electronics

Article

An Efficient Combination of Genetic Algorithm and Particle Swarm Optimization for Scheduling Data-Intensive Tasks in Heterogeneous Cloud Computing

Kaili Shao [1], Hui Fu [1] and Bo Wang [2,*]

[1] Faculty of Engineering, Huanghe Science and Technology University, Zhengzhou 450063, China; sklemail@hhstu.edu.cn (K.S.); iefh@hhstu.edu.cn (H.F.)
[2] Software Engineering School, Zhengzhou University of Light Industry, Zhengzhou 450002, China
* Correspondence: wangb@zzuli.edu.cn

Abstract: Task scheduling is still an open issue for improving the performance of cloud services. Focusing on addressing the issue, we first formulate the task-scheduling problem of heterogeneous cloud computing into a binary non-linear programming. There are two optimization objectives including the number of accepted tasks and the overall resource utilizations. To solve the problem in polynomial time complexity, we provide a hybrid heuristic algorithm by combing both benefits of genetic algorithm (GA) and particle swarm optimization (PSO), named PGSAO. Specifically, PGSAO integrates the evolution strategy of GA into PSO to overcome the shortcoming of easily trapping into local optimization of PSO, and applies the self-cognition and social cognition of PSO to ensure the exploitation power. Extensive simulated experiments are conducted for evaluating the performance of PGSAO, and the results show that PGSAO has 23.0–33.2% more accepted tasks and 27.9–43.7% higher resource utilization than eight other meta-heuristic and hybrid heuristic algorithms, on average.

Keywords: task scheduling; particle swarm optimization; genetic algorithm; cloud computing

Citation: Shao, K.; Fu, H.; Wang, B. An Efficient Combination of Genetic Algorithm and Particle Swarm Optimization for Scheduling Data-Intensive Tasks in Heterogeneous Cloud Computing. *Electronics* **2023**, *12*, 3450. https://doi.org/10.3390/electronics12163450

Academic Editor: Javid Taheri

Received: 19 July 2023
Revised: 9 August 2023
Accepted: 14 August 2023
Published: 15 August 2023

1. Introduction

Nowadays, cloud computing has been widely applied in all walks of life due to its numerous advantages, e.g., pay-as-you-go, good scalability, high availability, etc., and its market is continuing to grow. As predicted by Statista Inc., the public cloud computing market is expected to reach an estimated USD 495 billion in 2022 [1], and the European cloud computing market is projected to reach EUR 560 billion by 2030 [2]. But the cloud still suffers from many issues like low resource efficiency and high energy consumption, which results in high operating costs and increased CO_2 emissions [3,4]. One of the most promising ways to address these issues is task scheduling, which makes decisions on the resource where each task is processed for cloud computing [5,6].

Unfortunately, the task scheduling problem has been proven as NP-hard [7] in various distributed and parallel computing systems, including clouds. That is to say, there is no polynomial algorithm to exactly solve the problem. Therefore, the existing works mainly focused on the heuristic and meta-heuristic algorithms to achieve near-optimal scheduling strategies with acceptable time consumptions.

In general, heuristic algorithms are designed specifically for a problem, by some local search strategies. Meta-heuristic algorithms are proposed for solving general-purpose optimization problems, exploiting some global search strategies designed based on natural and social phenomena. Usually, heuristics take only a little time for problem-solving but have limited performance, due to only using local searches [8]. Meta-heuristics can achieve better performance than heuristics, thanks to their global search abilities, but with some time overheads [9,10].

Each heuristic or meta-heuristic algorithm has its own strengths and weaknesses, and there is no algorithm that can obtain the best solution alone all the time. Thus, the integration of two or more algorithms provides opportunities for better solutions by combining strengths of these algorithms and compensating for each other's weaknesses. Therefore, we focus on designing a hybrid heuristic algorithm for task scheduling in clouds, to optimize user satisfaction and resource efficiency. User satisfaction has a great influence on the income and reputation of a cloud provider for service delivery [11]. The resource efficiency determines cost efficiency and energy efficiency at a heavy degree [3,12,13].

To be specific, we design a hybrid heuristic algorithm by integrating genetic algorithm (GA) into particle swarm optimization (PSO), for task scheduling in clouds. GA and PSO are representative meta-heuristic algorithms, and both have been widely applied for various decision-making problems in many fields [14,15]. GA simulates the natural phenomenon of Darwinian evolution, which implements the population evolution by crossover, mutation, and selection operators. GA has a powerful exploration ability to search unexplored solution regions for a possibility of finding a better or the global optimal solution. But in general, GA has some disadvantages, such as a slow convergence speed. These disadvantages of GA can be remedied by PSO. The design inspiration of PSO is the social behaviour of bird flocking or fish schooling. PSO has advantages such as fast convergence rate, but is easily trapped into local optima.

In addition, many existing works focused on scheduling computer-intensive tasks, which ignored the data transmission delay [16]. These works are not appropriate for big data tasks. Thus, in this paper, we integrate the population evolution strategy of GA into PSO, and design a hybrid heuristic algorithm, PGSAO (PGSAO is the letter sequence constructed by alternating PSO and GA, which hints their integration), for scheduling data-intensive tasks in cloud computing. Specifically, PGSAO exploits GA's evolution operators to guarantee the population diversity and ensure a powerful exploration ability. Meantime, PGSAO uses the social behaviour of PSO to enhance its exploitation power. The contributions of this paper can be summarized as follows.

- We formulate the task scheduling problem with deadline constraints into a combinatorial optimization problem for heterogeneous cloud computing. There are two optimization objectives: the number of tasks that are finished within their respective deadlines (the major one) and resource utilization (the minor one), which are quantifying user satisfaction and resource efficiency, respectively.
- We propose a hybrid heuristic algorithm (PGSAO) for the task scheduling problem, by integrating GA into PSO. In PGSAO, each individual corresponds to a task scheduling solution, with an integer-encoding strategy. During the population evolution phase, PGSAO performs crossover and mutation operators on each individual to produce offspring, where the individual is crossed with not only another individual (like GA) but also its personal best and the global best codes (the self-cognition and social cognition of PSO). After each operator, every individual is replaced by its best offspring.
- We conduct extensive simulated experiments to evaluate the performance of PGSAO. Experiment results show that PGSAO can achieve 23.0–33.2% more finished tasks and 27.9–43.7% higher resource utilization, on average, compared with eight other methods, which verifies the performance superiority of PGSAO.

In the following of this paper, Section 2 analyses recent related works. Section 3 presents the formulation for task scheduling problem. Section 4 illustrates the proposed hybrid heuristic method, PGSAO. Section 5 evaluates the performance of PGSAO. Section 6 discusses interpretations and implications of our study. Section 7 concludes our work.

2. Related Works

With the development of more than ten years in both industry and academia, cloud computing has been applied in almost all fields. To improve the efficiency and effectiveness of cloud computing, numerous works address task scheduling problems based on heuristic and meta-heuristic algorithms.

Nabi et al. [17,18] employed PSO to improve makespan, resource utilization, and throughput for clouds, with an adaptive inertia weight strategy that updates the inertia weight based on improvement times of personal best values. Pirozmand et al. [19] and Pradhan et al. [20] exploited GA to optimize the energy consumption and the makespan for scheduling tasks in clouds. Malti et al. [21] proposed an efficient task-scheduling algorithm based on a flower pollination algorithm to improve makespan, resource cost, and execution reliability. Two works proposed by Mangalampalli et al. [22,23] used cat swarm optimization (CWO) and whale optimization algorithm (WOA) to schedule tasks on virtual servers for cloud computing, respectively, to improve makespan and energy consumption. Aghdashi and Mirtaheri [24] presented a task-scheduling method based on the hill-climbing algorithm to optimize response time in cloud environments. These above works used only one meta-heuristic algorithm, meaning they were not concerned with the performance improvement of task scheduling by exploiting the complementarity of different meta-heuristics.

The work proposed by Belgacem and Beghdad-Bey [25] focused on improving makespan and resource cost in a public cloud providing resources in the form of virtual machines (VM). Their work used Heterogeneous Earliest Finish Time (HEFT) to order the execution of tasks, and Ant COlony algorithm (ACO) to find an efficient assignment of tasks to VMs. Aktan and Bulut [26] presented two hybrid heuristic algorithms, GASA and DESA, to improve makespan and resource utilization for clouds. GASA and DESA apply Simulated Annealing (SA) on each individual in the last evolution of GA and Differential Evolution (DE), respectively. CWOA, proposed by Pradeep et al. [27], combined Cuckoo Search Algorithm (CSA) and WOA to increase the performance in memory utilization, makespan and energy efficiency. CWOA used Lévy flight to update the locations of humpback whales again at after every position update of WOA. GSAGA [28] combined GA with Gravitational Search Algorithm (GSA) to improve the resource cost for processing tasks in a cloud. QSSGWA [29] used quantum operator to improve the population initialization, and combined Salp Swarm Algorithm (SSA) with Grey Wolf Optimizer (GWO), which applied position update strategies of SSA and GWO for the first and second half of the population, respectively. To optimize makespan, Cheikh et al. [30] proposed a hybrid heuristic scheduling algorithm by combining PSO and Extremal Optimization (EO), which used PSO to provide the initial solution for EO. PSOM [31] performed the mutation operator of GA on each particle to improve PSO. Nwogbaga et al. [32] applied PSO after GA for the population evolution. Wang et al. [33] applied the evolution strategy of GA first, and then that of PSO in each evolution. These methods tried to combine benefits of two or more algorithms, but they exploited these algorithms sequentially and separately in each iteration or in the whole evolutionary process, which provides inefficient combinations.

Thus, in this paper, we present an efficient integration strategy for combining the benefits of both GA and PSO, and propose the PGSAO hybrid heuristic scheduling algorithm for optimizing the task-acceptance rate and the resource utilization in heterogeneous cloud computing.

3. Problem Formulation

In this paper, we consider the cloud computing scenario as providing services by several multi-core servers. For each task requested by users, the cloud provider uses one core of these servers for its processing. To achieve high resource efficiency and ensure high user satisfaction, the cloud provider should carefully decide which core is assigned for each task, which is the goal of task-scheduling methods. In the following, we formulate the optimization problem for task scheduling.

In the considered cloud computing, there are S servers represented as $s_j, j = 1, ..., S$. Server s_j has n_j cores each with g_j computing capacity for processing tasks. Assuming T tasks ($t_i, i = 1, ..., T$) are requested to be processed by the cloud, task t_i requires c_i computing resources (i.e., its computing size) for its completion. The input data needed by task t_i are a_i in amount. We consider that the cloud uses a distributed file system (DFS) for its data

management, which is widely used in various distributed systems. The DFS provides r_j read bandwidth for server s_j. Then, when t_i is assigned to a core in s_i, it needs c_i/g_j time for its computing, and requires a_i/r_j time for transmitting its input data. In this paper, we ignore the time consumed by output data transmission for a task, as is done by many works, because the amount of the output data is very small in many scenarios. The deadline of t_i is d_i, which means t_i requires that it must be finished before d_i if the cloud accepts its request. In this work, we concern hard deadline constraints. That is to say, when judging that a task cannot be finished within its deadline, the cloud has no profit for accepting and processing it, and the cloud will reject its request.

For the problem formulation of task scheduling, we define binary variables, $x_{i,j,k}$, $i = 1, ..., T, j = 1, ..., S, k = 1, ..., n_j$, to represent the decision-making of tasks assignments, as Equation (1). $x_{i,j,k} = 1$ if t_i is assigned to kth core of s_j for its processing, and $x_{i,j,k} = 0$, otherwise.

$$x_{i,j,k} = \begin{cases} 1, & \text{if } t_i \text{ is assigned to the } k\text{th core in } s_j \\ 0, & \text{otherwise} \end{cases}$$
$$i = 1, ..., T, j = 1, ..., S, k = 1, ..., n_j. \quad (1)$$

In this work, we do not consider the redundant execution, which improves the performance of task executions sometimes but consumes more resources. This can result in low resource efficiency. That is to say, there is at most one core that a task is assigned to, i.e.,

$$\sum_{j=1}^{S} \sum_{k=1}^{n_j} x_{i,j,k} \leq 1, i = 1, ..., T. \quad (2)$$

And Equation (3) gives the number of tasks assigned to cores for their processing, i.e., the number of tasks accepted by the cloud, N.

$$N = \sum_{i=1}^{T} \sum_{j=1}^{S} \sum_{k=1}^{n_j} x_{i,j,k}. \quad (3)$$

For each core, it cannot process two or more tasks simultaneously, i.e., the computing periods of any two tasks assigned to a core are non-overlapping. Thus, Equation (4) holds, where b_i and e_i represent the begin time and the end time of processing t_i. This is because, for two tasks (t_{i1} and t_{i2}) processed by one core, when t_{i1} is processed before t_{t2}, $b_{i1} < e_{i1} \leq b_{i2} < e_{i2}$. And if t_{i1} is processed after t_{t2}, $b_{i2} < e_{i2} \leq b_{i1} < e_{i1}$. For Equation (4), the equality holds if t_{i1} and t_{i2} are not assigned to one core.

$$x_{i1,j,k} x_{i2,j,k} (b_{i1} - e_{i2})(b_{i2} - e_{i1}) \leq 0,$$
$$i1 \neq i2, i1, i2 = 1, ..., T. \quad (4)$$

When a task, say t_i, is assigned to a core of server s_j, it requires a_i/r_j time for its data transfer, and c_i/g_j time for its computing. Thus, the begin time and the end time of task processing are satisfying Equation (5).

$$\sum_{j=1}^{S} \sum_{k=1}^{n_j} (x_{i,j,k} (b_i + \frac{a_i}{r_j} + \frac{c_i}{g_j})) \leq e_i, i = 1, ..., T. \quad (5)$$

And the deadline requirements of tasks can be formulated as Equation (6).

$$e_i \leq d_i, i = 1, ..., T. \quad (6)$$

For each server, its occupied time for task processing is the latest end time of tasks assigned to it, which is formulated as follow, where τ_i is the occupied time of server.

$$\tau_j = \max_i \max_k \{x_{i,j,k} e_i\}, j = 1, ..., S. \tag{7}$$

Then, the amount of occupied resources on server s_j is $\tau_j n_j g_j$. But for a server (s_j), the amount of resources actually used for task processing is the total computing size of tasks assigned to the server, which is

$$\sum_{i=1}^{T} \sum_{k=1}^{n_j} (x_{i,j,k} c_i).$$

Thus, the resource utilization of each server can be calculated by (Equation (8)). And the overall resource utilization of the cloud can be achieved by Equation (9).

$$u_j = \frac{\sum_{i=1}^{T} \sum_{k=1}^{n_j} (x_{i,j,k} c_i)}{\tau_j n_j g_j}, j = 1, ..., S. \tag{8}$$

$$U = \frac{\sum_{j=1}^{S} \sum_{i=1}^{T} \sum_{k=1}^{n_j} (x_{i,j,k} c_i)}{\sum_{j=1}^{S} (\tau_j n_j g_j)}. \tag{9}$$

Based on above formulations, we can model the task scheduling problem in the cloud as followings.

$$\text{Maximizing } \ N + U, \tag{10}$$

subject to Equations (1)–(9). The optimization objective is the number of accepted tasks (N) plus overall resource utilization (U). Noticing that $U \leq 1$, the maximization of accepted task number is the major objective, and the optimization of resource utilization is the minor one. The number of request tasks is fixed. Thus, the maximization of accepted task number is identical to the optimization of accept ratio that is the proportion of accepted task number to the total requested task number, which is one of the most commonly used measurements on user satisfaction. Resource utilization is one of the most frequently used indicators of resource efficiency.

The decision variables of the optimization problem are $x_{i,j,k}, i = 1, ..., T, j = 1, ..., S$, $k = 1, ..., n_j$. All of them are either 0 or 1. In addition, Equation (4) and (7) are nonlinear. Thus, the optimization problem is a kind of binary nonlinear programming problems (BNLP). There are existing tools, e.g., the optimization toolbox of MathWorks [34], that can solve the problem. But these tools have time overheads that increase with problem size exponentially, and thus are not applicable for solving problems with middle-to-large scales. Therefore, in the next section, we propose a hybrid heuristic algorithm with polynomial time for solving the problem.

4. Hybrid GA and PSO Scheduling Algorithm

In this section, we illustrate the hybrid heuristic method, PGSAO, for task scheduling in clouds, combining the benefits of both GA and PSO, which is outlined in Algorithm 1. As shown in the algorithm, we first design an encoding method to express the task-scheduling solutions as numerical values/vectors (codes) to construct the solution space. This is the first thing that must be carried out for all meta-heuristics. Meanwhile, the fitness function is designed for evaluating the goodness of codes or corresponding task scheduling solutions. With the encoding method and the fitness function, PGSAO uses following steps to search the solution space for an efficient task-scheduling strategy. We use the terminologies of PSO to describe search process of PGSAO. The searching actions correspond to the position (code) changes of particles.

Algorithm 1 PGSAO: the hybrid GA and PSO scheduling

Input: The encoding method, the fitness function, the information of tasks and servers, the parameters of GA;
Output: a task scheduling solution;
1: Initializing a population by randomly setting the code of each particle;
2: Calculating the fitness of each particle;
3: Setting the personal best code (pb) as the initialized one for each particle;
4: Setting the global best code (gb) as the code with the best fitness in the population;
5: **while** The terminal condition is not met **do**
6: **for** Each particle **do**
7: Crossing the particle's code with another's, its pb, and its gb, with a certain probability, respectively, and producing six new codes (each crossover operator produces two new codes);
8: Evaluating fitnesses of six new produced codes, and getting the best code (β) that has the best fitness from these six chromosomes;
9: Setting the particle's core as β;
10: If the fitness of β is better than that of pb, then updating pb as β;
11: If the fitness of β is better than that of gb, then updating gb as β;
12: Mutating the particle's code with a certain probability, which produces one new code;
13: Setting the particle's core as the new one;
14: Evaluating the fitness of the new code, and updating pb and gb if the new chromosome has a better fitness, respectively.
15: **end for**
16: **end while**
17: Decoding gb into a task scheduling solution;
18: **return** the task scheduling solution;

(1) PGSAO randomly initializes several starting points (codes of particles) for following searches (line 1), and evaluates the fitness of each code (line 2). Meanwhile, PGSAO records the personal best code as its initialized one for each particle (line 3), and the global best code as the best one in all codes of particles (line 4).
(2) PGSAO explores and exploits the solution space by updating codes of particles with the crossover and mutation operators of GA as well as the self-cognition and social cognition ideas of PSO (lines 5–16). In detail, PGSAO repeats the following steps for each particle (line 6), until the predefined terminal condition is reached (line 5).
    (a) With set crossover probability, PGSAO crosses the particle's code with the code of another particle, its personal best code and the global best code, respectively, by uniform crossover operator. The latter two crossovers are exploiting the self-cognition and social cognition in the population, respectively.
    (b) For each of the six new codes produced by crossover operators in the previous step, PGSAO evaluates its fitness, and compares its fitness with that of the personal best code and the global best code, respectively. When the new code has a better fitness, the personal/global best code is updated as the new code.
    (c) PGSAO updates the particle' code as the best one of the above six new codes. This step combines the position-updating strategy of PSO and the tournament selection operator of GA.
    (d) PGSAO performs a uniform mutation operator on the particle's code, with set mutation probability. This produces a new code.
    (e) PGSAO does the same as in steps 2b and 2c for the new code produced by the mutation operator in the previous step.
(3) PGSAO returns the best solution decoded by the global best code (lines 17 and 18).

Meta-heuristic algorithms like PGSAO can have two types of terminal conditions: setting a maximum number of iterations or defining a consecutive number of iterations where the best solution remains unchanged. In this paper, we use the first option.

In PGSAO, the fitness function is identical to the objective function, Equation (10), formulated in the above section. Both the number of accepted tasks and resource utilization can be achieved based on the task-scheduling solution decoded by the corresponding code.

Next, we illustrate the encoding method, the crossover and mutation operators in detail. And at the end of this section, we analyze the time complexity of PGSAO.

4.1. Encoding Method

In PGSAO, each task-scheduling solution is encoded into a vector, where tasks and dimensions of the vector have a one-to-one correspondence. The value in a dimension represents the computing core where corresponding task is assigned. Thus, the value range of each dimension is from 1 to the number of cores. Then, given a vector with the dimension of the task number, i.e., a code of particles, we can obtain an assignment of tasks to cores. For tasks assigned to a core, PGSAO exploits earliest deadline first (EDF) to decide their processing orders. Then, we can obtain a whole task-scheduling solution from a code of particles.

For example, there are six tasks ($t_1, ..., t_6$) and two servers (s_1 and s_2) each with two cores, in a cloud system. The cores of the first server are represented as p_{11} and p_{12}, and that of the second server are p_{21} and p_{22}. Then, the code [1, 1, 2, 2, 3, 4] corresponds to the assignment strategy that t_1 and t_2 are assigned to p_{11}, t_3 and t_4 to p_{12}, t_5 to p_{12}, and t_6 to p_{22}, as shown in Figure 1. And in p_{11} and p_{12}, the task with earlier deadline is processed first.

code	1	1	2	2	3	4

↓ decoding ↑ encoding

tasks	t_1	t_2	t_3	t_4	t_5	t_6
cores	p_{11}	p_{11}	p_{12}	p_{12}	p_{21}	p_{22}

Figure 1. An example illustrating the encoding method.

4.2. Crossover Operator

In this section, we use uniform crossover operator as it helps to ensure the population diversity for enhancing exploration power of PGSAO [35]. In every iteration of population evolution, PGSAO performs the crossover operator on each particle three times, with another particle randomly selected, its personal best code, and the global best code, respectively. For the uniform crossover operator, two new codes (offspring) are produced by two selected codes, by the following process. For each dimension, a value between 0 and 1 is generated randomly, and is judged on whether it is smaller than a predefined probability (set as the crossover probability in this work). If the value is smaller, two codes swap their values in the dimension, which represents that two tasks swap their assigned cores in the task scheduling.

For example, as shown in Figure 2, the uniform crossover operator is performed on two codes, [3, 9, 1, 3, 6, 3] and [5, 7, 9, 1, 6, 4]. By the above process, it is decided to swap the values of these two codes in the second, third, and sixth dimensions, based on generated random values. Then, two new codes are produced, which are [3, 7, 9, 3, 6, 4] and [5, 9, 1, 1, 6, 3], respectively. This crossover operator searches out two new solutions by swapping the cores for processing second, third, and sixth tasks, respectively, on their parents' solutions.

one code	3	9	1	3	6	3
another code	5	7	9	1	6	4

⇓ crossing

new code	3	7	9	3	6	4
new code	5	9	1	1	6	3

Figure 2. An example illustrating the uniform crossover operator.

4.3. Mutation Operator

For the same reason as using the uniform crossover operator, PGSAO uses the uniform mutation operator. Each particle code itself performs the mutation operator, with the following procedure. PGSAO produces a random value within [0, 1] for each dimension. And if the produced value is smaller than pre-set probability (set as the mutation probability in this paper), the value is changed to another value in corresponding dimension, which means that a task is re-assigned from one core to another core.

Figure 3 gives an example to illustrate the uniform crossover operator used by PGSAO. Given the code, [3, 9, 1, 3, 6, 3], to be performing the crossover operator, values are mutated from 1 and 3 to 9 and 6, respectively, in the third and sixth dimensions. And the new code produced by the crossover operator is [3, 9, 9, 3, 6, 6]. The mutation represents that the cores where the third and sixth tasks are assigned are changed from the first and ninth ones to third and sixth ones, respectively.

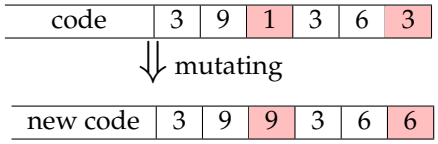


Figure 3. An example illustrating the uniform mutation operator.

4.4. Complexity Analysis

The main time overhead of PGSAO is the iterative process of population updating (lines 5–16 in Algorithm 1). In each iteration, PGSAO performs crossover operator three times and mutation operator one time. Both crossover and mutation operators have $O(T)$ time complexity, which can be easily obtained from the illustration in Sections 4.2 and 4.3. The fitness evaluation has $O(T^2)$ time complexity at worst, as we exploit EDF to decide the processing order of tasks for each core. Thus, the time complexity of each iteration is $O(T + T^2) = O(T^2)$. Thus, the time complexity of PGSAO is $O(IPT^2)$, where I is the maximum iterative number, and P is the number of particles. This time complexity is identical to that of many meta-heuristic algorithms including GA and PSO.

5. Performance Evaluation

In this section, we conduct simulated experiments to evaluate the performance of PGSAO, where simulation parameters are set referring to recent related works. In the following, we first illustrate the experiment environment, and then analyze the experiment results.

5.1. Experiment Environment

Referring to [17,36,37], we establish the following cloud computing environment. In the cloud, there are 50 servers and 1000 tasks. The number of each server is set in a range from 2 to 32, randomly. The computing capacity of each core is set between 1000 and 4000 Million Instructions Per Second (MIPS). The read bandwidth for each server is in the range between 100 and 1000 MB/s. Every task has a 1000 to 100,000 Million Instructions (MI) computing size, and 100 to 10,000 MB input data for processing. The deadline of a task is set in a range between 1 and 100 s. To evaluate the energy consumption of server s_j, we apply the widely used linear model,

$$E_j = \int_{\tau} (P_j^{idle} + (P_j^{full} - P_j^{idle}) \cdot u_j(\tau)) d\tau.$$

where E is consumed energy. $u(\tau)$ is resource utilization changed with time τ. P^{idle} and P^{full} are power consumption when the server is idle and fully loaded, respectively. The values of P^{idle} and P^{full} for each server is set referring to [38], as shown in Table 1.

Table 1. The power parameters of servers.

#Core	P^{idle}	P^{full}
$[1, 8)$	110	175
$[8, 16)$	125	210
$[16, 24)$	210	300
$[24, 32]$	350	500

We compare our method with following methods.

Hill Climbing (HC) is the method used in some works for task scheduling [24,39], which exploits a single individual search strategy. The basic idea of HC is to repeat replacing the search point with its better neighbor, until no neighbor is better than the current search point.

Genetic Algorithm (GA) is one of the most commonly used meta-heuristic algorithms for task scheduling, e.g., [19,20]. GA uses crossover, mutation, and selection operator to evolve its population.

GA with HC (GAHC) is proposed in [37]. It performs an iteration of HC on each individual before the evolution of GA. Meanwhile, GAHC replaces the selection operator with a replacement, such as the particle updating idea used in PSO.

GA with replacement (GAR) uses the replacement instead of the selection operator in GA, which is same as GAHC except that GAR does use the HC operator.

Particle swarm optimization (PSO) is also one of the most popular algorithm for the task-scheduling problem, e.g., [40]. PSO exploits the movement behaviour of a bird flock, fish school, or insect swarm, to evolve its population.

PSO with Mutation (PSOM) improves PSO by the mutation operator [31]. It performs the mutation operator on each particle after each evolution of PSO.

GA+PSO first exploits GA at the first half phase of population evolution, and then evolves the population from GA by PSO at the second [32].

GAPSO first uses GA to evolve its population, and then applies PSO on the evolution, in each iteration [33].

The metrics used for our performance comparison include following three aspects.

User Satisfaction has great influence on the income and reputation of clouds [41]. The following three metrics are used for its quantification. The **number** (N), **computing size** ($\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)$), and cumulated input **data amount** ($\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot a_i)$) of accepted tasks.

Resource Efficiency determines the operator cost of clouds, to a large extent. It is quantified by following three metrics, the overall **resource utilization** (U), **energy efficiency in computing**, and **energy efficiency in data processing**, which are, respectively, calculated by

$$\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)}{\sum_{j=1}^{S}E_j}$$

and

$$\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot a_i)}{\sum_{j=1}^{S}E_j}.$$

The latter two are the finished computing size and the processed data amount, respectively, by consuming one energy unit.

Processing Efficiency is the task processing rate on the cloud. Two metrics are applied for its evaluation, **the finished computing size and the processed data amount per time unit**, which can be calculated, respectively, by

$$\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)}{\max_{j=1}^{S}\tau_j}$$

and

$$\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot a_i)}{\max_{j=1}^{S}\tau_j}.$$

The experiment process involves first generating a cloud computing environment with the aforementioned parameters of servers and tasks. Then, the metrics of PGSAO and compared methods are measured on the generated cloud computing environment. To highlight the performance superior of PGSAO, we normalize each metric value of each method by dividing it into that of PGSAO, and report the normalized (relative) performance in the following. We repeat the above process more than 100 times, and present the boxplot graph for each metric, in the following.

5.2. Experiment Results

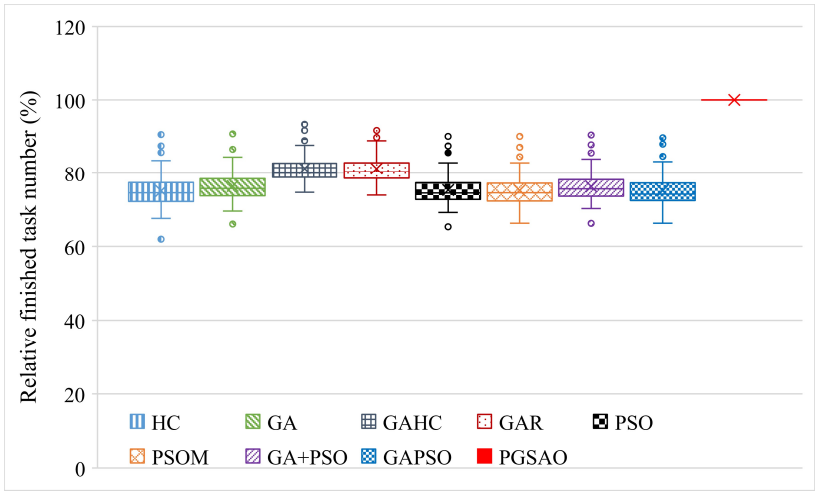
5.2.1. User Satisfaction

Figure 4 shows the relative performance achieved by various task-scheduling methods, in three metrics of user satisfaction. From these figures, we can see that PGSAO has 23.0–33.2% more accepted tasks, a 28.4–43.4% larger finished computing size, and a 22.8–33.9% greater amount of processed input data than other methods, on average. In addition, PGSAO always has better performance in these satisfaction metrics, compared with other methods. These phenomena verify the performance superiority of our hybrid heuristic method in optimizing user satisfaction.

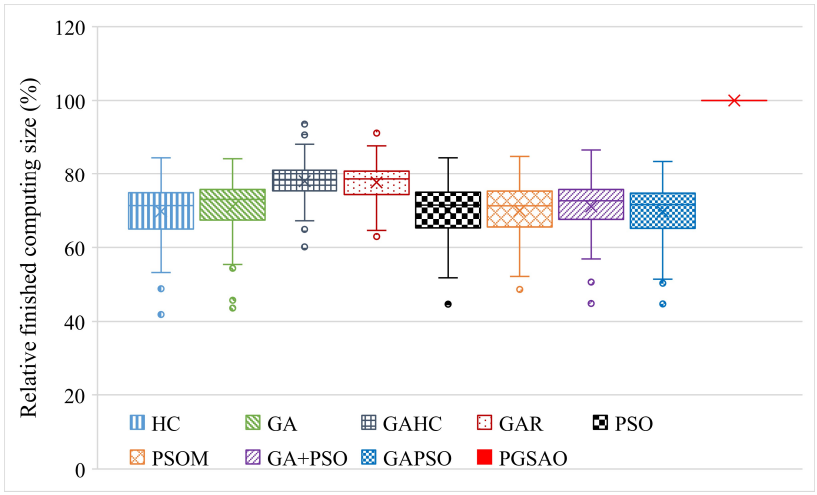
The reasons for why PGSAO has good performance mainly include following three aspects: the efficient integration strategy, the use of self- and social-cognition of PSO by the crossover operator of GA, and the replacement instead of the selection operator for particle updating. These are verified by our experiment results as illustrated in the following paragraphs, respectively.

As shown in Figure 4, PGSAO can accept 31.0% and 32.8% more tasks than GA and PSO, respectively, on average. And similar experiment results can be found in the other two satisfaction metrics. In addition, PGSAO has about a 30% larger number of accepted tasks on average, compared with PSOM, GA+PSO, and GAPSO, even though all of them are combining GA and PSO. This is mainly because PSOM, GA+PSO, and GAPSO are using the GA and PSO separately in the whole evolution process or each iteration, instead of organically integrating one algorithm into another, as is done by our method. These above results verify the high efficiency of our integration strategy.

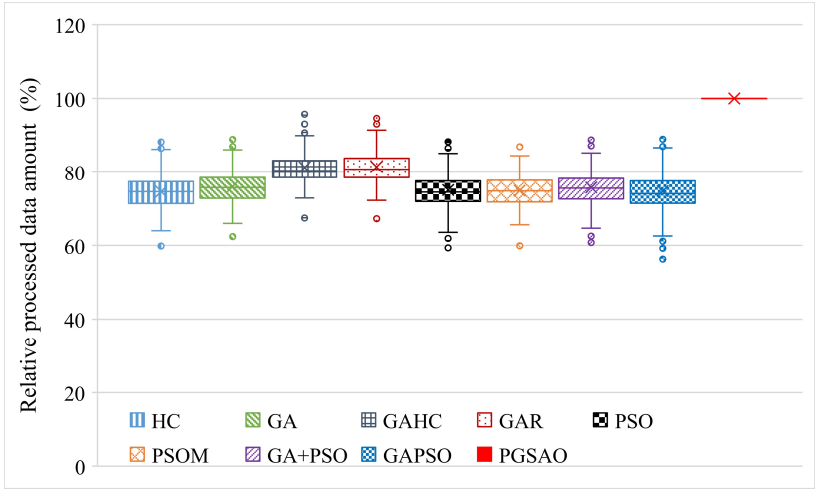
Figure 5 gives the performance of PGSAO without using self-cognition or social cognition, in optimizing the accepted tasks. PGSAO without self-/social-cognition is removing a crossover operator performed on the global/personal best code, from PGSAO. From Figure 5, we can see that PGSAO without self-/social-cognition has better performance than PSO and GA in user satisfaction. In addition, PGSAO has a greater number of accepted tasks than PGSAO without self-/social-cognition. This verifies the efficiency of integrating both self- and social-cognition into GA by our method. PGSAO without self-cognition has better user satisfaction than PGSAO without social-cognition. This is mainly because the global best code has better genes than the personal best code generally, and thus the crossover operator performed with the global best code can produce better offspring. Therefore, one of our future works is to design adaptive evolution strategies for increasing the likelihood of making better genes be transmitted to offspring for evolutionary algorithm-based methods.



(**a**) Finished task number



(**b**) Finished computing size



(**c**) Processed data amount

Figure 4. The relative user satisfactions achieved by various task-scheduling methods.

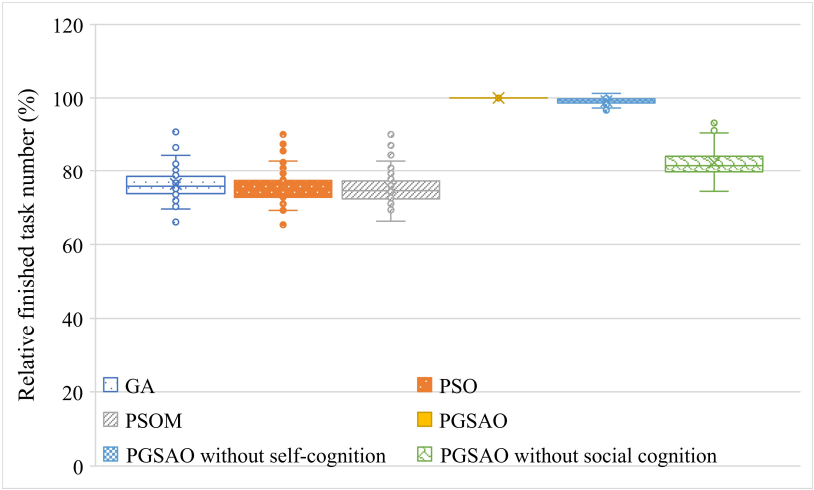


Figure 5. The improvement of PGSAO by the integration of GA and PSO in optimizing accepted tasks.

Figure 4 also shows that GAR and GAHC have comparable performance, and both of them have about 6.5% more accepted tasks than GA on average. Thus, the better performance of GAHC mainly comes from the replacement instead of integrating HC in GA. This may be because HC can improve the qualities of individuals, but it can decrease the population diversity for GA by moving each individual into a local optimum. Thus, PGSAO uses the replacement of PSO instead of the selection operator of GA.

From Figure 4, we also can see that PSOM, GA+PSO, and GAPSO have comparable performance with GA and PSO, but GAHC has better performance than GA and HC. This illustrates that the integration of two or more algorithms can improve the performance for each algorithm, but the integration strategy should be carefully designed to avoid an ineffectual integration.

5.2.2. Resource Efficiency

Figures 6 and 7 give the relative resource utilization and the relative energy efficiency when applying various methods, respectively. As shown in these figures, PGSAO achieves 27.9–43.7% greater resource utilization, 21.4–34.7% more computing size finished by a unit of energy, and 15.7–25.9% more data amount processed by per energy unit, compared to other methods, on average. Additionally, PGSAO has better resource utilization and energy efficiency than other methods all the time, except for very few outliers. These phenomena prove that our method has a very good performance in resource efficiency. The reason for these results is that PGSAO completes more tasks with comparable resources, compared with other scheduling methods. PGSAO has only −0.639–0.594% more occupied resources, but finishes a 28.4–43.4% greater computing size and processes a 22.8–33.9% greater data amount, on average, compared with other methods. This reflects the high resource efficiency of our method.

5.2.3. Processing Efficiency

The relative processing rates achieved by all task-scheduling methods are shown in Figure 8, which provides a similar result to the relative resource efficiency. PGSAO achieves a 28.6–45.7% higher computing rate and 23.2–36.2% faster data processing rate than others, on average, and always has fastest processing rate except for a few outliers. As is shown in our experiments, PGSAO has a 0.126–2.77% earlier makespan for completing accepted tasks on average, but finishes a greater computing size and data amount, compared with other methods. Thus, our proposed algorithm can provide better processing efficiency than other algorithms.

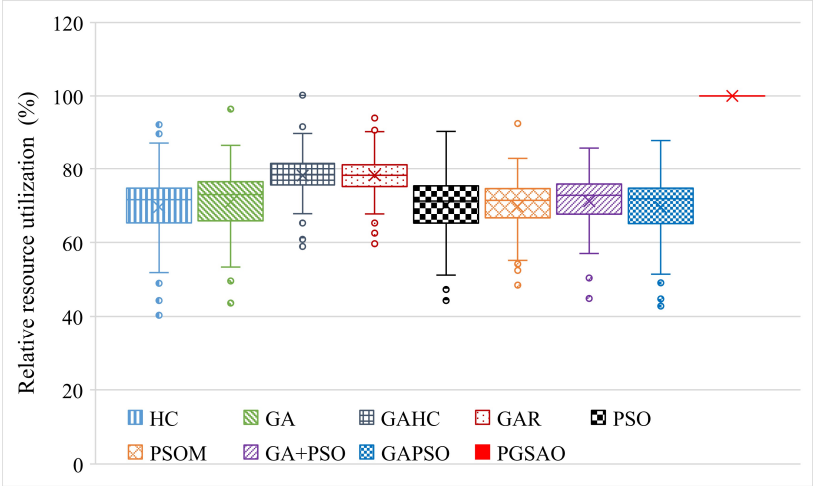
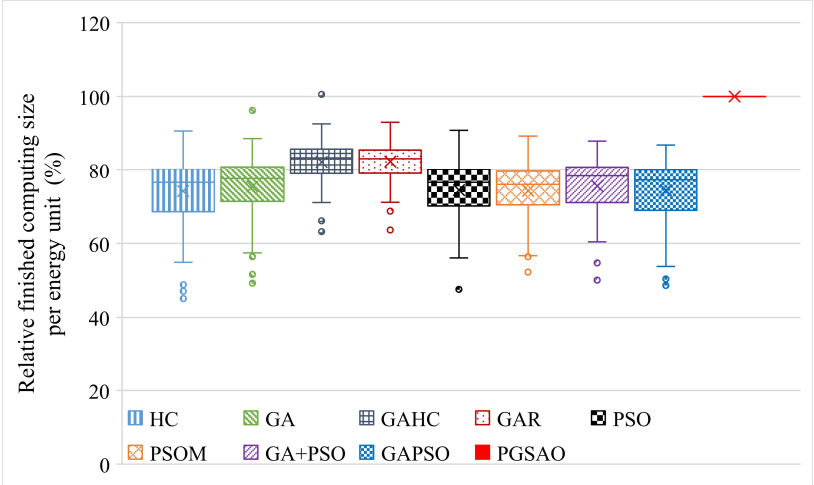
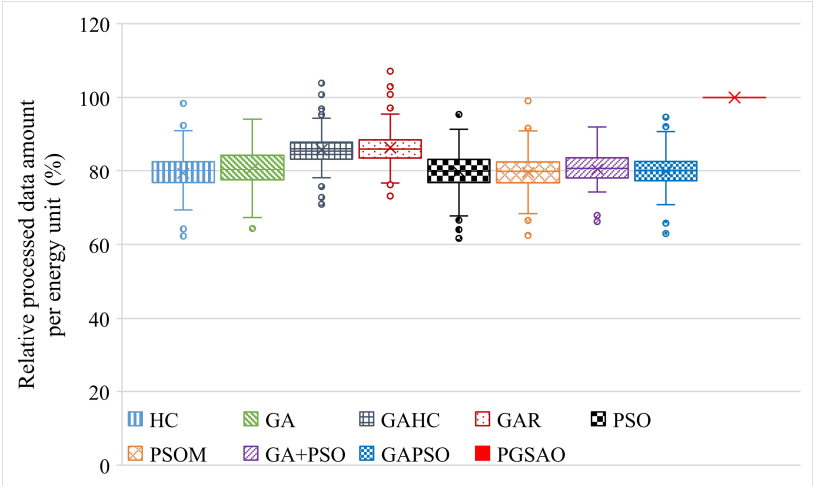


Figure 6. The relative resource utilizations achieved by various task-scheduling methods.

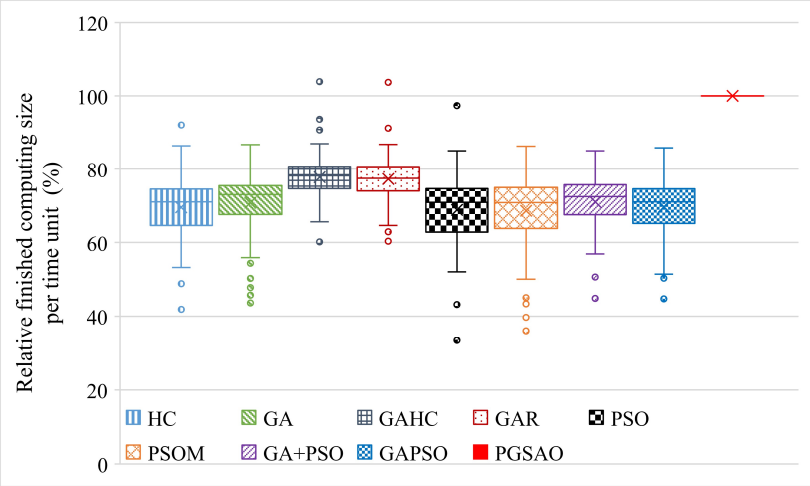


(**a**) Finished computing size per unit of energy

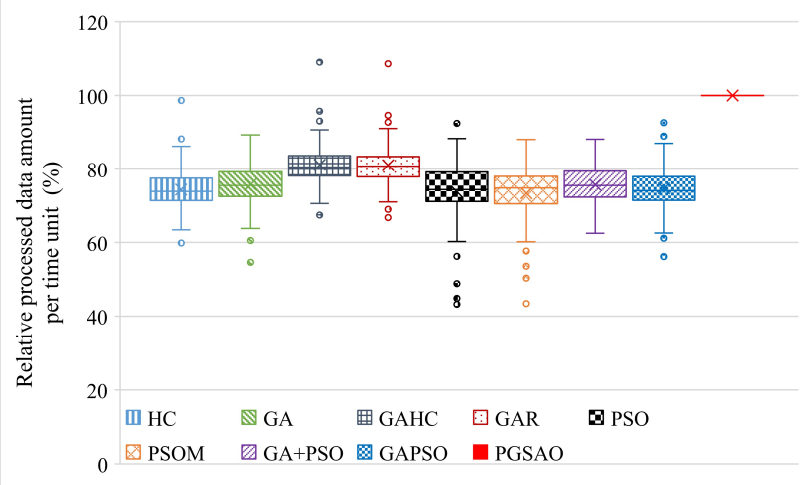


(**b**) Processed data amount per unit of energy

Figure 7. The relative energy efficiencies achieved by various task-scheduling methods.



(**a**) Finished computing size per time unit



(**b**) Processed data amount per time unit

Figure 8. The relative processing efficiencies achieved by various task-scheduling methods.

6. Discussion

In this work, we exploit an efficient combination approach to integrate two representative meta-heuristic algorithms, GA and PSO. Both the strengths and weaknesses of GA and PSO complement each other, where GA has a powerful global search ability benefiting from the crossover and mutation operators but slow convergence; On the contrary, PSO has fast convergence speed but can easily become trapped into local optima, due to its self and social cognitions of swarm. Therefore, by combining GA's operators and PSO's cognitions, PGSAO achieves a much better performance than GA and PSO as well as other hybrid heuristic scheduling algorithm. Thus, when trying to combine different algorithms, one should first analyze their benefits and corresponding elements, and then design an approach to combine these elements.

7. Conclusions

In this paper, we study the task-scheduling problem for clouds to optimize the user satisfaction and the resource efficiency. We first formulate the problem as a BNLP model, and then propose a polynomial time algorithm, PGSAO, to address the problem, by combining the advantages of both GA and PSO. Specifically, PGSAO uses the framework of PSO and the evolution strategy of GA. Meanwhile, PGSAO integrates the self- and social-cognition of PSO into GA, which are implemented by the crossover operators of each particle with its

personal and the global best codes, respectively. Simulated experiments verify the excellent performance of PGSAO in user satisfaction, resource efficiency and processing efficiency.

In this paper, we focused on independent tasks, such as bag-of-tasks, which is one of the most common application in various distributed systems. In the future, we will extend our method to support the scheduling of workflow applications with task dependencies. Additionally, we will explore a more efficient integration strategy to take full advantage of the complementarity of diversified heuristic and meta-heuristic algorithms. As with other meta-heuristic-based algorithms, PGSAO trades time overhead for scheduling performance. Therefore, meta-heuristic-based algorithms are suitable for batch tasks but not for real-time tasks. Thus, in the future, we will consider designing an adaptive approach to ensemble different kinds of algorithms for scheduling both real-time tasks and batch tasks.

Author Contributions: Conceptualization, K.S. and B.W.; methodology, K.S. and H.F.; software, H.F.; validation, K.S., H.F. and B.W.; formal analysis, H.F.; investigation, K.S.; resources, K.S., H.F. and B.W.; data curation, B.W.; writing—original draft preparation, K.S.; writing—review and editing, H.F. and B.W.; visualization, H.F.; supervision, K.S. and B.W.; project administration, K.S. and B.W.; funding acquisition, K.S. and B.W. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the key scientific and technological projects of Henan Province (Grant No. 232102211084, 232102210023, 232102210125), the National Natural Science Foundation of China (Grant No. 61975187, 62072414), Henan key scientific research project of higher universities (22A520033), Zhengzhou Basic Research and Applied Research Project (ZZSZX202107) and China Logistics Society (2022CSLKT3-334).

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

ACO	Ant colony algorithm
BNLP	Binary nonlinear programming problems
CSA	Cuckoo search algorithm
CWO	Cat swarm optimization
DE	Differential evolution
DFS	Distributed file system
EDF	Earliest deadline first
EO	Extremal optimization
GA	Genetic algorithm
GAHC	GA with HC
GAR	GA with replacement
GSA	Gravitational Search Algorithm
GWO	Grey wolf optimizer
HC	Hill climbing
HEFT	Heterogeneous earliest finish time
MI	Million instructions
MIPS	MI per second
PSO	Particle swarm optimization
PSOM	PSO with mutation
SA	Simulated annealing
SSA	Salp swarm algorithm
VM	Virtual machines
WOA	Whale optimization algorithm

References

1. Statista Inc. Public Cloud Services End-User Spending Worldwide from 2017 to 2023. 2022. Available online: https://www.statista.com/statistics/273818/global-revenue-generated-with-cloud-computing-since-2009/ (accessed on 1 August 2023).
2. Statista Inc. Europe: Cloud Computing Market Size Forecast 2017–2030. 2022. Available online: https://www.statista.com/statistics/1260032/european-cloud-computing-market-size/ (accessed on 1 August 2023).
3. Guo, J.; Chang, Z.; Wang, S.; Ding, H.; Feng, Y.; Mao, L.; Bao, Y. Who Limits the Resource Efficiency of My Datacenter: An Analysis of Alibaba Datacenter Traces. In Proceedings of the Proceedings of the International Symposium on Quality of Service, New York, NY, USA, 24–25 June 2019; IWQoS '19; Article ID: 39; pp. 1–10. [CrossRef]
4. Katal, A.; Dahiya, S.; Choudhury, T. Energy efficiency in cloud computing data centers: A survey on software technologies. *Clust. Comput.* **2023**, *26*, 1845–1875. [CrossRef] [PubMed]
5. Ghafari, R.; Kabutarkhani, F.H.; Mansouri, N. Task scheduling algorithms for energy optimization in cloud environment: A comprehensive review. *Clust. Comput.* **2022**, *25*, 1035–1093. [CrossRef]
6. Jamil, B.; Ijaz, H.; Shojafar, M.; Munir, K.; Buyya, R. Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments: A Taxonomy, Review, and Future Directions. *ACM Comput. Surv.* **2022**, *54*, 233. [CrossRef]
7. Du, J.; Leung, J.Y.T. Complexity of Scheduling Parallel Task Systems. *SIAM J. Discret. Math.* **1989**, *2*, 473–487. [CrossRef]
8. Durasević, M.; Jakobović, D. Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: A survey. *Artif. Intell. Rev.* **2023**, *56*, 3181–3289. [CrossRef]
9. S., V.C.S.; S., A.H. Nature inspired meta heuristic algorithms for optimization problems. *Computing* **2022**, *104*, 251–269. [CrossRef]
10. Abualigah, L.; Elaziz, M.A.; Khasawneh, A.M.; Alshinwan, M.; Ibrahim, R.A.; Al-qaness, M.A.A.; Mirjalili, S.; Sumari, P.; Gandomi, A.H. Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: A comprehensive survey, applications, comparative analysis, and results. *Neural Comput. Appl.* **2022**, *34*, 4081–4110. [CrossRef]
11. Dinachali, B.P.; Jabbehdari, S.; Javadi, H.H.S. A pricing approach for optimal use of computing resources in cloud federation. *J. Supercomput.* **2023**, *79*, 3055–3094. [CrossRef]
12. Jahangard, L.R.; Shirmarz, A. Taxonomy of green cloud computing techniques with environment quality improvement considering: A survey. *Int. J. Energy Environ. Eng.* **2022**, *13*, 1247–1269. [CrossRef]
13. Chi, H.R.; Wu, C.K.; Huang, N.F.; Tsang, K.F.; Radwan, A. A Survey of Network Automation for Industrial Internet-of-Things Towards Industry 5.0. *IEEE Trans. Ind. Inform.* **2023**, *19*, 2065–2077. [CrossRef]
14. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef]
15. Houssein, E.H.; Gad, A.G.; Hussain, K.; Suganthan, P.N. Major Advances in Particle Swarm Optimization: Theory, Analysis, and Application. *Swarm Evol. Comput.* **2021**, *63*, 100868. [CrossRef]
16. Shao, K.; Song, Y.; Wang, B. PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing. *Mathematics* **2023**, *11*, 1548. [CrossRef]
17. Nabi, S.; Ahmed, M. PSO-RDAL: Particle swarm optimization-based resource- and deadline-aware dynamic load balancer for deadline constrained cloud tasks. *J. Supercomput.* **2022**, *78*, 4624–4654. [CrossRef]
18. Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing. *Sensors* **2022**, *22*, 920. [CrossRef] [PubMed]
19. Pirozmand, P.; Hosseinabadi, A.A.R.; Farrokhzad, M.; Sadeghilalimi, M.; Mirkamali, S.; Slowik, A. Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing. *Neural Comput. Appl.* **2021**, *33*, 13075–13088. [CrossRef]
20. Pradhan, R.; Satapathy, S.C. Energy Aware Genetic Algorithm for Independent Task Scheduling in Heterogeneous Multi-Cloud Environment. *J. Sci. Ind. Res.* **2022**, *81*, 776–784. [CrossRef]
21. Malti, A.N.; Hakem, M.; Benmammar, B. Multi-objective task scheduling in cloud computing. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7252. [CrossRef]
22. Mangalampalli, S.; Swain, S.K.; Mangalampalli, V.K. Multi Objective Task Scheduling in Cloud Computing Using Cat Swarm Optimization Algorithm. *Arab. J. Sci. Eng.* **2022**, *47*, 1821–1830. [CrossRef]
23. Mangalampalli, S.; Swain, S.K.; Mangalampalli, V.K. Prioritized Energy Efficient Task Scheduling Algorithm in Cloud Computing Using Whale Optimization Algorithm. *Wirel. Pers. Commun.* **2022**, *126*, 2231–2247. [CrossRef]
24. Aghdashi, A.; Mirtaheri, S.L. Novel dynamic load balancing algorithm for cloud-based big data analytics. *J. Supercomput.* **2022**, *78*, 4131–4156. [CrossRef]
25. Belgacem, A.; Beghdad-Bey, K. Multi-objective workflow scheduling in cloud computing: Trade-off between makespan and cost. *Clust. Comput.* **2022**, *25*, 579–595. [CrossRef]
26. Aktan, M.N.; Bulut, H. Metaheuristic task scheduling algorithms for cloud computing environments. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6513. [CrossRef]
27. Pradeep, K.; Ali, L.J.; Gobalakrishnan, N.; Raman, C.J.; Manikandan, N. CWOA: Hybrid Approach for Task Scheduling in Cloud Environment. *Comput. J.* **2022**, *65*, 1860–1873. [CrossRef]
28. Pirozmand, P.; Javadpour, A.; Nazarian, H.; Pinto, P.; Mirkamali, S.; Ja'fari, F. GSAGA: A hybrid algorithm for task scheduling in cloud infrastructure. *J. Supercomput.* **2022**, *78*, 17423–17449. [CrossRef]
29. Jain, R.; Sharma, N. A quantum inspired hybrid SSA–GWO algorithm for SLA based task scheduling to improve QoS parameter in cloud computing. *Clust. Comput.* **2022**, 1–24 . [CrossRef]

30. Cheikh, S.; Walker, J.J. Solving Task Scheduling Problem in the Cloud Using a Hybrid Particle Swarm Optimization Approach. *Int. J. Appl. Metaheuristic Comput.* **2022**, *13*, 1–25. [CrossRef]
31. Hafsi, H.; Gharsellaoui, H.; Bouamama, S. Genetically-modified Multi-objective Particle Swarm Optimization approach for high-performance computing workflow scheduling. *Appl. Soft Comput.* **2022**, *122*, 108791. [CrossRef]
32. Nwogbaga, N.E.; Latip, R.; Affendey, L.S.; Rahiman, A.R.A. Attribute reduction based scheduling algorithm with enhanced hybrid genetic algorithm and particle swarm optimization for optimal device selection. *J. Cloud Comput.* **2022**, *11*, 15. [CrossRef]
33. Xian Wang, B.; Wu, P.; Arefzaeh, M. A new method for task scheduling in fog-based medical healthcare systems using a hybrid nature-inspired algorithm. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7155. [CrossRef]
34. MathWorks, I. Optimization Toolbox: Solve Linear, Quadratic, Conic, Integer, and Nonlinear Optimization Problems. 2022. Available online: https://ww2.mathworks.cn/en/products/optimization.html (accessed on 1 August 2023).
35. Jong, K.A.D.; Spears, W.M. A formal analysis of the role of multi-point crossover in genetic algorithms. *Ann. Math. Artif. Intell.* **1992**, *5*, 1–26. [CrossRef]
36. Nabi, S.; Ahmed, M. OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained Cloud tasks. *J. Supercomput.* **2021**, *77*, 7476–7508. [CrossRef]
37. Hussain, A.A.; Al-Turjman, F. Hybrid Genetic Algorithm for IOMT-Cloud Task Scheduling. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 6604286. [CrossRef]
38. Wang, B.; Wang, C.; Huang, W.; Song, Y.; Qin, X. Security-aware task scheduling with deadline constraints on heterogeneous hybrid clouds. *J. Parallel Distrib. Comput.* **2021**, *153*, 15–28. [CrossRef]
39. Athmani, M.E.; Arbaoui, T.; Mimene, Y.; Yalaoui, F. Efficient Heuristics and Metaheuristics for the Unrelated Parallel Machine Scheduling Problem with Release Dates and Setup Times. In Proceedings of the Genetic and Evolutionary Computation Conference, Boston, MA, USA, 9–13 July 2022; GECCO '22; pp. 177–185. [CrossRef]
40. Teraiya, J.; Shah, A. Optimized scheduling algorithm for soft Real-Time System using particle swarm optimization technique. *Evol. Intell.* **2022**, *15*, 1935–1945. [CrossRef]
41. Wang, B.; Cheng, J.; Cao, J.; Wang, C.; Huang, W. Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction. *PeerJ Comput. Sci.* **2022**, *8*, e893. [CrossRef] [PubMed]