicated system with many shared resources and unpredictable requests and is influenced by uncontrollable external events. The machines are also located in cloudy environments in different areas and have different processing capabilities and specifications and different costs. In this case, the cost and duration of the schedule and the resources allocated are important and cannot be neglected. Therefore, in order to provide an optimal schedule, there must be coordination between the task schedule and allocation of resources. In this way, we can achieve an optimal schedule by reducing costs (the cost considered in this paper is the price of energy consumed) and the duration of the schedule. Multi-objective optimization for cloud resources requires complex policies and decisions. Among the important issues in the cloud computing, energy consumption and makespan are the most important ones. Therefore, we must be able to achieve the optimal schedule in time and energy usage by balancing the desired goals.

2.2 System model

The cloud computing network discussed here is composed of a set p of m diverse processes that are altogether connected with a high-speed network. Each processor is Dynamic Voltage Scaling (DVS) enabled; in other words, each processor is capable of working with various voltages scales as a set v which can set with regard to the input task. As clock frequency transition overheads take an insignificant amount of time (about $10\mu s - 15\mu s$) [45], these overheads are not taken into consideration in our paper, and the inter-processor communications are conducted with the same speed on all links.

3 The proposed algorithm

Parallel programs have implementation priority, including programs utilized in technological and engineering scopes. These programs can be established in homogeneous or heterogeneous systems such as cloud infrastructure. The concept of cloud computing is derived from heterogeneous distributed computations, greed computations, useful computations, and automated computations. Users of cloud systems do not own any part of the infrastructure and can easily access the services provided through the internet and pay for the service used, and such a system can provide any kind of service to its users, including computing resources, web services, social networks and communication services. Many specifications indicate the quality of the services provided, and the highlights include the cost of producing the service and, ultimately, the cost the user has to pay, makespan, and the energy expended to perform the service. In this paper, the proposed algorithm (GAECS), which is a combination of GA and ECS model, is a time and energy-conscious method for scheduling tasks in cloud computing systems and includes two steps. Since tasks have priority and must be performed in a specific order, we used the GA to prepare the appropriate sequence for performing tasks based on their priority. Therefore, the first phase of the our method is dependent upon the GA, and the role of the GA is to generate the chromosome genes. In fact, the GA completes the tasks of $T_1, T_2, \ldots, T_n$. At this stage, the tasks are based on their genes. In order to make optimal use of resources and reduce energy consumption, the ECS method, which is a conscious method of energy, has been used to assign tasks to processors. In fact, in the second phase, the processor part and the gene voltage are completed. Whenever the first part of the chromosome is complete, the ECS pattern is called. In other words, ECS completes the remaining parts of the chromosome, namely $P(T_1), P(T_2), \ldots, P(T_n)$ and $V(T_1), V(T_2), \ldots, V(T_n)$. Whenever the working part, the processor, and the voltage of the genes are completed, the function of the GA evaluator is called, and the role of this operator is to compute the energy usage and the schedule duration of each chromosome. In the mutation, the genes are changed so that their priority is not violated. In practice, the combination of the two parents is chosen at random and is divided into left and right. The child consists of the right part of the two parents.

3.1 GAECS for task scheduling in cloud computing

In this paper, we use the GA to find a solution to scheduling. In the following, the algorithm operators, including chromosome display, initial population generation, selection, crossover, mutation, are fully explained.

In the GA, the variables are coded into elements called genes, and the answers to the problem are strings of genes, each of which is called a chromosome. The elements in this paper are the tasks related to a program that is arranged on the chromosome according to their priority. In each iteration of the algorithm, the value of genes can be changed. Also, by applying mutation and crossover operators, the value of genes is changed again, and a new chromosome (scheduling) is obtained.

Table 1 shows a sample of a chromosome for an eight-function program and three processors. It shows the information about the makespan on each processor and the energy consumption. For instance, in this chromosome, the task of T_1 is on the P_2 processor and V_3 voltage.

3.2 Initial population

To improve the task speed and achieve the desired result, we choose the initial population as smartly as possible, in fact, to produce the initial population, we act as follow:

First, using the three prioritization methods, including the upward priority, the downward priority, and the combination of these two methods, the first three chromosomes are produced, and the rest of the initial population chromosomes are replaced by these three chromosomes.

Upward priority The upward priority of a task is equal to the average cost remaining until all tasks related to the current task are completed, which must be performed after the completion of the current task. The upward priority is shown as $Rank_b(T_i)$ and is calculated based on Eq. 1.

$$Rank_b(T_i) = \overline{W(T_i)} + max\left(\overline{C(T_i + T_j)} + Rank_b(T_j)\right) \tag{1}$$

In Eq. 1, T_i and T_j represent the tasks of i and j, respectively. W is the weight of each task. C is the capacity of each task. $Rank_b$ shows that which task is performed sooner or later.

In this Equation, $T_j \in Succ(T_i)$ and $Succ(T_i)$ are sets of successor tasks of T_i. The upward priority of the graph starts the tasks from the last task (T_{exit}) and upward.

Table 1 Chromosome sample with eight tasks on three processors

T_1	T_2	T_4	T_5	T_3	T_6	T_7	T_8
P_2	P_1	P_2	P_3	P_3	P_1	P_2	P_3
V_3	V_2	V_1	V_1	V_2	V_1	V_2	T_3

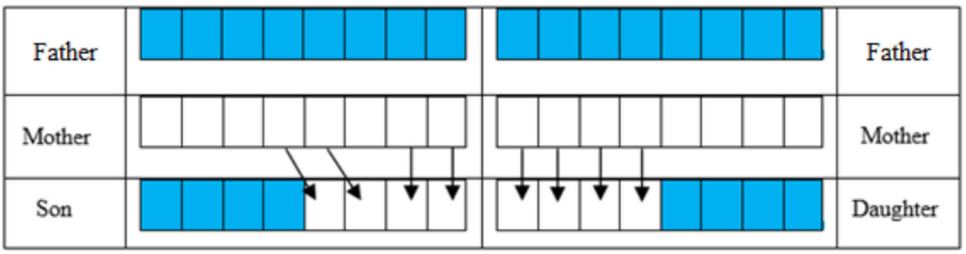

Fig. 1 Crossover operation

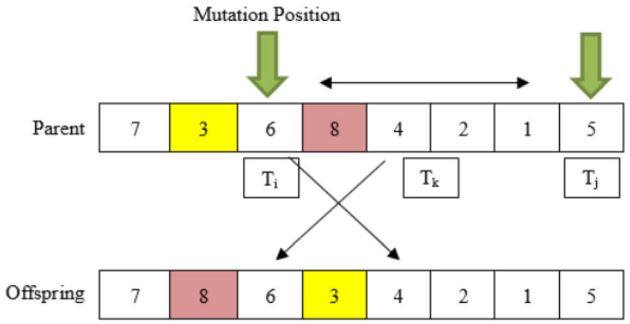

Fig. 2 Mutation operation

Downward priority Similar to the upward priority, the downward priority of tasks is shown as $Rank_t(T_i)$ and is calculated as Eq. 2.

$$Rank_t(T_i) = max\left(\overline{W(T_i)} + \overline{C(T_j, T_i)} + Rank_i(T_j)\right) \quad (2)$$

In Eq. 2, $T_j \in Pred(T_i)$ and $Pred(T_i)$ are the sums of the predecessor tasks of T_i. The downward priority is calculated by navigating the task graph, starting from the first task (T_{entry}). In this method, the priority of the first task is 0. $Rank_t(T_i)$ is equal to the largest distance from the first task to the desired task (T_i) except for the computational cost of the T_i task.

Combining priorities To schedule a Directed Acyclic Graph (DAG), we can perform tasks based on upward priority or downward priority or a combination of them. In the combination of priorities, one level is defined for each task based on Eq. 3 and it shows the level of the task in the graph.

$$Level(T_i) = \begin{cases} 0, & \text{if } T_i = T_{entry} \\ max\left(Level(T_j) + 1\right), & \text{otherwise} \end{cases} \quad (3)$$

3.3 Parent selection

In the GA, two generations are needed to produce chromosomes in the next generations, and the choice of the parent is based on different algorithms, but the algorithm used to pick the parent here is the roulette wheel selection. After calculating the fitness of each chromosome, chromosomes with a better fitness value with a ratio that is not constant, for example, we can copy 20% of the best members of the original population and enter to the new population. In this way, we give more chances to members with different fitness values. Then, we randomly select the number of pairs we want to combine in the next phase.

3.4 Crossover operator

This operator combines the information of the two selected parents' chromosomes and produces the child's chromosomes. Given how the information used in this issue is displayed, we are looking to produce two children from each combination and try to make one of these two children better than the parents. The crossover operator in our problem is that two parents are selected at random, and in each of them, we select a random point called the crossover point. We insert the first part of the two parents in two separate chromosomes, and the information in the second part of the chromosomes will be similar to the information in the second part of the parent information.

Father and mother mean two parent chromosomes, which are divided into left and right, and the left part is copied directly to the left part of the child's chromosomes, i.e., daughter and son, and the second part of the child's chromosomes is similar to the right part of one of the parents' chromosomes. Figure 1 shows how to perform a crossover operation.

3.5 Mutation operator

This operator ensures population diversity through random changes in gene information and aims to prevent the chromosomes from becoming very similar over several generations. The mutation in our problem is based on the displacement of independent genes at the same level as the graph. In this way, first, a mutation point is selected randomly, and the information of that gene is replaced by the first independent gene (task) after it, and thus a new chromosome is produced. Figure 2 shows how the mutation operation works.

In this Figure, as shown in the example chromosome, we first find the first T_i substitute from the chosen point to the end of the priority queue (T_j) and then exchange T_i with the

first predecessor T_j called T_k. These two colors (yellow and pink) are the point of exchanging of genes.

3.6 Fitness function

In the GA, the fitness function determines the fitness of a solution to the problem and shows the extent to which the constraints and optimization levels are met.

In this case, the fitness of the solution is calculated based on energy consumption and makespan and on the foundation of the fitness function, and the task is gave to the processor and voltage, which has higher fitness.

Due to the fact that a significant part of the energy use of data centers is connected to the processing of computing and storage disks, networks, and cooling systems, the energy consumption considered in this paper is the energy consumption of processors.

As we know, the energy expending of CMOS processor batteries is comprised of static and dynamic consumption, and because dynamic consumption is more important than static, only dynamic consumption is considered in this paper. Thus, by reducing power consumption, we can increase system efficiency.

Power battery consumption is calculated by Eq. 4:

$$P = A \cdot C \cdot V^2 \cdot f \tag{4}$$

In this Equation, A is equal to the number of switches in each clock cycle, C is the capacitance, V is the voltage, and f is equal to the frequency.

Energy consumption is directly related to the square of voltage and frequency. Therefore, reducing these two factors will have a significant effect on reducing energy consumption, and because the frequency is directly related to voltage, the dynamic energy consumption of processors will be in the form of Eq. 5:

$$P = A \cdot C \cdot V^3 \tag{5}$$

And the run time of the i-th task is in the form of Eq. 6:

$$T_i = EST + w_{ij} \tag{6}$$

In this Equation, EST is the starting time of a task, and w is the time required to perform task i on the processor j.

Based on the obtained Equations, the energy use for each operation on each processor is as Eq. 7:

Table 3 Makespan performance comparison for all approaches with 30 tasks

Algorithm	Average	SD	ACT
GSA	1782.8995	94.0909	1.3694
ABC	1657.3083	56.5111	1.1077
DA	1824.5682	130.4172	9.6238
Linear-PSO	1640.4564	76.7758	1.3092
Sigmod-PSO	1651.6592	99.0259	1.2898
Chaotic-PSO	1619.4062	72.0869	1.2992
Simulated-PSO	1601.0727	76.6309	1.2852
Logarithm-PSO	1588.7585	98.883	1.2968
GAECS	1567.3214	54.618	2.7927

Table 4 Makespan performance comparison for all approaches with 50 tasks

Algorithm	Average	SD	ACT
GSA	2871.1385	150.2514	1.7026
ABC	2679.8311	127.0006	1.2681
DA	2926.4281	249.6792	15.2899
Linear-PSO	2619.7364	149.7841	1.3844
Sigmod-PSO	2615.6082	149.8076	1.3651
Chaotic-PSO	2595.0782	136.3308	1.3722
Simulated-PSO	2615.6865	129.9891	1.3514
Logarithm-PSO	2561.1159	111.5794	1.3648
GAECS	2371.5648	108.6831	2.1124

$$E = P \cdot w_{ij} \Rightarrow E = A \cdot C \cdot V^3 \cdot w_{ij} \tag{7}$$

In this Equation, w_{ij} is the time needed to run task i-th on the processor j-th, and the total energy is calculated as Eq. 8:

$$E = P \cdot w_{ij} \Rightarrow E = \sum_{i=1}^{n} A \cdot C \cdot V^3 \cdot w_{ij} \tag{8}$$

And assigning a task to the processor is measured by Eq. 9.

The symmetry of the sum of the energy distinction between the present and former state to the current energy and the ratio of the distinction between the current and previous completion time to the current time difference and

Table 2 Computing power of VMs

VM ID (1-10)	1	2	3	4	5	6	7	8	9	10
Computing power (kBps)	20	25	30	50	55	60	65	66	70	73
VM ID (11-20)	11	12	13	14	15	16	17	18	19	20
Computing power (kBps)	75	80	85	90	35	40	45	50	58	82

Table 5 Makespan performance comparison for all approaches with 100 tasks

Algorithm	Average	SD	ACT
GSA	5688.6096	201.5961	3.0662
ABC	5560.9419	286.2516	1.812
DA	5805.9244	427.2967	35.1305
Linear-PSO	4897.1396	218.0289	1.7437
Sigmod-PSO	5108.8297	111.9476	1.702
Chaotic-PSO	4907.816	175.8923	1.7129
Simulated-PSO	5019.1349	273.517	1.5996
Logarithm-PSO	4838.3595	198.5673	1.6597
GAECS	4611.5380	112.4317	3.7639

Table 6 Makespan performance comparison for all approaches with 200 tasks

Algorithm	Average	SD	ACT
GSA	11702.2816	346.5467	3.9206
ABC	12214.6315	710.2963	1.6917
DA	11010.5597	868.8301	86.3467
Linear-PSO	9647.5628	412.1729	1.8739
Sigmod-PSO	9823.4108	308.8059	1.842
Chaotic-PSO	9536.7212	311.8092	1.8727
Simulated-PSO	9790.0579	314.9119	1.8398
Logarithm-PSO	9463.4511	243.2355	1.8742
GAECS	8136.8792	241.1193	4.4319

Table 7 Makespan performance comparison for all approaches with 300 tasks

Algorithm	Average	SD	ACT
GSA	18307.5988	584.0432	6.7591
ABC	19421.5846	1072.29	2.9392
DA	16892.2548	1488.8774	137.098
Linear-PSO	14244.2165	862.2472	2.1473
Sigmod-PSO	14738.0963	442.2664	1.882
Chaotic-PSO	14352.9043	391.3443	1.9294
Simulated-PSO	14762.6141	741.0269	1.8726
Logarithm-PSO	14185.258	349.483	1.9063
GAECS	14181.320	347.649	4.5843

the minimum current and previous start time should be equal to the maximum value:

$$Fitness = -\left[\frac{E_c - E_p}{E_c} + \frac{EFT_c - EFT_p}{EFT_c - min\left(EST_c - EST_p\right)}\right] \tag{9}$$

In this Equation, the task is first given to the first processor with the initial frequency, and for the next step, that is, for the different frequencies of that processor, it calculates this formula. If this value is better than the previous state, it will assign the task to the processor with the new frequency. In this Equation, E_c equals to current energy, E_p is the previous energy, EFT_c is the current completion time, EFT_p is the previous completion time, EST_c is the current starting time, and EST_p is the previous starting time.

3.7 ECS method

This method is an energy-based, voltage-based method for scheduling tasks that first calculates and compares different allocation modes to schedule tasks and finally assigns the task to the best option (processor and voltage). The purpose of this method is to schedule tasks on processors with the criterion of reducing time and energy consumption. First, the tasks are organized descendingly and for each task, the amount of consumed energy and the time of doing it are calculated on each processor and voltage, and the task is assigned to the processor and voltage, which these two criteria should be minimal [46].

3.8 Termination condition

The condition for ending the algorithm here is to achieve maximum iteration. Whenever the algorithm attains its maximum iteration, the algorithm finishes. The maximum number of iterations in this algorithm is 100.

4 Simulation results

The proposed algorithm (GAECS) is described in the former part, and in this part, we will discuss the results of simulating and comparing the results obtained using proposed algorithm with the results obtained using other methods. As mentioned, we used the GA and the ECS model in the proposed method for optimal resource scheduling and energy saving. In this method, the chromosomes are first sorted according to their priority so that we have optimal primary chromosomes. In practice, the crossover and mutation of genes are based on their priority, and thus their order of execution is never violated. In the process of allocating tasks to the processor, it is also tried to assign tasks to a processor that has both a minimum time and energy factor.

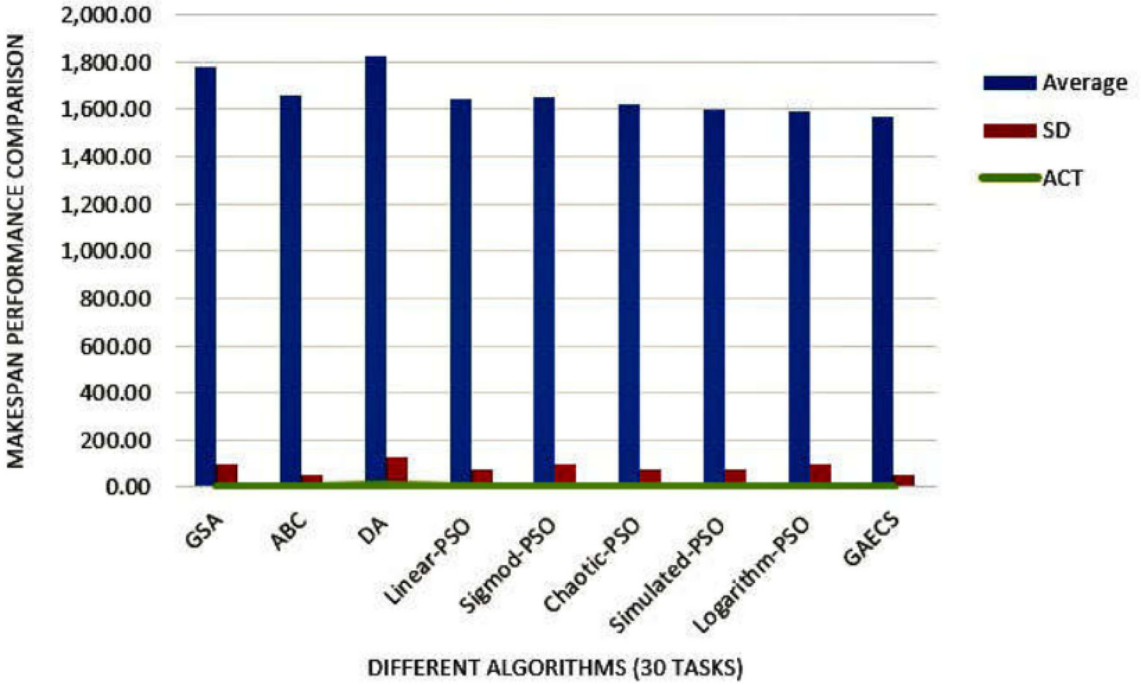

Fig. 3 Makespan convergence curve for all approaches with 30 tasks

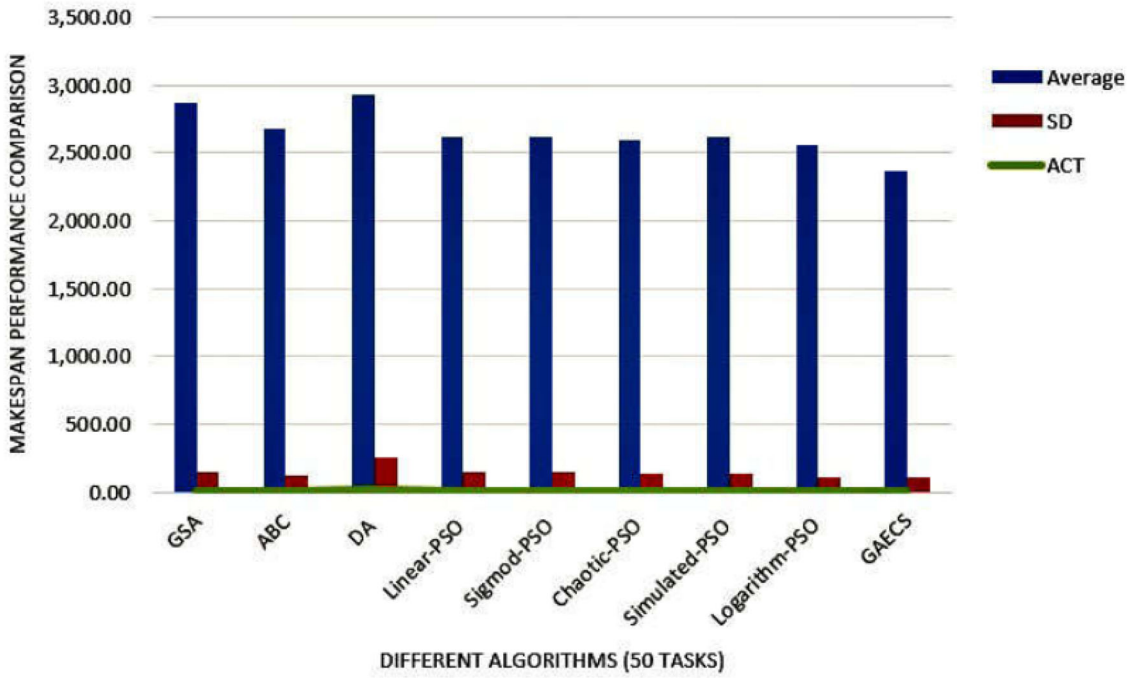

Fig. 4 Makespan convergence curve for all approaches with 50 tasks

Here, we depict the appropriate condition for evaluating and proving the improvement of the proposed method over other methods. In this environment, we show that using the proposed algorithm, we will improve energy and time, and in this environment, we compare our method with other similar methods and show the results.

4.1 Simulation environment

We implemented and evaluated all algorithms using MATLAB 2014a. The operating environment for running was a personal computer with an Intel (R) Core i7 2.4GH processor, 8GB of memory and Windows 10.

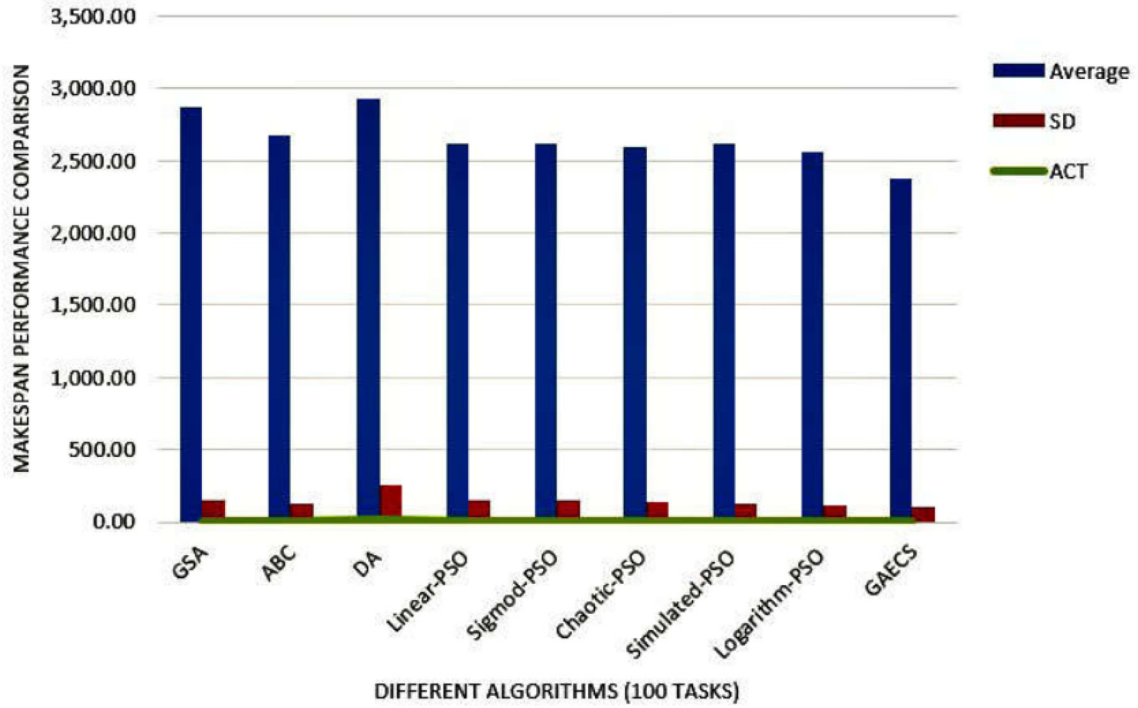

Fig. 5 Makespan convergence curve for all approaches with 100 tasks

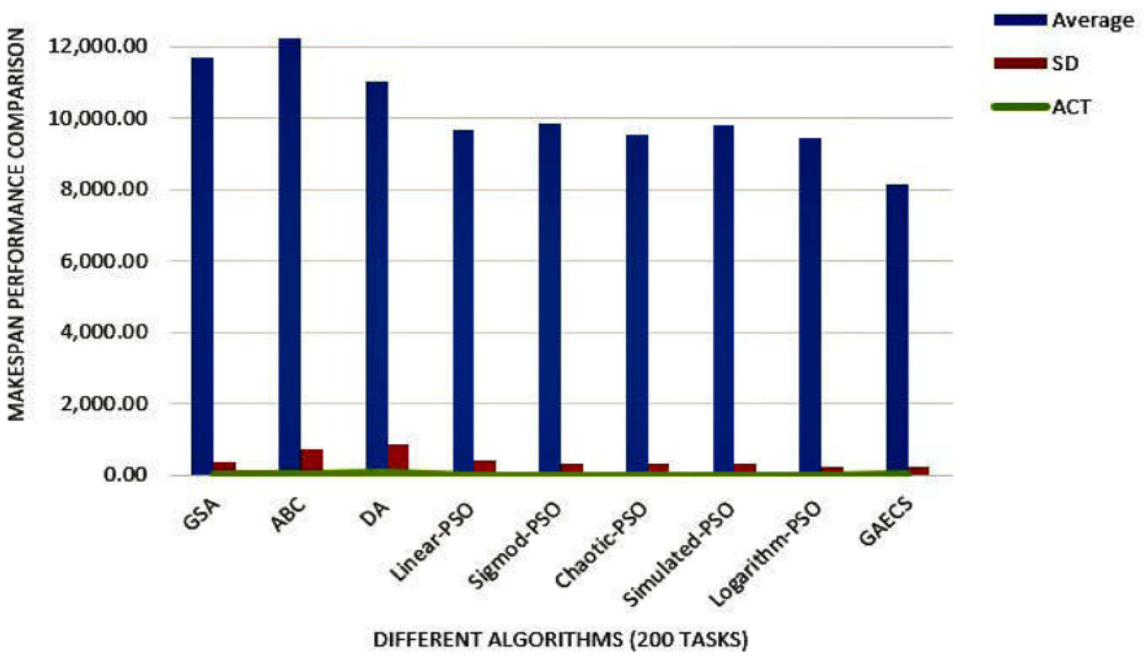

Fig. 6 Makespan convergence curve for all approaches with 200 tasks

In this evaluation, experiments are run with five stochastic datasets of 30, 50, 100, 200 and 300 tasks.

For scheduling the tasks of datasets whose file size is 32 to 64 MB, 20 VMs have been used. Table 2 shows the capabilities of these VMs.

The GAECS is compared with Gravitational Search Algorithm (GSA) [47], ABC [48], Dragonfly Algorithm (DA) [49], Linear-PSO [40], Sigmod-PSO [40], Chaotic-PSO [40] and Simulated-PSO [40], Logarithm-PSO [40], the simulation results of which can be seen in Tables 3,4, 5, 6, 7.

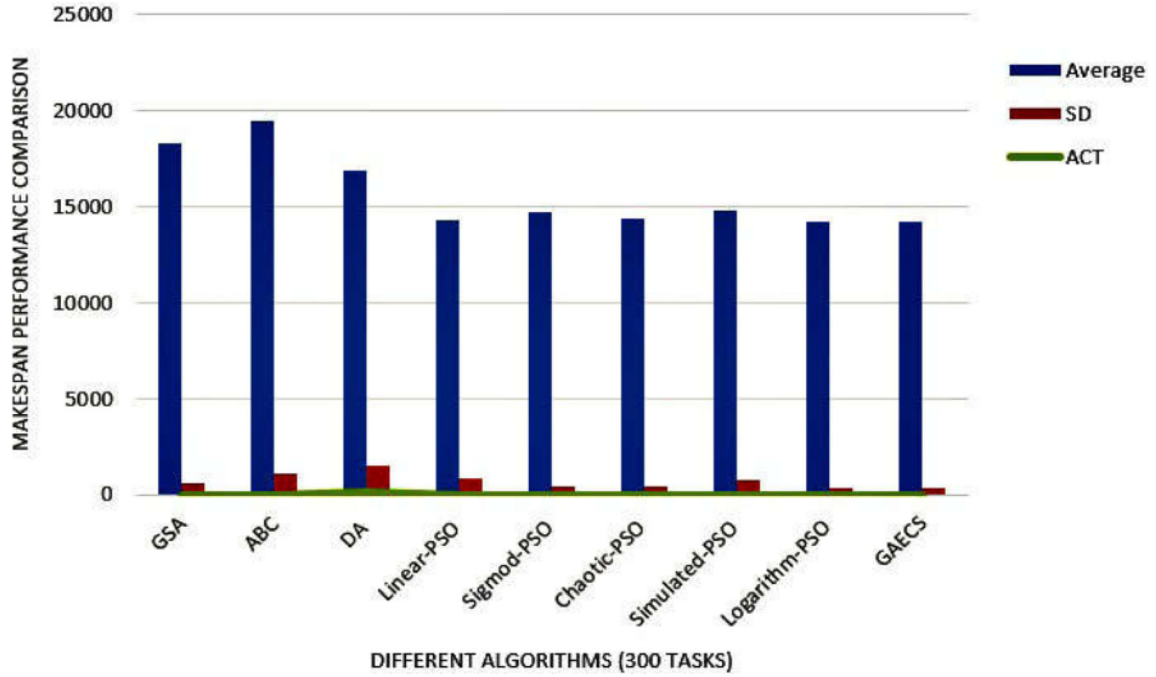

Fig. 7 Makespan convergence curve for all approaches with 300 tasks

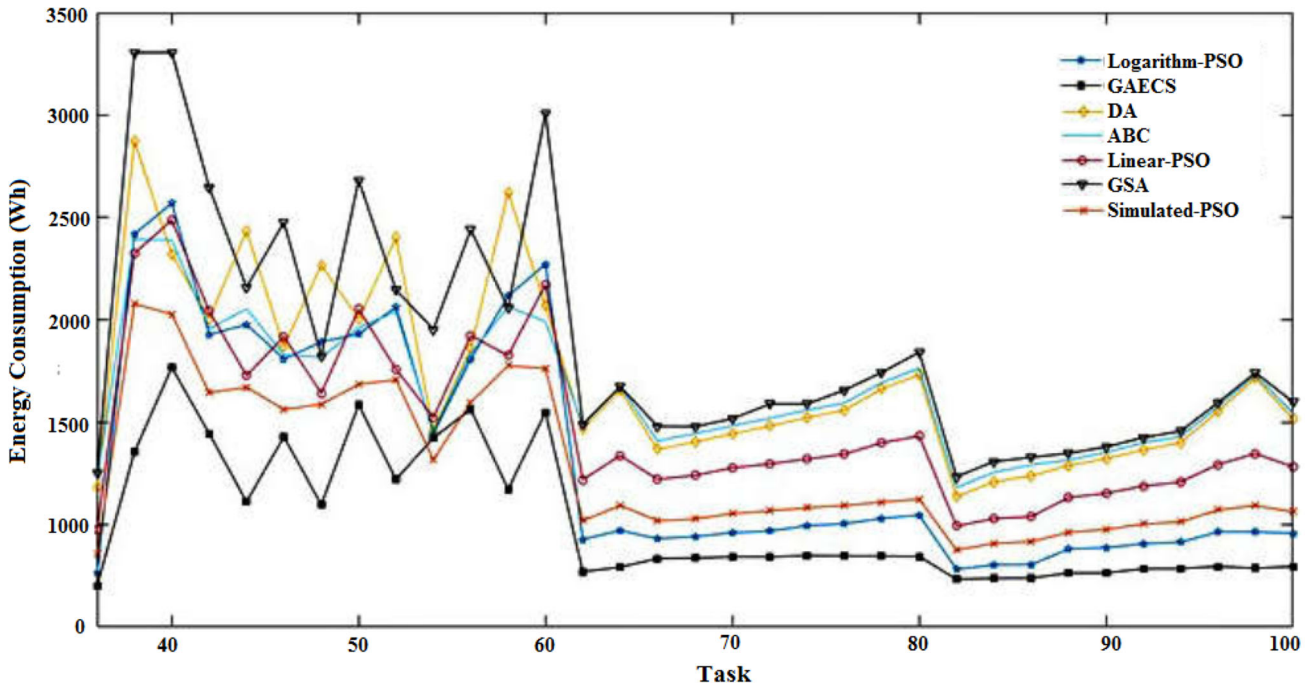

Fig. 8 Comparison of energy consumption of the GAECS with comparable algorithms for the number of tasks from 40 to 100

As you can see, the GAECS has been able to solve problems properly and compete with comparable algorithms. As the GAECS is a hybrid algorithm, it has a higher Average Computation Time (ACT) than the comparable algorithm. All algorithms were run 20 times and then these average results are presented in Tables. In Tables 3, 4, 5, 6, 7, *Average*, *SD* and *ACT* columns represent Average Value (Mean), Standard Deviation (SD) and average computation time, respectively.

Tables 3, 4, 5, 6, 7 show the best position as shown in Fig. 3 for changes to the proposed algorithm (GAECS) and compared algorithms. Comparative algorithms that have

similar differences based on Average, SD, and ACT have been investigated. As can be seen, the GAECS has more optimal conditions to find the Makespan, and these conditions are considered for 30 tasks. As shown, the Linear-PSO and Sigmod-PSO methods acted linearly similar to the GAECS. In Figs. 4, 5, 6, 7, the conditions for the GAECS are improved by increasing the number of tasks. As shown in Figs. 3, 4, 5, 6, 7, GAECS is superior to other algorithms in terms of accuracy. All the compared algorithms perform very well in these cases. PSO-based approaches also work well in all cases. Note that, as the number of tasks increases, the DA computational time increases rapidly.

Figure 8 shows the energy results of the proposed algorithm with other algorithms in watt per hour (Wh). All performance metrics are evaluated for several 40–100 tasks. As can be seen in this figure, the energy increases with increasing tasks in all algorithms. The GAECS algorithm has better conditions for finding energy in different tasks, and this operation is due to the low value of makespan, and these 40 to 100 tasks are considered. As shown, Linear-PSO and Sigmod-PSO methods, such as GAECS, operate linearly. Due to the low computational complexity (time-space), we will have an energy reduction in the proposed method. GAECS is superior to the comparable algorithms in terms of accuracy, and note that as the number of tasks increases, the DA energy computation time increases rapidly [50].

5 Conclusion

Cloud computing system is a prevailing model for clients to use cloud resources due to the "pay for use" model, but due to the high volume of resources and high requests from users, the issue of scheduling and energy use has become one of the important obstacle of this type of system. Much research has been done on this topic. Scheduling is a way to allocate resources to users, and the main goal of a schedule is to reduce makespan. Scheduling is a way to allocate resources to users, and the primary goal of a schedule is to reduce makespan. In most of the methods presented for cloud system scheduling, little attention has been paid to the energy issue, and in some methods that have been optimized for energy consumption, the makespan has increased. For example, the Hybrid GA algorithm is an energy-conscious scheduling method for dependent tasks, but despite the fact that tasks are interdependent, the scheduling and production of primary chromosomes have not been optimally performed. The ECS algorithm does not consider all possible combinations to perform tasks. Therefore, it is better to use scheduling algorithms that can optimize time and energy usage. In this study, a two-step algorithms called GAECS is presented with the aim of reducing time and energy. In the proposed GAECS algorithm, we used the GA to create optimal schedules and three ranking algorithms to create the primary chromosomes. We also used the ECS algorithm, which is an energy-aware technique, to optimize the allocation of resources to processors. In the GAECS algorithm, first three primary chromosomes are produced using three prioritization algorithms, and the primary chromosomes are given to the GA, and the primary population is completed in the GA. Then, using the defined crossover and mutation operators, better chromosomes are selected, and finally, the optimal chromosomes are selected in terms of time and energy and are assigned to resources. The GAECS algorithm is compared to eight algorithms. The results of the evaluation of the GAECS show that the GAECS has a better makespan and energy consumption.

Declaration

Conflict of interest In the present work, we have not used any material from previously published. So we have no conflict of interest.

References

1. Kalra M, Singh S (2015) A review of metaheuristic scheduling techniques in cloud computing. Egypt Inf J 16(3):275–295. https://doi.org/10.1016/j.eij.2015.07.001
2. Prasanna Kumar KR, Kousalya K (2019) Amelioration of task scheduling in cloud computing using crow search algorithm. Neural Comput Appl 32:5901–5907
3. Srichandan S, Ashok Kumar T, Bibhudatta S (2018) Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm. Future Comput Inf J, vol. 3, pp. 210-230
4. Basu S, Karuppiah M, Selvakumar K, Li K, Islam SKH, Hassan MM, Bhuiyan MZA (2018) An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment. Future Generat Comput Syst 88:254–261
5. Gamal M, Rizk R, Mahdi H, Elhady B (2019) Bio-inspired based task scheduling in cloud computing. Mach Learn Paradig: Theor Appl 801:289–308

6. George Amalarethinam DI, Kavitha S (2018) Rescheduling enhanced Min-Min (REMM) algorithm for meta-task scheduling in cloud computing. International Conference on Intelligent Data Communication Technologies and Internet of Things, vol. 26, pp. 895–902
7. Alworafi MA, Mallappa S (2019) A collaboration of deadline and budget constraints for task scheduling in cloud computing. Cluster Comput, pp. 1-11
8. Valarmathi R, Sheela T (2019) Ranging and tuning based particle swarm optimization with bat algorithm for task scheduling in cloud computing. Cluster Comput 22:11975–11988. https://doi.org/10.1007/s10586-017-1534-8
9. Lee YC, Zomaya AY (2009) Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, 2009, pp 92–99. https://doi.org/10.1109/CCGRID.2009.16
10. Mezmaz M, Melab N, Kessaci Y, Lee YC, Talbi EG, Zomaya AY, Tuyttens D (2011) A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J Parallel Distribut Comput 71:1497–1508
11. Shojafar M, Kardgar M, Hosseinabadi AR, Shamshirband S, Abraham A (2016) TETS: A genetic-based scheduler in cloud computing to decrease energy and makespan. The 15th International Conference on Hybrid Intelligent Systems (HIS 2015), Chapter Advances in Intelligent Systems and Computing 420, Seoul, South Korea, vol. 420, pp. 103–115,
12. Polepally V, Chatrapati KS (2017) Dragonfly optimization and constraint measure-based load balancing in cloud computing. Clust Comput 22:1–13
13. Sangaiah AK, Hosseinabadi AR, Shareh MB, Bozorgi Rad SY, Zolfagharian A, Chilamkurti N (2020) IoT resource allocation and optimization based on heuristic algorithm''. Sensors 20(2):1–26
14. Hosseinabadi AR, Farahabadi AB, Rostami MS, Lateran AF (2013) Presentation of a new and beneficial method through problem solving timing of open shop by random algorithm gravitational emulation local search. Int J Comput Sci Issues 10(1):745–752
15. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
16. Raju YHP, Devarakonda N (2018) Makespan efficient task scheduling in cloud computing. *Emerging Technol Data Mining Inf Secur*, pp. 283-298
17. Farahabadi AB, Hosseinabadi AR (2013) Present a new hybrid algorithm scheduling flexible manufacturing system consideration cost maintenance. Int J Sci Eng Res 4(9):1870–1875
18. Madni SHH, Abd Latiff MS, Ali J (2019) Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. Cluster Comput 22:301–334
19. Kashikolaei SMG, Hosseinabadi AR, Saemi B, Shareh MB, Sangaiah AK, Bian GB (2019) An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. *J Supercomput*, p. 1-28
20. Zhu X, Hussain M, Li X (2019) Energy-efficient independent task scheduling in cloud computing". *Human Center Comput*, pp. 428–439
21. Lee YC, Zomaya AY (2009) Minimizing Energy Consumption for Precedence-constrained Applications Using Dynamic Voltage Scaling, Cluster Comput Grid, pp. 92-99
22. Hosseinabadi AR, Vahidi J, Saemi B, Sangaiah AK, Elhoseny M (2018) Extended genetic algorithm for solving open-shop scheduling problem. Soft Comput, pp. 1–18
23. Hosseinabadi AR, Kardgar M, Shojafar M, Shamshirband S, Abraham A (2014) GELS-GA: Hybrid metaheuristic algorithm for solving multiple travelling salesman problem In: IEEE International Conference on Intelligent Systems Design and Applications (ISDA), pp. 76–81
24. Rostami AS, Mohanna F, Keshavarz H, Hosseinabadi AR (2015) Solving multiple traveling salesman problem using the gravitational emulation local search algorithm. Appl Math Inf Sci 9(2):699–709
25. Hosseinabadi AR, Vahidi J, Balas VE, Mirkamali SS (2018) OVRP_GELS: Solving open vehicle routing problem using the gravitational emulation local search algorithm. Neural Comput Appl 29(10):955–968
26. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179(13):2232–2248
27. Delaram J, Fatahi Valilai O (2018) A mathematical model for task scheduling in cloud manufacturing systems focusing on global logistics. Procedia Manuf 17:387–394
28. Karaboga D, Basturk B (2007) Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. In: International Fuzzy Systems Association World Congress, pp. 789-798, Springer
29. Carballal A, Pazos-Perez RI, Rodriguez-Fernandez N, Santos I, Garca-Vidaurrazaga MD, Rabunal J (2020) A point-based redesign algorithm for designing geometrically complex surfaces. A case study: Miralles's croissant paradox. IET Image Process 14(12):2948–2956
30. Pierezan J, Coelho LDS (2018) Coyote optimization algorithm: A new metaheuristic for global optimization problems. In: Proc. of IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, pp. 1–8
31. Pierezan J, Maidl G, Yamao EM, Coelho LDS, Mariani VC (2019) Cultural coyote optimization algorithm applied to a heavy duty gas turbine operation. Energy Convers Manage 199:1–18
32. Johnson S (2012) Emergence: the connected lives of ants, brains, cities, and software. Scribner, New York, NY, USA
33. Alworafi MA, Dhari A, El-Booz SA, Nasr AA, Arpitha A, Mallappa S (2018) An enhanced task scheduling in cloud computing based on hybrid approach". *Data Analyt Learn*, pp. 11-25
34. Kundra V (2011) "Federal cloud computing strategy"
35. Awad AI, El-Hefnawy NA, Abdel-Kader HM (2015) Enhanced particle swarm optimization for task scheduling in cloud computing environments. Procedia Comput Sci 65:920–929
36. Lakraa AV, Yadav DK (2015) Multi-objective tasks scheduling algorithm for cloud computing throughput optimization. Procedia Comput Sci 48:107–113
37. Wang T, Wei X, Liang T, Fan J (2018) Dynamic tasks scheduling based on weighted bi-graph in mobile cloud computing. Sustain Comput: Inf Syst 19:214–222
38. Pinedo M (2008) Scheduling: theory, algorithms, and systems. Springer, Berlin. https://www.springer.com/gp/book/9781489990433
39. Ajeena Beegom AS, Rajasree MS (2019) Integer-PSO: a discrete PSO algorithm for task scheduling in cloud computing systems''. Evolut Intell 12:227–239
40. Hosseinabadi AR, Rostami NSH, Kardgar M, Mirkamali SS, Abraham A (2017) A new efficient approach for solving the capacitated vehicle routing problem using the gravitational emulation local search algorithm. Appl Math Model 49:663–679
41. Hosseinabadi AR, Siar H, Shamshirband S et al. (2015) Using the gravitational emulation local search algorithm to solve the multi-objective flexible dynamic job shop scheduling problem in small and medium enterprises. Ann Oper Res 229:451–474. https://doi.org/10.1007/s10479-014-1770-8
42. Sangaiah AK, Suraki MY, Sadeghilalimi M, Bozorgi SM, Hosseinabadi AAR, Wang J (2019) A new meta-heuristic algorithm for solving the flexible dynamic job-shop problem with parallel

machines. Symmetry 11:165. https://doi.org/10.3390/sym11020165
43. Nategh MN, Hosseinabadi AR, Balas VE (2018) Ant_VRP: ant-colony-based meta-heuristic algorithm to solve the vehicle routing problem. Int J Adv Intel Paradig 11(3):315–334
44. Tirkolaee EB, Hosseinabadi AR, Soltani M, Sangaiah AK, Wang J (2018) A hybrid genetic algorithm for multi-trip green capacitated arc routing problem in the scope of urban services. Sustainability 10:1–21
45. Narendrababu Reddy G, Phani Kumar S (2018) Modified ant colony optimization algorithm for task scheduling in cloud computing systems. Smart Intel Comput Appl 104:357–365
46. Abd Elaziz M, Xiong S, Jayasena KPN, Li L (2019) Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. Knowl-Based Syst 169:39–52
47. Hosseinabadi AR, Slowik A, Sadeghilalimi M, Farokhzad M, Babazadeh M, Sangaiah AK (2019) "An ameliorative hybrid meta-heuristic algorithm for solving the capacitated vehicle routing problem. IEEE Access 7:175454–175465
48. Huang X, Li C, Chen H, An D (2020) Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. Cluster Comput 23:1137–1147
49. Peng H, Wena W, Tseng M, Li L (2019) Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. Appl Soft Comput 80:534–545
50. Sanaj MS, Joe Prathap PM, Jayasena KPN, Li L (2019) Nature inspired chaotic squirrel search algorithm (CSSA) for multi objective task scheduling in an IAAS cloud computing atmosphere. Engineering Science and Technology, an International Journal

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com