


Article

An Improved Genetic Algorithm with Swarm Intelligence for Security-Aware Task Scheduling in Hybrid Clouds

Yinfeng Huang ¹, Shizheng Zhang ² and Bo Wang ^{2,*} ¹ School of Engineering, University of Glasgow, Glasgow G12 8QQ, UK² Software Engineering School, Zhengzhou University of Light Industry, Zhengzhou 450002, China

* Correspondence: wangb@zzuli.edu.cn

Abstract: The hybrid cloud has attracted more and more attention from various fields by combining the benefits of both private and public clouds. Task scheduling is still a challenging open issue to optimize user satisfaction and resource efficiency for providing services by a hybrid cloud. Thus, in this paper, we focus on the task scheduling problem with deadline and security constraints in hybrid clouds. We formulate the problem into mixed-integer non-linear programming, and propose a polynomial time algorithm by integrating swarm intelligence into the genetic algorithm, which is named SPGA. Specifically, SPGA uses the self and social cognition exploited by particle swarm optimization in the population evolution of GA. In each evolutionary iteration, SPGA performs the mutation operator on an individual with not only another individual, as in GA, but also the individual's personal best code and the global best code. Extensive experiments are conducted for evaluating the performance of SPGA, and the results show that SPGA achieves up to a 53.2% higher accepted ratio and 37.2% higher resource utilization, on average, compared with 12 other scheduling algorithms.

Keywords: cloud computing; genetic algorithm; hybrid cloud; swarm intelligence; task scheduling



Citation: Huang, Y.; Zhang, S.; Wang, B. An Improved Genetic Algorithm with Swarm Intelligence for Security-Aware Task Scheduling in Hybrid Clouds. *Electronics* **2023**, *12*, 2064. <https://doi.org/10.3390/electronics12092064>

Academic Editor: Dimitris Kanellopoulos

Received: 31 January 2023

Revised: 23 March 2023

Accepted: 18 April 2023

Published: 29 April 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Nowadays, cloud computing has become the platform for service delivery in almost all fields, due to its various advantages. One of the greatest benefits of cloud computing is that it provides “infinite” resources for its users, but service providers, especially small and medium-size enterprises (SMEs), have limited physical resources in their own private clouds in the real world. Thus, when the loads are increased, a private cloud may not be able to afford all user requests. This problem can be solved by following three approaches [1]. One approach is rejecting some user requests. This method can reduce the profit and the reputation for a service provider [2]. By the second approach, the provider increases the scale of the cloud by adding enough infrastructures to meet the users' peak demand. However, this approach requires a high investment cost, which impedes its adoption, as most of SMEs cannot afford it. In addition, the peak load is usually transient [3], which makes the second approach inefficient. The third approach exploits the hybrid cloud technology, where the provider temporarily extends its private cloud by renting on-demand resources from public clouds, when some user demand cannot be met. Therefore, most SMEs prefer a mixture of cloud computing models [4]. The hybrid cloud model has been applied in many fields, e.g., healthcare [5], virtual reality [6], smart farms [7], and robotics [8].

For providing services by hybrid clouds, task scheduling is one of the most important problems that the service provider must address, to optimize resource efficiency and various quality of service (QoS) satisfactions [1,9]. Task scheduling involves deciding what resource to use for every task's processing, with varying performance requirements. Unfortunately, the task scheduling problem is NP-hard [10], and no exact algorithm can be applied for

medium- and large-scale clouds, as the time complexity of an exact algorithm is increased exponentially with the problem size. There are mainly two categories of algorithms, heuristics and meta-heuristics, for solving task scheduling with polynomial time. Heuristics provide approximate optimal solutions by local search strategies specifically designed for the problem. Meta-heuristics are designed according to natural and social rules, and use some random and generally global search methods. In general, meta-heuristics can achieve better performance than heuristics, but with more time overheads. Based on the “No free lunch” theorems [11], all heuristic and meta-heuristic methods have their own strengths and weaknesses. Thus, one promising way to design hybrid heuristic algorithms is by combining the advantages of multiple methods for better performance. Even though there are several works proposing hybrid heuristic scheduling algorithms for hybrid clouds, most of them only perform two or more methods separately, which leads to an inefficient combination. Therefore, in this paper, we focus on designing an integration method with high efficiency for combining multiple meta-heuristic algorithms for enhancing the performance of task scheduling in hybrid clouds. For addressing task scheduling in hybrid clouds, existing works have simplified the problem using some assumptions. For example, some works are not concerned with the security requirements of tasks when they are processed by the public cloud, even though security is one of the top concerns for some users, especially enterprise users [1]. Several works ignored the resource heterogeneity between the private and public clouds. These simplifications lead to inefficient or even infeasible applications of these works.

To optimize user satisfaction and resource efficiency with security and deadline requirements, we design a hybrid heuristic algorithm (named SPGA) for security-aware task scheduling in hybrid clouds, by combining the genetic algorithm (GA) and particle swarm optimization (PSO)—two representative and popular meta-heuristics. We integrate the social behavior exploited by PSO into GA to overcome the low convergence rate of GA [12] and the PSO problem of being easily trapped into local optima [13]. To be specific, SPGA uses the code (gene values of every chromosome, particle position) of each individual (chromosomes for GA, particle for PSO) to represent a task–resource assignment. To achieve the best assignment, SPGA exploits the evolution framework of PSO, and the evolution operators of GA with the self and social cognition for the population evolution. To improve the performance of decoded assignments, SPGA re-assigns tasks with unmet requirements from one resource to another, which helps improve user satisfaction by increasing the number of accepted tasks. The main contributions of this paper can be summarized as follows.

- This paper models the task scheduling problem of hybrid clouds into mixed-integer non-linear programming (MINLP). There are two optimization objectives. The major one is maximizing the accepted ratio, which is one of the most common metrics for quantifying user satisfaction. The minor one is maximizing resource utilization, which is a popular measurement for resource efficiency.
- This paper proposes a security-aware hybrid PSO and GA algorithm, SPGA, to resolve the task scheduling problem in polynomial time. SPGA integrates the self and social cognition into GA to combine the fast convergence rate of PSO and the powerful global search ability of GA. In addition, SPGA takes into account the security constraints during the search process.
- Extensive experiments are conducted to evaluate the performance of SPGA. Results show that SPGA achieves a 13.6–53.2% higher accepted ratio and 0.43–37.2% higher resource utilization, on average, compared with 12 other scheduling algorithms based on heuristics, meta-heuristics, and hybrid heuristics.

In the following, we formulate the task scheduling problem in Section 2, and propose the hybrid heuristic algorithm to solve the scheduling problem with polynomial time in Section 3. In Section 4, we evaluate our proposed algorithm with extensive simulated experiments. Related works are analyzed in Section 5. We conclude our work and present some future works in Section 6.

2. Problem Formulation

In a hybrid cloud, there are one or more private/local clouds and public clouds, as shown in Figure 1. To keep the description concise, we assume that a hybrid cloud consists of one private cloud and one public cloud. Users send their requests to the service provider for various services. The service provider assigns each request task to a private/local resource based on a scheduling strategy. When the private resources are not enough to satisfy some tasks' requirements, the provider rents resources from the public cloud for these tasks. There are some tasks with security requirements. These tasks can be processed by private resources only, as the public resources are shared by various Internet users and cannot generally guarantee security [14,15]. Sometimes, for a request task, neither the private nor the public cloud can meet its requirements. In such cases, the provider has to reject the request. A task scheduling solution must decide on the resource where each task is assigned for processing. Next, we formulate the scheduling problem, optimizing the user satisfaction and resource efficiency, with deadline and security requirements of user request tasks.

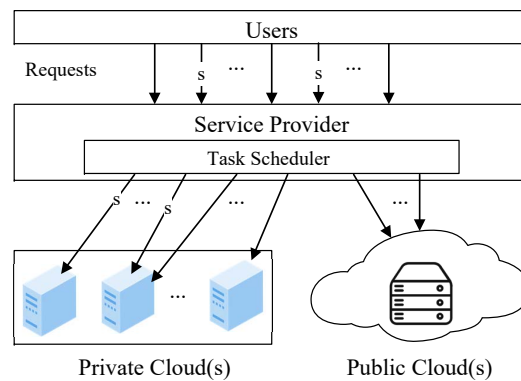


Figure 1. The architecture of hybrid clouds.

2.1. Hybrid Cloud System Model

In the considered hybrid cloud system, the private cloud has P servers, $p_j, j = 1, \dots, P$, which provide the security environments for processing user requests. The local servers can be physical machines (PM), virtual machines (VM), or even both, as the provider owns the private cloud and can operate the underlying infrastructures. In server p_j , the number of computing cores is c_j , each with g_j computing capacity. We assume that data are organized by a distributed file system (DFS) in the private cloud, which is one of the most popular ways used by various distributed and parallel systems. The read bandwidth in DFS is r_j for data transfers required by task processing on server p_j .

The public cloud provides computing resources in the form of a virtual machine (VM). There are V VM types ($v_k, k = 1, \dots, V$) provided by the public cloud. The configurations of VM type v_k are c_k^v computing cores, and each core has g_k^v computing capacity. The downlink bandwidth of v_k is b_k . The price of v_k is f_k per hour.

T tasks, $t_i, i = 1, \dots, T$, are requested by users to be processed by the hybrid cloud. For task t_i , the computing resource required by its completion is h_i , and the amount of input data is a_i . In this paper, we assume the transmission latencies of returning results to users are short enough to be ignored because the resulting data are much less than the input data most of the time. There are two requirements for each task, the deadline and the security. The deadline of t_i is d_i —when the task must be finished before. The security requirement (s_i) of every task is considered as a binary option in this paper. If t_i has the security requirement ($s_i = 1$), it can be processed only by the private clouds, and otherwise ($s_i = 0$), by any of the hybrid resources. When either requirement cannot be satisfied for a task, the provider rejects its request as it consumes resources without any pay for its processing.

2.2. Task Processing Model

To satisfy all user requirements, n_k VM instances ($w_{k,l}, l = 1, \dots, n_k$) with type v_k are rented from the public cloud. For the formulation of the task scheduling problem, we define the following binary variables, $x_{i,j,m}$ and $y_{i,k,l,m}$, to represent the task-core assignment solution, as Equations (1) and (2).

$$x_{i,j,m} = \begin{cases} 1, & \text{if } t_i \text{ is assigned to the } m\text{th core in } p_j, 1 \leq i \leq T, 1 \leq j \leq P, 1 \leq m \leq c_j. \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$y_{i,k,l,m} = \begin{cases} 1, & \text{if } t_i \text{ is assigned to the } m\text{th core in VM instance } w_{k,l}, \\ 0, & \text{otherwise} \end{cases}, \quad (2)$$

$i = 1, \dots, T, k = 1, \dots, V, l = 1, \dots, n_k, m = 1, \dots, c_k^v.$

Each task can only be processed by one computing core when it is accepted (in this paper, we do not consider task redundancy for improving the performance of task processing, due to its redundant resource consumption). Then, these binary variables satisfy the constraints of Equation (3). The number of accepted tasks N can be achieved by Equation (4), where z_i is calculated by Equation (5), which is an indicator of whether task t_i is accepted in the hybrid cloud. The accepted ratio can be calculated by Equation (6).

$$\sum_{j=1}^P \sum_{m=1}^{c_j} x_{i,j,m} + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} y_{i,k,l,m} \leq 1, i = 1, \dots, T. \quad (3)$$

$$N = \sum_{i=1}^T z_i. \quad (4)$$

$$z_i = \sum_{j=1}^P \sum_{m=1}^{c_j} x_{i,j,m} + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} y_{i,k,l,m}, i = 1, \dots, T. \quad (5)$$

$$AR = \frac{N}{T} \times 100\%. \quad (6)$$

Tasks with security requirements cannot be assigned to any VM instance rented from the public cloud. This can be formulated as Equation (7).

$$\sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} y_{i,k,l,m} \leq 1 - s_i, i = 1, \dots, T. \quad (7)$$

As the computing can be started only when the input data transfer is finished for each task, the inequalities (8) hold. b_i^{data} and e_i^{data} are the start (begin) and finish (end) time of data transfer of t_i . b_i^{com} and e_i^{com} represent the computing begin and end time of t_i .

$$e_i^{data} \leq b_i^{com}, i = 1, \dots, T. \quad (8)$$

When task t_i is assigned to a core of p_j , the time consumed by the input data transfer is a_i/r_j , and its computing time is h_i/g_j . For task t_i assigned to one core of a VM with type v_k , the input data transfer and the computing consume a_i/b_k and h_i/g_k^v time, respectively. Then, the following equalities hold. When a task is not accepted, its begin and end time are 0, and Equations (8)–(10) also hold.

$$e_i^{data} = b_i^{data} + \sum_{j=1}^P \sum_{m=1}^{c_j} x_{i,j,m} \cdot \frac{a_i}{r_j} + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} y_{i,k,l,m} \cdot \frac{a_i}{b_k}, i = 1, \dots, T. \quad (9)$$

$$e_i^{com} = b_i^{com} + \sum_{j=1}^P \sum_{m=1}^{c_j} x_{i,j,m} \cdot \frac{h_i}{g_j} + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} y_{i,k,l,m} \cdot \frac{h_i}{g_k^v}, i = 1, \dots, T. \tag{10}$$

In the hybrid cloud, we consider that the input data are transferred sequentially for multiple tasks assigned to one core, which can ensure the performance of data operators from one hard disk or network card by avoiding I/O interferences [16]. In addition, we consider non-pre-emptive tasks to avoid the overhead of frequent context switching. Thus, there is no overlap time between data transfers and computing of two tasks on each core. Then, when tasks t_{i1} and t_{i2} are assigned to one core, we have Equations (11) and (12), which can be reformulated as Equations (13) and (14) for all cases, respectively. Where, $\sum_{j=1}^P \sum_{m=1}^{c_j} (x_{i1,j,m} \cdot x_{i2,j,m}) + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} (y_{i1,k,l,m} \cdot y_{i2,k,l,m})$ is 1 when t_{i1} and t_{i2} are assigned to one core, and 0, otherwise.

$$\begin{cases} e_{i1}^{data} \leq b_{i2}^{data}, & \text{if the input data of } t_{i1} \text{ is transferred before that of } t_{i2}, i1, i2 = 1, \dots, T. \\ e_{i2}^{data} \leq b_{i1}^{data}, & \text{otherwise} \end{cases} \tag{11}$$

$$\begin{cases} e_{i1}^{com} \leq b_{i2}^{com}, & \text{if the computing of } t_{i1} \text{ is started before that of } t_{i2}, i1, i2 = 1, \dots, T. \\ e_{i2}^{com} \leq b_{i1}^{com}, & \text{otherwise} \end{cases} \tag{12}$$

$$\left(\sum_{j=1}^P \sum_{m=1}^{c_j} (x_{i1,j,m} \cdot x_{i2,j,m}) + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} (y_{i1,k,l,m} \cdot y_{i2,k,l,m}) \right) \cdot (b_{i2}^{data} - e_{i1}^{data}) \cdot (b_{i1}^{data} - e_{i2}^{data}) \leq 0, i1, i2 = 1, \dots, T. \tag{13}$$

$$\left(\sum_{j=1}^P \sum_{m=1}^{c_j} (x_{i1,j,m} \cdot x_{i2,j,m}) + \sum_{v=1}^V \sum_{l=1}^{n_k} \sum_{m=1}^{c_k^v} (y_{i1,k,l,m} \cdot y_{i2,k,l,m}) \right) \cdot (b_{i2}^{com} - e_{i1}^{com}) \cdot (b_{i1}^{com} - e_{i2}^{com}) \leq 0, i1, i2 = 1, \dots, T. \tag{14}$$

For each core, its occupied time for task processing is the latest end time of tasks assigned to it. The occupied time of one private server or public VM instance is the maximum occupied time of its cores. Thus, the occupied time of each server or VM instance can be calculated by Equations (15) and (16). Where o_j and $o_{k,l}^v$ are the occupied time of server p_j and VM instance $w_{k,l}^v$, respectively. The occupied time of VM instances are rounded up to full hours, as VM instances are charged by the hour.

$$o_j = \max_{m=1}^{c_j} \max_{i=1}^T \{x_{i,j,m} \cdot e_i\}, j = 1, \dots, P. \tag{15}$$

$$o_{k,l}^v = \lceil \frac{\max_{m=1}^{c_k^v} \max_{i=1}^T \{y_{i,k,l,m} \cdot e_i\}}{3600} \rceil \times 3600, k = 1, \dots, V, l = 1, \dots, n_k. \tag{16}$$

Then, the rent cost for VM instance $w_{k,l}$ is $f_k \cdot o_{k,l}^v$, and the total cost for renting all VM instances is

$$C = \sum_{k=1}^V \sum_{l=1}^{n_k} (f_k \cdot o_{k,l}^v). \tag{17}$$

The computing resource utilization is the ratio of computing resources consumed by processing tasks to the occupied computing resources. Thus, the resource utilization of each server or VM instance can be calculated by Equations (18) and (19), and the overall resource utilization of the hybrid cloud can be achieved by Equation (20).

$$u_j = \frac{\sum_{m=1}^{c_j} \sum_{i=1}^T (x_{i,j,m} \cdot h_i)}{o_j \cdot g_j}, j = 1, \dots, P. \tag{18}$$

$$u_{k,l}^v = \frac{\sum_{m=1}^{c_k^v} \sum_{i=1}^T (y_{i,k,l,m} \cdot h_i)}{o_{k,l}^v \cdot g_k^v}, k = 1, \dots, V, l = 1, \dots, n_k. \tag{19}$$

$$U = \frac{z_i \cdot h_i}{\sum_{j=1}^P (o_j \cdot g_j) + \sum_{k=1}^V \sum_{l=1}^{n_k} (o_{k,l}^v \cdot g_k^v)}. \quad (20)$$

2.3. Task Scheduling Model

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

$$\text{maximizing } N + U. \quad (21)$$

Where constraints include Equations (1)–(20) and the deadline requirements, Equation (22).

$$e_i \leq d_i, i = 1, \dots, T. \quad (22)$$

The decision variables include $x_{i,j,m}, i = 1, \dots, T, j = 1, \dots, P, m = 1, \dots, c_j$, and $y_{i,k,l,m}, i = 1, \dots, T, k = 1, \dots, V, l = 1, \dots, n_k, m = 1, \dots, c_k^v$, which are binary and decide the task-core assignments, as well as b_i^{data} and $b_i^{com}, i = 1, \dots, T$, which are real numbers and imply the processing order of tasks on each core. The objective (21) is to maximize the number of accepted tasks plus the overall resource utilization. As the resource utilization is no more than 1, the maximization of the accepted task number is the major optimization objective, which is identical to the optimization of the accepted ratio because the number of tasks, T , is fixed. Some constraints are non-linear, e.g., Equation (15). Thus, the optimization problem is mixed-integer non-linear programming (MINLP). The problem can be solved by existing MINLP solvers, e.g., the Optimization Toolbox of Matlab. However, these tools are only applicable for small-sized problems because they are exponential-average complexity. Thus, in the next section, we propose a polynomial algorithm for solving the problem.

3. SPGA: Security-Aware Hybrid Heuristic Scheduling Algorithm

Meta-heuristics have been employed for solving various optimization and decision-making problems, inspired by nature and society laws [17]. GA and PSO are two of most popular meta-heuristics due to their good performance and easy implementation. GA simulates Charles Darwin's theory of evolution. GA has powerful global search ability, which benefits from its evolution operators, but slow convergence speed. PSO imitates flocking bird's foraging. In general, PSO has a fast convergence rate, but can be easily trapped into local optima, due to the exploitation of self and social cognition. Thus, GA and PSO complement each other well. Next, we illustrate our proposed method, SPGA, which combines PSO and GA for task scheduling, aware of security requirements. The process of SPGA is outlined in Algorithm 1.

First, we design an encoding/decoding method to map between codes used by SPGA and task scheduling solutions, and a fitness function to evaluate the goodness of individuals with various codes. Based on the designed encoding method and the fitness function, SPGA executes the population evolution process with a similar framework to PSO with self and social cognition, and employs the crossover and mutation operators of GA for updating individuals. The fitness function is the optimization objective presented in the previous section, i.e., $N + U$, when applying the corresponding task solution for each code. Therefore, the fitness evaluation of an individual first maps its code into a task solution, and then collects information, aggregating N and calculating U by Equation (20). Then, the fitness value of the individual is $N + U$.

To exploit the self and social cognition of PSO in GA, we introduce a personal best code for each individual, and a global best code. For every individual, the personal best code recodes the best one in all codes, initialized or updated, from start till now. The global one recodes the best personal best code of all individuals all the time.

As shown in Algorithm 1, the first SPGA initializes a population consisting of multiple individuals, where each code is randomly set in the range of allowed values (line 1). The fitness value is evaluated for each individual (line 4), and the personal best code is set as the initialized code for each individual (line 6). The global best code is initialized as the code with

the best fitness in all individuals (line 8). After the initialization, SPGA evolves the population by performing the crossover and mutation operators with self and social cognition.

SPGA repeats the following steps until the terminal condition is reached (line 9). First, SPGA performs the crossover operators on each individual with a certain probability (the crossover probability), three times. For an individual, these three crossover operators are performed with another random individual, its personal best code, and the global best code, respectively (line 11). These three performed operators are, respectively, corresponding with the inertia, the self-cognition, and the social cognition in the position update of PSO. A cross can produce two new codes (offspring), and the crossover operator produces six offspring for each individual at most. For these offspring produced from an individual, SPGA evaluates their fitness (line 12), and replaces the individual with the offspring with the best fitness (line 13). Meanwhile, SPGA updates the personal best code of each individual to the best offspring if the best offspring has a better fitness (line 15), and performs the same action on the global best code (line 17). Then, SPGA performs the mutation on each individual with set mutation probability (line 18), and evaluates its fitness (line 19). When the new individual has better fitness than its personal/the global best code, SPGA updates the personal/global best code as the new one (line 20).

After the iteration finishes when the terminal condition is reached, SPGA achieves and returns the best task solution decoded by the global best code (lines 21 and 22). The fitness evaluation and the decoding methods are shown in Algorithm 2, detailed in Section 3.1.

Algorithm 1 SPGA: Security-aware hybrid PSO and GA scheduling

Require: information of request tasks and available hybrid cloud resources

Ensure: a task scheduling solution

- 1: Initializing a population;
- 2: **if** the storage of the population run out of memory **then**
- 3: **return** NULL; //an error
- 4: Evaluating fitness of each individual using Algorithm 2;
- 5: Initializing the global best code as the code of the first individual;
- 6: **for** each individual **do** Initializing its personal best code as its current code;
- 7: **if** the current code has better fitness than the global best code **then**
- 8: Updating the global best code as the current code
- 9: **while** terminal condition isn't reached **do**
- 10: **for** each individual **do**
- 11: Crossing it with another individual, its personal best and the global best codes, respectively, with the crossover probability;
- 12: Evaluating fitness of each offspring;
- 13: Replacing the individual with its best offspring;
- 14: **if** the individual has better fitness than its personal best code **then**
- 15: Updating its personal best code as the current code of the individual;
- 16: **if** the individual has better fitness than the global best code **then**
- 17: Updating the global best code as the current code of the individual;
- 18: Mutating each individual, with the mutation probability;
- 19: Evaluating fitness of mutated individuals;
- 20: Updating its personal and the global best codes, same to lines 14–17;
- 21: Decoding the global best code into a task scheduling solution, using Algorithm 2;
- 22: **return** the task scheduling solution;

There are mainly two kinds of terminal conditions set for SPGA or other meta-heuristic-based algorithms. The first is setting the maximum iteration number, and the second one is the maximum number of continuous iterations that the global best fitness does not change. In this paper, we adopt the first approach.

3.1. Encoding/Decoding Method

In SPGA, the task assignment solution is encoded into a code of individuals. The dimension of a code has a one-to-one correspondence with tasks, and the value in a dimension indicates the resource that the corresponding task is assigned. The candidate resources are different for tasks. Tasks with security requirements can be processed in the private cloud only while other tasks can be processed in both the private and public clouds. Thus, the value range of a dimension is 1 to the number of cores in the private cloud for tasks with security requirements. This ensures that tasks with security requirements can never be assigned to the public cloud. For tasks without security requirements, which can be processed by both private and public clouds, we use extra possible value(s) to represent the public cloud that corresponding tasks are assigned to.

In the initialization phase, the value is randomly set in each dimension for every individual, usually by uniform distribution. If we use only one extra value to represent the task assigned to the public cloud, then the probability of the public cloud that a task is assigned to is the same as that of a private core. This can make SPGA perform well when the private cloud has enough or only lacks a few resources for satisfying user requirements. However, this is unreasonable when user loads are heavy. At this point, the private cloud will be overloaded due to only a small number of tasks assigned to the public cloud, even though the public cloud can generally provide much more resources than the private cloud. In this paper, we set the maximum value as twice the number of private cores. This means that the private and public clouds have identical probabilities for each assignment of a task without security requirements.

For example, in a hybrid cloud, there are a total of 10 computing cores (c_1, \dots, c_{10}) in the private cloud, and 8 tasks (t_1, \dots, t_8). Then, the number of dimensions is 8 for each individual code. If the first four tasks have security requirements, while the last four tasks do not, the possible values are 1–10 and 1–20 in the first and second dimensions, respectively. Values of 1–10 correspond to the 10 cores, and 11–20 correspond to the public cloud, for each task assignment. For example, the core $\langle 3, 3, 5, 7, 7, 11, 12, 15 \rangle$ represents that t_1 and t_2 are assigned to c_3 , t_3 to c_5 , t_4 and t_5 to c_7 , and t_6 , t_7 , and t_8 to the public cloud.

Now, we can achieve a task assignment solution from each code (line 1 in Algorithm 2). To provide a complete solution of task scheduling, SPAG should decide the processing order of tasks assigned to the same core or the public cloud. In this paper, we exploit classical heuristic algorithms, earliest deadline first (EDF) and first fit (FF), to decide the task processing orders, which are simple but effective. The heuristic method for the processing order is also a promising research direction for improving the performance of task scheduling, and there are several works focusing on it. These works are complementary to our work, and we will study it and design new ordering methods complementary to our method in the future.

As shown in Algorithm 2, given the assignment solution decoded from a code, SPAG adopts EDF to order the processing of tasks on each core (line 2), due to its awareness of deadline and its better performance over other heuristic methods (FF, FFD, and SJF), as shown in our experimental results. For tasks assigned to the public clouds, SPAG uses FF, which iteratively schedules every task to the first VM instance meeting the deadline constraint (line 3). As the public cloud is assumed to provide “infinite” VM instances, all kinds of scheduling algorithms can achieve an identical number of accepted tasks in the public cloud. This is because one VM instance can be rented for each task to achieve the maximum number of accepted tasks processed by the public cloud, when applying any scheduling algorithm. Thus, in this paper, we use one of the simplest methods, FF, as the accepted ratio is our major objective. However, various scheduling algorithms provide a different resource efficiency for the public cloud. Thus, the design of highly efficient scheduling algorithm is one of our future works.

Algorithm 2 Decoding a code into a task scheduling solution

Require: a code

Ensure: the corresponding task scheduling solution, and the fitness

- 1: Decoding the code into assignments of tasks to private computing cores or the public cloud;
 - 2: For each private computing core, scheduling tasks with EDF, and accumulating the accepted task number;
 - 3: For the public cloud, scheduling tasks with FF, and accumulating the accepted task number;
 - 4: For all tasks with deadline conflicts, scheduling them on available hybrid resources by FF, and accumulating the accepted task number;
 - 5: Calculating the resource utilization by Equation (20);
 - 6: Calculating the fitness by adding the accepted task number to the resource utilization
 - 7: **return** the task scheduling solution and the fitness;
-

The task assignment solution obtained from one code does not consider the load balance among private computing cores, and thus may lead to overloads for some cores and underloads for others. This can decrease both the accepted ratio and the resource efficiency. Thus, after the above task scheduling (assignment and ordering) for all private cores and the public cloud, SPGA re-schedules rejected tasks in private cores by FF (line 4), to more fully utilize the private resources for processing more tasks.

SPGA accumulates accepted tasks in the above processes, and calculates the overall resource utilization by Equation (20). Then, the fitness of the code can be achieved by adding the accepted task number to the overall resource utilization. By this time, the task solution and fitness are provided according to a code.

3.2. Crossover Operator

By exploiting the self and social cognition used for the position update in PSO, SPGA crosses an individual with not only another individual, as done by GA, but also its personal and the global best codes. Each performed crossover operator produces two new codes (offspring) from two origin codes (parents). In this paper, to ensure population diversity, we adopt the uniform crossover operator in SPGA.

Given two codes, the uniform crossover operator generates a random value in the range of 0 to 1, for each dimension. If the random value is less than the set probability, two codes swap their values in the dimension, and otherwise, performs no action.

For example, as shown in Figure 2, the uniform crossover operator is performed on two codes, $\langle 4, 9, 14, 14, 5, 4, 10, 8 \rangle$ and $\langle 5, 17, 11, 13, 14, 19, 13, 10 \rangle$. The random values generated for all dimension are $\langle 0.06, 0.62, 0.29, 0.51, 0.99, 0.46, 0.08, 0.85 \rangle$. If the probability is set to 0.5 for crossing in each dimension, then the operator conducts the crossing (swap) operation on the first, third, sixth, and seventh dimensions, and two new codes are produced, which are $\langle 5, 9, 11, 14, 5, 19, 13, 8 \rangle$ and $\langle 4, 17, 14, 13, 14, 4, 10, 10 \rangle$.

Dimension:	1	2	3	4	5	6	7	8
Random values:	0.06	0.62	0.29	0.51	0.99	0.46	0.08	0.85
Code (parent) 1:	4	9	14	14	5	4	10	8
Code (parent) 2:	5	17	11	13	14	19	13	10
	↓		↓			↓	↓	
New code (offspring) 1:	5	9	11	14	5	19	13	8
New code (offspring) 2:	4	17	14	13	14	4	10	10

Figure 2. An example for illustrating the crossover operator, where the crossing probability is set to 0.5 in each dimension.

3.3. Mutation Operator

The mutation operator helps to improve the population diversity by producing new codes from one code, trying to reach solution regions that have not been searched, and thus make the global search ability powerful for GA. SPGA performs the uniform mutation operator on each individual with the mutation probability. For the implementation of the uniform mutation operator on an individual, SPGA generates a random value between 0 and 1 for every dimension. If the generated random value is smaller than the probability set for the mutating, SPGA changes the value into another possible value in the dimension of the individual code. Next, we present an example to help readers understand the uniform mutation operator.

There is a code with eight dimensions, which is $\langle 6, 5, 9, 9, 14, 1, 8, 15 \rangle$, as shown in Figure 3. For all dimensions, respectively, SPGA generates eight random values that are $\langle 0.05, 0.85, 0.22, 0.79, 0.64, 0.26, 0.10, 0.55 \rangle$. The probability for mutating a dimension is set to 0.4, then, the values of the code are changed into first, third, sixth, and seventh dimensions, as the random values are greater than 0.4 in these dimensions, and the new code $\langle 8, 5, 5, 9, 14, 17, 13, 15 \rangle$ is produced.

Dimension:	1	2	3	4	5	6	7	8
Random values:	0.05	0.85	0.22	0.79	0.64	0.26	0.10	0.55
Code (parent):	6	5	9	9	14	1	8	15
	↓		↓			↓	↓	
New code (offspring):	8	5	5	9	14	17	13	15

Figure 3. An example for illustrating the mutation operator, where the probability for mutating in a dimension is set to 0.4.

3.4. Complexity Analysis

For a crossover or mutation operator, SPGA goes through every dimension for each individual. Thus, in each iteration of SPGA, the time complexity is $O(T \cdot POP \cdot CF)$, where POP is the number of individuals, and CF is the complexity of the fitness evaluation. In the process of a fitness evaluation, SPGA sequentially applies EDF, FF, and FF for the ordering in each core, the scheduling in the public cloud, and the re-scheduling in the private cloud. The time complexities of EDF and FF are $O(T^2)$ and $O(T)$ in each resource unit, respectively. Thus, the fitness evaluation has the time complexity of $O(T^2 \cdot NC + T \cdot NV + T \cdot NC) = O(T^2 \cdot NC + T \cdot NV)$, where NC is the total number of cores in the private cloud, and NV is the number of rented VM instances from the public cloud. Thus, each iteration of the population evolution has $O(POP \cdot (T^3 \cdot NC + T^2 \cdot NV))$ time complexity, and the time complexity of SPGA is $O(ITE \cdot POP \cdot (T^3 \cdot NC + T^2 \cdot NV))$, where ITE is the number of iterations. This time complexity is the same as most meta-heuristic algorithms, including GA and PSO.

4. Results

In this section, we compare our method with several classical and state-of-the-art methods by conducting extensive simulated experiments, to verify the superior performance of SPGA. We first illustrate the experiment environment, and then analyze the experiment results, in the following subsections.

4.1. Experiment Environment

We establish a simulated hybrid cloud system consisting of one private and one public cloud, referring to [18–21], with various random parameters. In the private cloud, there are 10 servers. The number of cores is set in the range of 2 to 32 in each server, and the computing capacity of each core is set between 1000 and 4000 million instructions per second (MIPS). The read bandwidth is randomly set in the range of 100 to 1000 MB/s for each server in the private cloud. The public cloud provides four VM types, and their configuration sets of the computing core and capacity are same as that of private servers.

The network bandwidth for data transfer is set between 10 and 100 Mbps. The price of each VM type is 0.1–1 dollar per hour.

One thousand tasks are random generated to be processed in the hybrid cloud. Each task requires [1000, 100,000] million instructions (MIs) computing resource (size), and has [10, 1000] MB input data to be processed. The deadline of a task is set between 1 and 100 s. The probability that a task has the security requirement is half.

We compared our method with the following works. For methods using the crossover and mutation operators, the crossover probability for each individual and the crossing probability for each dimension were both set to 0.5. The mutation probability for each individual and the mutating probability for each dimension were both set to 0.1. For PSO, PSOM, GA+PSO, and GAPSO, the acceleration coefficients were set to 2.0, and the inertia weight was linearly decreasing from 0.9 to 0.4.

- FF is one of the most classic and most common scheduling methods for various computing systems. FF iteratively processes every task in the first computing unit (private core or public VM instance) that can satisfy its requirements.
- FFD (first fit decreasing) prioritizes the task with maximal computing size on the first fit computing unit.
- EDF processes tasks in the order of ascending deadline.
- SJF is contrary to FFD, which prioritizes the task with the smallest computing size.
- HC (hill climbing) is a representative meta-heuristic algorithm with one-point search, and is the basic idea used for task scheduling in several works [22,23]. Given a start point, HC recursively replaces it with its better neighbor, until no neighbor is better than the current point.
- GA is one of the most representative and popular meta-heuristic algorithms, and has been adopted in many task scheduling methods, e.g., [24,25]. GA evolves a population by selection, crossover, and mutation operators, sequentially, in each iteration.
- GAHC (GA with HC) is the method proposed by Hussain and Al-Turjman [20]. This method replaces each individual with its better neighbor (HC operator) before every population evolution. In addition, GAHC replaces every individual with its best offspring (replacement operator) after each evolution, without performing the selection operator for the population evolution.
- GAR (GA with replacement) is the same as GAHC except that GAR does not perform the HC operator. GAR adopts the replacement operator to replace the selection operator in GA.
- PSO is one of most commonly used meta-heuristics, e.g., [26], which evolve the population based on the behavior of looking for food by bird flocks.
- PSOM (PSO with mutation operator) is the algorithm employed by Hafsi et al. [27]. In the end of each evolution, PSOM performs the mutation operator on individuals, to improve the population diversity for PSO.
- GA+PSO was proposed by Nwogbaga et al. [28]. This method respectively adopts GA and PSO in the first and second half phases of the whole population evolution.
- GAPSO is the algorithm used in [29]. In each iteration of the population evolution, GAPSO performs GA and PSO sequentially on each individual's evolution.

We compare SPGA with the above methods in the following three aspects.

- User satisfaction determines the income and reputation of the cloud service provider, at a large extent, and is usually quantified by the accept ratio of user requests. Three metrics are adopted for user satisfaction in our experiments: the accepted ratio (AR), which is the major optimization objective of our work, the computing size of the accepted tasks (H calculated by Equation (23)), and the processed input data for accepted tasks (A calculated by Equation (24)).

$$H = \sum_{i=1}^T (z_i \cdot h_i). \quad (23)$$

$$A = \sum_{i=1}^T (z_i \cdot a_i). \tag{24}$$

- Resource efficiency can greatly affect the cost of service provisioning and the green level of cloud operation. In this paper, we use the overall computing resource utilization (U) for the measurement of resource efficiency, which is one of the most commonly used metrics. In addition, we adopt the energy efficiency and the cost efficiency to quantify the resource efficiency of the private and public clouds, respectively. There are two metrics for each of the energy and cost efficiencies, which are the computing size finished and the input data processed by per unit of energy or cost. These four metric values are respectively calculated by the following equations, where in the superscripts of the left hand sides, $size$ and $data$ represent the finished computing size and processed data, respectively, and e and c represent the energy and cost efficiencies, respectively. E is the energy consumption of private servers, which is evaluated by a popular linear model, as shown in Equation (29). Where $u_j(\tau)$ is the computing resource utilization of server p_j , which is changed with time τ . W_j^{idle} and W_j^{full} are the power consumed by server p_j when its resource utilization is 0 and 100%, respectively. The values of W_j^{idle} and W_j^{full} are set referring to [21]. $\langle W_j^{idle}, W_j^{full} \rangle$ is set to $\langle 110, 175 \rangle$, $\langle 125, 210 \rangle$, $\langle 210, 300 \rangle$, and $\langle 350, 500 \rangle$ for servers with [1, 8], [8, 16], [16, 24], and [24, 32] cores, respectively.

$$R^{com,e} = \frac{H}{E} \tag{25}$$

$$R^{data,e} = \frac{A}{E} \tag{26}$$

$$R^{com,c} = \frac{H}{C} \tag{27}$$

$$R^{data,c} = \frac{A}{C} \tag{28}$$

$$E = \sum_{j=1}^P \int_{\tau} (W_j^{idle} + (W_j^{full} - W_j^{idle}) \cdot u_j(\tau)) d\tau. \tag{29}$$

- Processing efficiency is the load processed per time unit, which reflects the speed of parallel computing. Two metrics, the finished computing size and the processed data amount per time unit (\mathfrak{S}^{com} and \mathfrak{S}^{data}), are used in our experiments, which are calculated by the following equations, respectively.

$$\mathfrak{S}^{com} = \frac{H}{\max_{i=1}^P e_i} \tag{30}$$

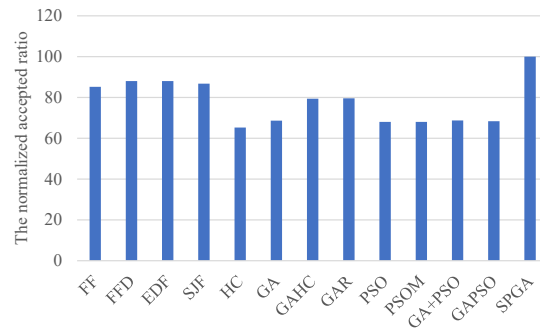
$$\mathfrak{S}^{data} = \frac{A}{\max_{i=1}^P e_i} \tag{31}$$

Our experiments repeated the following steps more than 100 times. First, we generated a simulated hybrid cloud system with the aforementioned parameters. Then, we measured each performance metric for every scheduling method in the generated hybrid cloud. Last, we normalized every metric value by dividing it by that of the SPGA, to highlight the relative performance of the measured methods. In the following results, we report the average normalized value for each metric and every method.

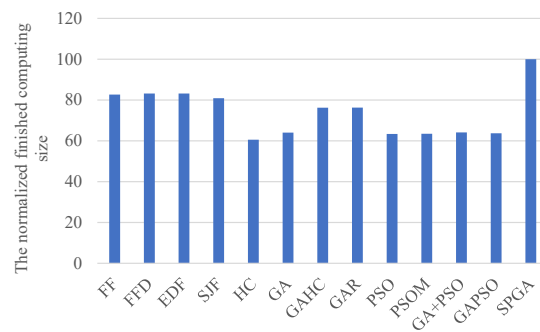
4.2. User Satisfaction

Figure 4 gives the performance achieved by various methods in three satisfaction metrics. From the figure, we can see that SPGA can accept 13.6–53.2% more tasks, finish 20.2–65.1% more computing, and process 13.9–53.4% more data than other methods, on average. This result

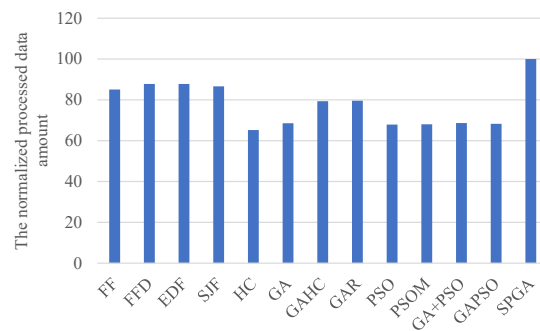
shows that our method has good performance in optimizing user satisfaction. There are mainly two advantages of our method. One is the re-scheduling of rejected tasks to improve the task assignment directly mapped from the code for each individual (line 4 in Algorithm 2), another is the exploitation of self and social cognition. The first improvement can be applied in any meta-heuristic-based method.



(a)



(b)



(c)

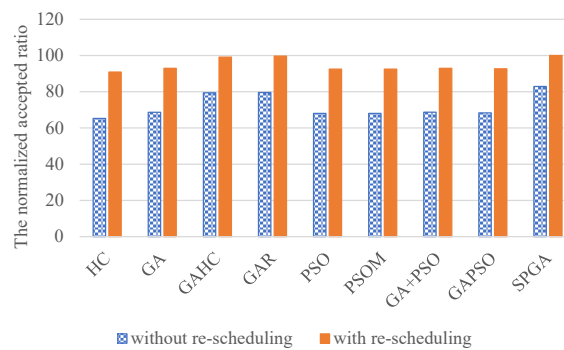
Figure 4. The relative values achieved by various task scheduling methods in satisfaction metrics. (a) Accept ratio; (b) finished computing size; (c) processed data amount.

Figure 5a shows the performance of the compared meta-heuristic-based methods and our method with and without the improvement of re-scheduling, in an accepted ratio. The re-scheduling approach can improve performance of these meta-heuristic-based methods by 20.6–39.2%, on average. Meanwhile, SPGA has a smaller improvement degree than other methods, by re-scheduling, which is a further proof of the excellence performance of SPGA.

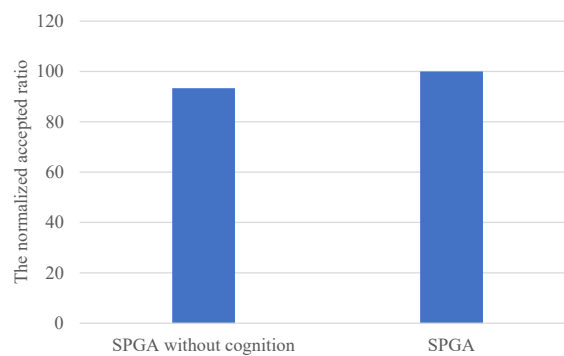
Figure 5b present the performance results of SPGA with and without self and social cognition in an accepted task number, where SPGA without cognition performs the crossover operator on an individual only with another individual in each evolutionary iteration. The results show that SPGA has 7.15% more accepted tasks than SPGA without cognition, on average. This illustrates that it is a wise decision to integrating the cognition of PSO into GA.

In addition, as shown in Figure 5a, SPGA has about 8% better performance than GA with and PSO with re-scheduling improvement, in accepted ratio, on average. This result validates the good integration solution of SPGA. Meanwhile, PSOM, GA+PSO, and GAPSO all achieve comparable performance to GA or PSO. This is mainly because their combination strategy is performs GA and PSO separately in each evolutionary iteration or the whole evolutionary process, without exploiting both advantages simultaneously.

We can also see from Figure 4 that heuristic-based methods (FF, FFD, EDF, and SJF) are better than meta-heuristic-based methods, except our method. In addition, SPGA has much better performance than PSOM, GA+PSO, and GAPSO, even though they all combine GA and PSO. This shows that we should carefully design the strategy to exploit meta-heuristic and hybrid heuristic-based algorithms for task scheduling. Otherwise, we may get task scheduling methods with poor performance and high overheads.



(a)



(b)

Figure 5. The improvement of the re-scheduling and cognition for SPGA. (a) Re-scheduling; (b) cognition.

4.3. Resource Efficiency

Figure 6 shows the normalized resource utilization when applying various methods. Our method has slightly higher utilization than FF, FFD, EDF, and SJF, and about 25% higher than others. This phenomenon verifies the good performance of our method in the overall resource efficiency for task scheduling in hybrid clouds. The immediate cause of these results is that SPGA finished about 20% more computing size but occupied about 18% more computing resources, on average, compared with these heuristics. Compared to other meta-heuristics, SPGA finishes 31.1–65.1% more computing with only 2.71–20.0% more

occupied resources. Therefore, SPGA can achieve a good satisfaction without sacrificing or even with improved resource efficiency.

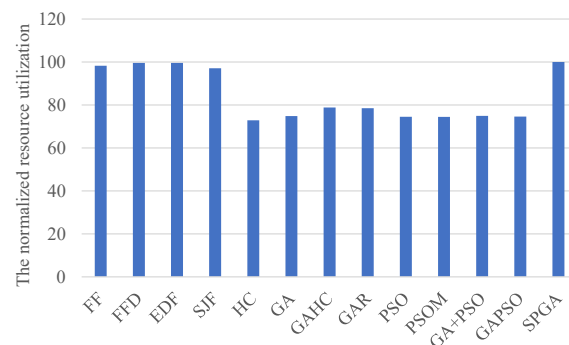


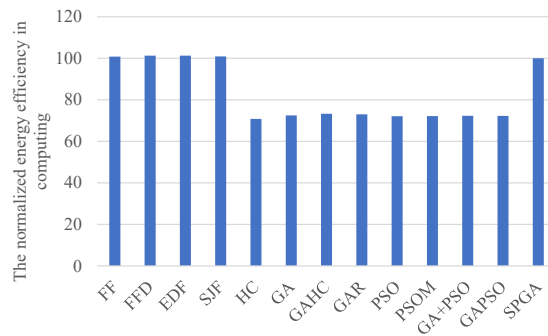
Figure 6. The relative resource utilizations achieved by different methods.

In fact, when applying these heuristic algorithms, the vast majority of accepted tasks are processed by the private cloud. This is because heuristic algorithms use the private resources in preference of lower operation costs of the private resources than the rent cost of public resources. This is how most existing heuristic-based algorithms work for task scheduling in hybrid clouds. This can cause the issue that there are very few private resources for tasks that cannot be finished by the public resources in later phases of scheduling, which is because in the early phase, all tasks are processed by the private resources first, including tasks with requirements that can be met by the public cloud. This issue can be addressed by meta-heuristic algorithms, as for each task, there is a fifty–fifty chance to schedule every task to the public cloud, which is implemented by our encoding method. Thus, there can be more private resources for tasks whose requirements cannot be satisfied by the public cloud, by scheduling some later tasks to the public clouds.

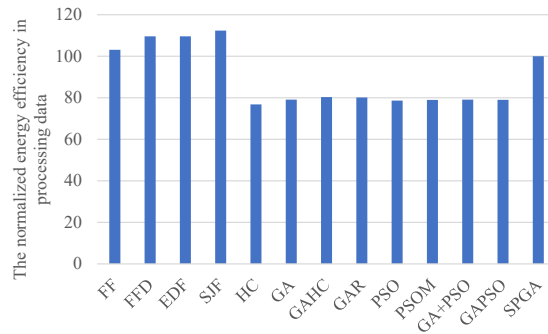
These above situations result in heuristic algorithms having comparable resource efficiencies in the private cloud, but much less in the public cloud, compared with SPGA. This result can be further confirmed by Figures 7 and 8, which respectively present the energy efficiency of the private cloud and the cost efficiency of the public cloud. From these figures, SPGA achieves comparable energy efficiency and much better cost efficiency than heuristic algorithms. On the contrary, SPGA outperforms other meta-heuristic-based algorithms in energy efficiency. This is mainly because other meta-heuristic-based algorithms do not employ the re-scheduling strategy for balancing the load among private computing cores, and thus have poor resource efficiencies.

Figure 7b also shows that heuristic-based algorithms have better energy efficiency in processing data than meta-heuristic-based methods. This is mainly because heuristic-based algorithms process more tasks with less input data in the private cloud. Our experiment results show that heuristic-based algorithms achieve about 12% more tasks than SPGA, but comparable amounts of processed data to SPGA. The reasons are as follows. The processing latencies of tasks with a small amount of input data are low, and others are high. In the first half phase of scheduling, heuristic-based algorithms assign some tasks with low latencies to the private cloud, even though they can be processed in the public cloud. This results in few available private resources for meeting the security or deadline constraints of tasks in the latter scheduling phase. Therefore, there are much less tasks scheduled to the public cloud when using heuristics, compared with meta-heuristics.

When there are less tasks processed by the public cloud, each VM instance has lower reusability for task processing. This can result in a worse performance of pipeline processing, leading to more idle CPU time when waiting for the input data. Thus, meta-heuristic algorithms have better cost efficiency than heuristic algorithms, as shown in Figure 8.

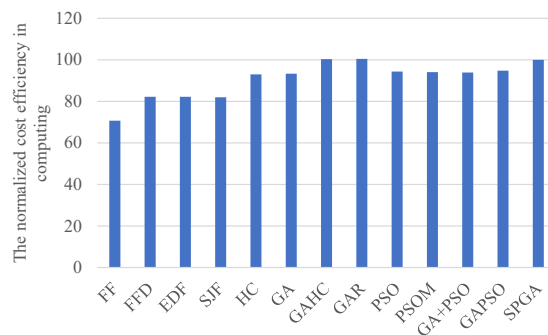


(a)

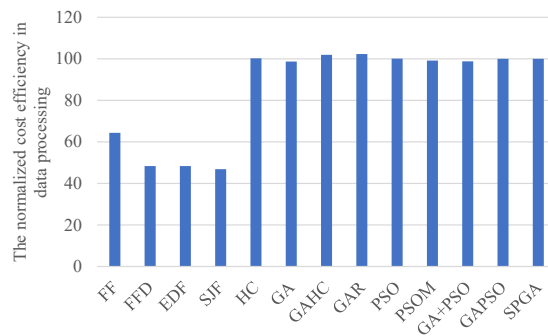


(b)

Figure 7. The energy efficiency of the local cloud when applying different methods. (a) In computing; (b) in data processing.



(a)

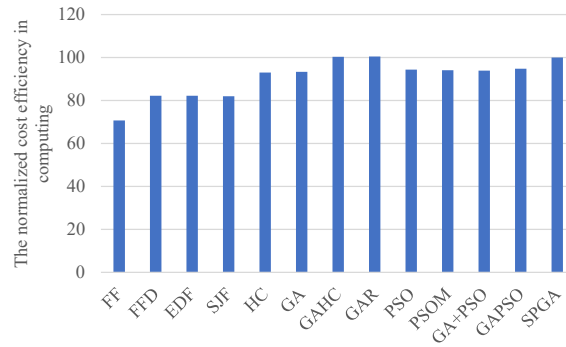


(b)

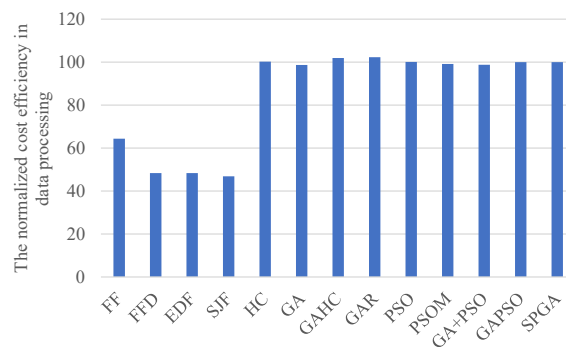
Figure 8. The cost efficiency of the public cloud when applying different methods. (a) In computing; (b) in data processing.

4.4. Processing Efficiency

Figure 9 gives the processing efficiency achieved by different scheduling algorithms, which shows that SPGA is the best. This is because SPGA finishes the most computing size and processes the largest amount of data, but it has a comparable timespan to others, as shown in Figure 10. Thus SPGA achieves the highest rates of computing and data processing overall. This further validates the high efficiency of our method.



(a)



(b)

Figure 9. The overall computing and data processing rates achieved by various methods. (a) Computing rate; (b) data processing rate.

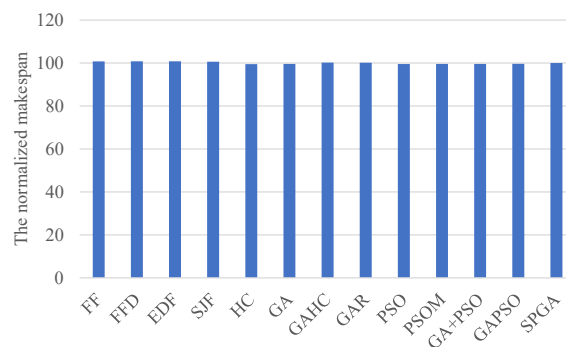


Figure 10. The relative makespans achieved by various methods.

5. Related Work

Hybrid clouds combine the benefits of both the private and public clouds, provide a cost-efficient way to address the temporal variation of request loads. There are several works studying task scheduling for hybrid clouds to improve the task execution performance and the resource efficiency.

Islam et al. [30] presented two heuristic algorithms for scheduling Spark jobs on hybrid cloud VMs, to optimize the resource cost and the deadline met percentage. The first one used first fit (FF), which iteratively placed a task (executor) in the first VM with all constraints. The second is using the idea of best fit (BF), which assigned each task to the VM with minimal increased costs. Min-CAMin and Max-CAMin were proposed by Stavrinides and Karatza [31], which enhance Min–Min and Max–Min by taking into account the resource cost. Min–CAMin/Max–CAMin performs the task of VM assignment with the minimum/maximum completion time in all assignments of tasks to VMs with minimal cost every time. The heuristic algorithms can provide an approximate optimal solution with a very small time overhead, but their performances are generally limited because they only use local search strategies. Thus, several works exploited the global search abilities of meta-heuristic algorithms to achieve better scheduling solutions.

Yin et al. [32] proposed an immune algorithm-based method with a task assignment strategy, for scheduling bag-of-tasks applications on hybrid clouds, to maximize the profit with a certain probability of deadline violations. Their proposed method used immune algorithms to decide the task scheduling order. The task assignment strategy used FF to schedule tasks on the private cloud, and the idea of BF to schedule tasks on public clouds for tasks not satisfied by the private cloud. Gandhi and Revathi [33] used an improved ant colony optimization for task scheduling in the private cloud, and sent tasks with execution times exceeding deadlines to the public cloud. Lin et al. [34] employed ant colony optimization (ACO) algorithm for container scheduling on the hybrid cloud, where all resources are provided in the form of VM. Hussain et al. [35] employed a quantum-inspired genetic algorithm for task scheduling in hybrid clouds, to optimize the makespan and cost. Alharbe and Rakrouki [36] presented a game theory-based method for scheduling VMs to physical machines on a hybrid cloud consisting of a group of clouds. In their method, players include users, the hybrid cloud provider, and the provider of every cloud. The objective of each user is minimizing the completion time, and each provider aims to maximize their profit. Rizvi et al. [37] applied salp swarm algorithm (SSA) and solved the stagnation problem by the fitness-based quasi-reflection method, to optimize the execution cost with a deadline constraint for a workflow in hybrid cloud-fog computing. To improve the total cost and total delay, Shahjalal et al. [38] exploited gray wolf optimization (GWO) for deploying virtual network functions (VNFs) on the hybrid cloud with several edge clouds and one central cloud. This work considered every cloud as a server, and did not consider the scheduling within a cloud. Abbes et al. [39] adopted a binary PSO (BPSO)-based method to decide the private or public cloud for each service's deployment. This work improved BPSO by increasing the probability that the value is updated to 1 in each dimension, i.e., a service is more likely deployed on the public cloud. This can improve the service quality but increase the resource cost. These above researchers applied only one meta-heuristic algorithm, and did not consider exploiting the complementary advantages of two or more algorithms for better performance.

PSO-SA [40] algorithm performed simulated annealing (SA) after each iteration of PSO to improve the global search ability of PSO. Yuan et al. [41,42] proposed GSPSO to improve the total cost for hybrid cloud computing, which performed GA, PSO, and SA, sequentially, in each iteration. Lei et al. [43] first used a list scheduling to assign each task to the resource with minimal cost increment, and then applied simulated annealing (SA) to improve the scheduling solution. This work ensured the security by encryption methods for the intermediate data transfer across the Internet. CR-PSO [44] used chemical reaction optimization (CRO) to create the initial population for PSO. Attiya et al. [45] presented a hybrid swarm intelligence method combining SSA and manta ray foraging optimization (MRFO). This method applied the update strategy of MRFO first, and then that of SSA, in every population evolution. Even though these works exploited two or more heuristic/meta-heuristic algorithms for improving the scheduling performance, they only applied these algorithms separately in each iteration or the whole process of the population evolution, which can result in a limited performance of combination.

Therefore, in this paper, we proposed an efficient integration solution to combine GA and PSO for task scheduling in the hybrid cloud, concerning the heterogeneity of hybrid resources, as well as the deadline and security constraints.

6. Conclusions

In this paper, we focus on the task scheduling problem with security and deadline constraints on hybrid clouds. To solve the problem, we first model it as an MINLP, and then propose a hybrid heuristic algorithm by combining both GA and PSO. To address the poor convergence performance of GA and the easy trapping into local optima of PSO, the proposed hybrid heuristic algorithm integrates the cognition exploited by PSO into GA. Extensive experimental results confirm the efficiency and effectiveness of our proposed hybrid heuristic algorithm.

In this paper, we consider the binary security model, where tasks with a security requirement must be processed in the private cloud. In some situations, tasks with less sensitivity can be serviced by the public cloud, with the help of data protection technologies. Data protection technologies have various overheads, and thus there is a tradeoff between the user satisfaction and these additional overheads, which we will study in the future.

Author Contributions: Conceptualization, Y.H. and B.W.; methodology, Y.H.; software, S.Z.; validation, Y.H., S.Z. and B.W.; formal analysis, S.Z.; investigation, Y.H.; resources, S.Z.; data curation, B.W.; writing—original draft preparation, Y.H.; writing—review and editing, B.W.; visualization, S.Z.; supervision, B.W.; project administration, B.W.; funding acquisition, 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), and the National Natural Science Foundation of China (Grant No. 61975187, 62072414).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

DFS	Distributed file system
EDF	Earliest deadline first
FF	First fit
FFD	First fit decreasing
GA	Genetic algorithm
GAHC	GA with hill climbing
GAR	GA with Replacement
HC	Hill climbing
MINLP	Mixed-integer non-linear programming
MIPS	Million instructions per second
PM	Physical machine
PSO	Particle swarm optimization
PSOM	PSO with mutation operator
QoS	Quality of service
SJF	Shortest job first
SME	Small and medium-size enterprise
SPGA	Security-aware hybrid PSO and GA algorithm
VM	Virtual machine

References

1. Wang, B.; Wang, C.; Song, Y.; Cao, J.; Cui, X.; Zhang, L. A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds. *Clust. Comput.* **2020**, *23*, 2809–2834. [[CrossRef](#)]
2. Muralidharan, C.; Anitha, R. Trusted cloud broker for estimating the reputation of cloud providers in federated cloud environment. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6537. [[CrossRef](#)]
3. Lu, C.; Chen, W.; Ye, K.; Xu, C.Z. Understanding the Workload Characteristics in Alibaba: A View from Directed Acyclic Graph Analysis. In Proceedings of the 2020 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS), Shenzhen, China, 23 May 2020; pp. 1–8. [[CrossRef](#)]
4. Boutaba, R.; da Fonseca, N.L.S. Cloud Architectures, Networks, Services, and Management. In *Cloud Services, Networking, and Management*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2015; Chapter 1, pp. 1–22. [[CrossRef](#)]
5. Awotunde, J.B.; Bhoi, A.K.; Barsocchi, P. *Hybrid Cloud/Fog Environment for Healthcare: An Exploratory Study, Opportunities, Challenges, and Future Prospects*; Springer: Singapore, 2021; pp. 1–20. [[CrossRef](#)]
6. Jebbar, Y.; Promwongsa, N.; Belqasmi, F.; Glitho, R.H. A Case Study on the Deployment of a Tactile Internet Application in a Hybrid Cloud, Edge, and Mobile Ad Hoc Cloud Environment. *IEEE Syst. J.* **2022**, *16*, 1182–1193. [[CrossRef](#)]
7. Chamara, N.; Islam, M.D.; Bai, G.F.; Shi, Y.; Ge, Y. Ag-IoT for crop and environment monitoring: Past, present, and future. *Agric. Syst.* **2022**, *203*, 103497. [[CrossRef](#)]
8. Sheng, H.; Wei, S.; Yu, X.; Tang, L. Research on robot grabbing system based on hybrid cloud. *J. China Univ. Posts Telecommun.* **2021**, *28*, 48–54. [[CrossRef](#)]
9. Ajmal, M.S.; Iqbal, Z.; Khan, F.Z.; Ahmad, M.; Ahmad, I.; Gupta, B.B. Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers. *Comput. Electr. Eng.* **2021**, *95*, 107419. [[CrossRef](#)]
10. Du, J.; Leung, J.Y.T. Complexity of Scheduling Parallel Task Systems. *SIAM J. Discret. Math.* **1989**, *2*, 473–487. [[CrossRef](#)]
11. Wolpert, D.; Macready, W. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [[CrossRef](#)]
12. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [[CrossRef](#)]
13. Tsao, Y.C.; Delicia, M.; Vu, T.L. Marker planning problem in the apparel industry: Hybrid PSO-based heuristics. *Appl. Soft Comput.* **2022**, *123*, 108928. [[CrossRef](#)]
14. Bal, P.K.; Mohapatra, S.K.; Das, T.K.; Srinivasan, K.; Hu, Y.C. A Joint Resource Allocation, Security with Efficient Task Scheduling in Cloud Computing Using Hybrid Machine Learning Techniques. *Sensors* **2022**, *22*, 1242. [[CrossRef](#)] [[PubMed](#)]
15. Li, W.; Fan, Q.; Dang, F.; Jiang, Y.; Wang, H.; Li, S.; Zhang, X. Multi-Objective Optimization of a Task-Scheduling Algorithm for a Secure Cloud. *Information* **2022**, *13*, 92. [[CrossRef](#)]
16. Min, J.; Liu, M.; Chugh, T.; Zhao, C.; Wei, A.; Doh, I.H.; Krishnamurthy, A. Gimbal: Enabling Multi-Tenant Storage Disaggregation on SmartNIC JBOFs. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference, Virtual Event, 23–27 August 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 106–122. [[CrossRef](#)]
17. Vinod, C.S.S.; Anand, H.S. Nature inspired meta heuristic algorithms for optimization problems. *Computing* **2022**, *104*, 251–269. [[CrossRef](#)]
18. 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](#)]
19. 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](#)]
20. Hussain, A.A.; Al-Turjman, F. Hybrid Genetic Algorithm for IOMT-Cloud Task Scheduling. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 6604286. [[CrossRef](#)]
21. 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](#)]
22. Aghdashi, A.; Mirtaheeri, S.L. Novel dynamic load balancing algorithm for cloud-based big data analytics. *J. Supercomput.* **2022**, *78*, 4131–4156. [[CrossRef](#)]
23. 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 GECCO'22 Genetic and Evolutionary Computation Conference, Boston, MA, USA, 9–13 July 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 177–185. [[CrossRef](#)]
24. 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](#)]
25. 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](#)]
26. Teraiya, J.; Shah, A. Optimized scheduling algorithm for soft Real-Time System using particle swarm optimization technique. *Evol. Intell.* **2022**, *15*, 1935–1945. [[CrossRef](#)]
27. 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](#)]
28. 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](#)]

29. 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](#)]
30. Islam, M.T.; Wu, H.; Karunasekera, S.; Buyya, R. SLA-Based Scheduling of Spark Jobs in Hybrid Cloud Computing Environments. *IEEE Trans. Comput.* **2022**, *71*, 1117–1132. [[CrossRef](#)]
31. Stavrinides, G.L.; Karatza, H.D. Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud. *Multimed. Tools Appl.* **2021**, *80*, 16781–16803. [[CrossRef](#)]
32. Yin, L.; Zhou, J.; Sun, J. A stochastic algorithm for scheduling bag-of-tasks applications on hybrid clouds under task duration variations. *J. Syst. Softw.* **2022**, *184*, 111123. [[CrossRef](#)]
33. Gandhi, S.Y.; Revathi, T. An improved hybrid cloud workflow scheduling algorithm based on ant colony optimization. *Int. J. Health Sci.* **2022**, *6*, 869–882. [[CrossRef](#)]
34. Ant Colony Algorithm for Container-based Microservice Scheduling in Hybrid Cloud. In Proceedings of the 2021 International Conference on Big Data and Intelligent Algorithms (BDIA 2021), Chongqing, China, 9–11 July 2021. [[CrossRef](#)]
35. Hussain, M.; Wei, L.F.; Abbas, F.; Rehman, A.; Ali, M.; Lakhan, A. A multi-objective quantum-inspired genetic algorithm for workflow healthcare application scheduling with hard and soft deadline constraints in hybrid clouds. *Appl. Soft Comput.* **2022**, *128*, 109440. [[CrossRef](#)]
36. Alharbe, N.; Rakrouki, M.A. A Game Theory-based Virtual Machine Placement Algorithm in Hybrid Cloud Environment. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 619–629. [[CrossRef](#)]
37. Rizvi, N.; Ramesh, D.; Rao, P.C.S.; Mondal, K. Intelligent Salp Swarm Scheduler With Fitness Based Quasi-Reflection Method for Scientific Workflows in Hybrid Cloud-Fog Environment. *IEEE Trans. Autom. Sci. Eng.* **2023**, *20*, 862.
38. Shahjalal, M.; Farhana, N.; Roy, P.; Razzaque, M.A.; Kaur, K.; Hassan, M.M. A Binary Gray Wolf Optimization algorithm for deployment of Virtual Network Functions in 5G hybrid cloud. *Comput. Commun.* **2022**, *193*, 63–74. [[CrossRef](#)]
39. Abbas, W.; Kechaou, Z.; Hussain, A.; Qahtani, A.M.; Almutiry, O.; Dhahri, H.; Alimi, A.M. An Enhanced Binary Particle Swarm Optimization (E-BPSO) algorithm for service placement in hybrid cloud platforms. *Neural Comput. Appl.* **2023**, *35*, 1343–1361. [[CrossRef](#)]
40. Research on Task Scheduling Based on Particle Swarm Optimization Simulated Annealing Algorithm in Hybrid Cloud Environment. In Proceedings of the 2021 6th International Conference on Intelligent Information Processing (ICIIP 2021), Bucharest, Romania, 29–31 July 2021. [[CrossRef](#)]
41. Cost-minimized User Association and Partial Offloading for Dependent Tasks in Hybrid Cloud-edge Systems. In Proceedings of the 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, 20–24 August 2022. [[CrossRef](#)]
42. Yuan, H.; Bi, J.; Zhou, M. Temporal Task Scheduling of Multiple Delay-Constrained Applications in Green Hybrid Cloud. *IEEE Trans. Serv. Comput.* **2021**, *14*, 1558–1570. [[CrossRef](#)]
43. Lei, J.; Wu, Q.; Xu, J. Privacy and security-aware workflow scheduling in a hybrid cloud. *Future Gener. Comput. Syst.* **2022**, *131*, 269–278. [[CrossRef](#)]
44. Dubey, K.; Sharma, S. A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing. *Sustain. Comput. Inform. Syst.* **2021**, *32*, 100605. [[CrossRef](#)]
45. Attiya, I.; Elaziz, M.A.; Abualigah, L.; Nguyen, T.N.; El-Latif, A.A.A. An Improved Hybrid Swarm Intelligence for Scheduling IoT Application Tasks in the Cloud. *IEEE Trans. Ind. Inform.* **2022**, *18*, 6264–6272. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.