Neural Computing and Applications
https://doi.org/10.1007/s00521-019-04119-7

DEEP LEARNING FOR BIG DATA ANALYTICS

An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments

Zhou Zhou[1,2] · Fangmin Li[1] · Huaxi Zhu[3] · Houliang Xie[3] · Jemal H. Abawajy[4] · Morshed U. Chowdhury[4]

Received: 28 November 2018 / Accepted: 22 February 2019

Abstract
Cloud computing is an emerging distributed system that provides flexible and dynamically scalable computing resources for use at low cost. Task scheduling in cloud computing environment is one of the main problems that need to be addressed in order to improve system performance and increase cloud consumer satisfaction. Although there are many task scheduling algorithms, existing approaches mainly focus on minimizing the total completion time while ignoring workload balancing. Moreover, managing the quality of service (QoS) of the existing approaches still needs to be improved. In this paper, we propose a novel algorithm named MGGS (modified genetic algorithm (GA) combined with greedy strategy). The proposed algorithm leverages the modified GA algorithm combined with greedy strategy to optimize task scheduling process. Different from existing algorithms, MGGS can find an optimal solution using fewer number of iterations. To evaluate the performance of MGGS, we compared the performance of the proposed algorithm with several existing algorithms based on the total completion time, average response time, and QoS parameters. The results obtained from the experiments show that MGGS performs well as compared to other task scheduling algorithms.

Keywords Cloud computing · Genetic algorithm · Greedy strategy · Task scheduling optimization

1 Introduction

Cloud computing has emerged as a new computing paradigm that provides virtualized computing resources to Cloud service consumers as a service [1, 2]. Cloud services are dynamically provisioned to Cloud service consumers based on Service-Level Agreements (SLA). Normally, SLA is established through negotiation between the Cloud service providers and Cloud service consumers. Cloud services can be automatically scaled up or down and provisioned to Cloud service consumers' on-demand based on pay-as-you-go payment model [3, 4]. From the perspective of the specific application, the service provided by cloud computing can be classified into three types, that is, the infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [5]. Since Cloud service consumers pay for using the Cloud services, they generally expect Quality of Service (QoS) guaranteed.

With increasing adoption of cloud computing services particularly infrastructure as a service (IaaS) by organizations, there are a significant number computing tasks that have been executing in cloud computing environment. A variety of applications such as scientific applications, content delivery, database, social networking, CRM, and web hosting are employed by organizations within a cloud computing service provider's infrastructure. These applications tasks some instant small or large and the applications related computing jobs competing for Cloud resources which poses a serious challenge. The main challenge is how cloud services can be reasonably provisioned with the objective of achieving maximum utilization of resources without compromising high QoS guarantees. This is a key issue that determines the efficiency of cloud

Zhou Zhou and Huaxi Zhu are co-first authors.

✉ Houliang Xie
gcc@smail.hunnu.edu.cn

1 Department of Mathematics and Computer Science, Changsha University, Changsha 410083, China

2 Department of Computer Science, Hunan University, Changsha 410083, China

3 Information Engineering Department, Zhangjiajie Institute of Aeronautical Engineering, Zhangjiajie 427000, China

4 School of Information Technology, Deakin University, Geelong, VIC 3217, Australia

Published online: 08 March 2019

computing as well as the common concerns for both cloud service providers and cloud service consumers [6, 7]. This issue generally addressed by using efficient task scheduling algorithm optimizes both cloud service providers and cloud service consumers need.

In the cloud computing infrastructure services environment, task scheduling refers to the process of efficient scheduling of computing tasks, and reasonable allocation of computing resources under certain constraints. This process can be considered as a mapping between computing tasks and cloud computing resources, which is based on the premise of satisfying certain optimization goals. Combining the physical characteristics of computing resources (e.g., CPU, memory, disk, bandwidth) and the task execution features (e.g., execution time and costs) requires a suitable scheme to complete task scheduling on time and at the negotiated reasonable cost. Therefore, choosing a suitable task scheduling algorithm to improve the utilization rate of cloud computing resources while maintaining high QoS guarantees is an important problem that continues to receive attention in research. Genetic algorithm (GA) [8, 9], as a random search method that simulates biological evolution process, has been widely used in cloud computing resource scheduling. The genetic algorithm is used due to its efficient, parallel, and global search characteristics. However, the traditional genetic algorithm has some disadvantages, such as local convergence and slow convergence speed.

In this paper, we proposed an improved GA for cloud resource scheduling, through the proper encoding, selection of task completion time, crossover and mutation operation, and enhancing the efficiency. The proposed algorithm reduces the overall execution time, average response time and achieves load balancing over the Cloud resources with minimum financial cost. Our proposed algorithm named as a modified GA algorithm combined with greedy strategy (MGGS) to handle the problem of task scheduling.

The novelty of the paper is that it leverages the modified GA combined with greedy strategy to deal with the task scheduling problem. Different from existing algorithm, MGGS can find an optimal solution using fewer number of iterations. Specifically,

1. The MGGS not only considers the overall execution time, average response time, but also takes into account the QoS and the load balance over the resources with minimum monetary cost.
2. We are presenting the formalize definition of QoS for the task scheduling algorithm. Different from the other definitions, it can be directly used to measure the performance of task scheduling algorithms.
3. The proposed MGGS optimized the task scheduling, thus will be finding an optimal solution at the less number of iterations.
4. The efficiency of the proposed algorithm is evaluated through the extensive experimental analysis to prove its effectiveness in solving the scheduling problem in the cloud environments.

Due to the randomness of jobs, we did not consider stochastic service system theory in MGGS at this stage. This may be noted as one of the drawbacks of our proposed algorithm. However, we will introduce stochastic service system theory in our future work to improve cloud computing for resource management.

The rest of the paper is organized as follows: Sect. 2 presents the related work. The proposed MGGS algorithm is proposed in Sect. 3. The performance analysis is presented in Sect. 4. Section 5 concludes the paper.

2 Related work

Many algorithms to deal with the problem of task scheduling in cloud computing have been proposed with the aim to minimize the overall execution time, average response time, and achieve the load balance. These task scheduling algorithms can be divided into three classes.

Task scheduling algorithm of the first kind is based on genetic algorithm (GA) [10–12]. The main idea of GA concerns about the usage of the coding and decoding technology, definition of the fitness function, the using of the selection, reproduction, crossover, and mutation approaches, for the purposing of improving the efficiency of resources scheduling and achieving the load balance over the resources with minimum total monetary cost. Specifically, in [10], Pizzuti et al. put forward a multi-objective GA algorithm to discover community structure in a complex network. The proposed algorithm in the paper mainly optimizes two objective functions that maximizes the intra-connections inside each community and minimizes inter-connections between different communities. To efficiently allocate tasks in cloud computing, Manasrah et al. [11] proposed a hybrid GA-PSO algorithm and the main idea is to reduce the makespan and cost and balance the load of the dependent tasks over the heterogeneous resources in cloud computing environments. Sathappan et al. [12] presented a modified GA for multi-objective task scheduling on a heterogeneous computing system, and the experimental results show that the proposed algorithm is better than other scheduling algorithms.

The second kind of task scheduling algorithm is based on the greedy strategy [13–16]. When solving a problem, it always makes what seems to be the best choice at the

moment. In other words, instead of finding the global optimum, what it does is in some sense the local optimal solution. Greedy algorithm is not the overall optimal solution to all problems. The key is the selection of greedy strategy. For example, Etminani et al. [13] proposed a new task scheduling algorithm named Min–Min to optimize the task scheduling. Min–Min algorithm prefers assigning small tasks to fast resources to execute so that the total completion time is minimum. However, Min–Min can cause the slow resource with light workload while the fast resource with heavy workload. That is to say, Min–Min can lead to the low utilization rate in a cloud computing environment. Different from Min–Min, Moreno et al. [14] put forward job scheduling and resource management techniques named Max–Min. Max–Min is fond of scheduling big tasks. Therefore, Max–Min can also cause the resource in cloud computing environments with low utilization rate. To improve the resource utilization rate, Zhou et al. [15] presented a modified algorithm called Min–Max based on the combination of both Min–Min and Max–Min algorithm. Also, in [16], Mao et al. leveraged Max–Min algorithm to execute the large tasks first.

The other algorithms concerning to task scheduling include the particle swarm optimization (PSO) [17–19], ant colony optimization (ACO) [20–22], simulated annealing algorithm (SAA) [23], round-robin scheduling algorithm such as FCFS (first come first service) [24], and so on. Specifically, Almaamari et al. [18] proposed a dynamic adaptive particle swarm optimization algorithm (DAPSO) to improve the performance of the PSO algorithm, for the purpose of optimizing the task runtime by minimizing the makespan of a particular task set, thus maximizing resource utilization. In [22], Dai et al. integrate ACO with GA to deal with the task scheduling under the consideration of time-consuming, expenditure, security, and reliability (four-dimensional QoS objectives), and the experiment illustrates that the proposed algorithm has a better performance both in balancing resources and guaranteeing QoS.

3 MGGS algorithm

In this part, we will introduce the goal of task scheduling algorithm, main idea of MGGS algorithm (concerning about the coding, fitness function, crossover, mutation operator, greedy selection), flowchart of MGGS, and the specific implementation of MGGS.

3.1 Task scheduling algorithm goal

Different task scheduling algorithms can be applied into different fields and yield different effects. Therefore, there are many indicators that can be used to evaluate the task scheduling algorithms. However, as an excellent task scheduling algorithm, it should meet some following basic goals:

1. Minimize the makespan: It refers to the algorithm completes the task scheduling with the minimum overall execution time.
2. Minimize the average response time: The average response time is also an important indicator to evaluate the task scheduling algorithm. As an excellent task scheduling algorithm, the average response time of an excellent algorithm should be less.
3. Balance the workload: Resource utilization rate is closely related to the balance of the workload. As an excellent algorithm, most of the (computing) resources in cloud environments should be fully utilized.
4. Maximize the QoS: The QoS factor plays an important role and can be used to evaluate the task scheduling algorithm. In general, the higher the QoS, the better when other indicators remain unchanged.
5. Minimize the total monetary cost: In general, under the meeting of user QoS, the less the total monetary cost, the better.

3.2 Main idea of MGGS algorithm

3.2.1 Coding

Coding problem is the first problem that GA needs to solve. Generally, there are two commonly used coding such as decimal code and binary code. The advantage of decimal code is that it is easy to understand and not to be decoded. Compared with decimal code, binary code has a good characteristic of stability and with a large population of diversity. In this paper, we use binary code for MGGS algorithm. For example, supposing there are three tasks in the cloud environments. Serial number "011110" shows that the "task 1" will be assigned to virtual machine no. 1 (binary code "01") to execute, and "task 2" will be assigned to virtual machine no. 3 (binary code "11") to execute, and "task 3" will be assigned to virtual machine no. 2 (binary code "10") to execute.

In this paper, the number of tasks is the length of the chromosome. The computing resource number occupied by the task is expressed by the value of each gene in the chromosome, as shown in Fig. 1. Figure 1 shows the chromosome coding of task scheduling. The main problem of task scheduling algorithms should deal with is to schedule N tasks (denoted by $T_1, T_2, T_3, \ldots, T_N$) to M (denoted by $R_1, R_2, R_3, \ldots, R_M$) ($N > M$) computing resources. Figure 1 illustrates the example of task scheduling algorithm, and the length of chromosome is N (also called the number of tasks). The decoding of chromosome

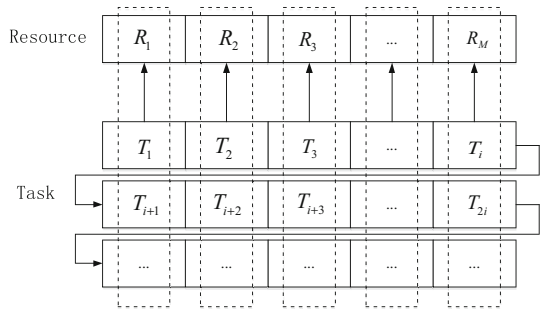

Fig. 1 Chromosome coding of task scheduling

corresponds to the dotted box in Fig. 1, that is: R_1: $\{T_1, T_{i+1}, \ldots\}$ R_2: $\{T_2, T_{i+2}, \ldots\}$ … R_M: $\{T_i, T_{2i}, \ldots\}$.

3.2.2 Fitness function

To deal with the efficiency problem of task scheduling algorithm, the fitness of each code indicates the efficiency of the code. The higher the fitness value, the lower the efficiency of the task scheduling corresponding to the coding; otherwise, the higher the efficiency. The fitness values for each individual are expressed as follows:

$$f(c_1, c_2 \ldots, c_N) = \frac{1}{\max(d(c_1), d(c_2) \ldots, d(c_N))} \tag{1}$$

where parameter c_i is the virtual machine (VM) that processes the task i, $d(c_i)$ refers to the expected total time for the VM c_i to execute all assigned tasks.

3.2.3 Selection

To reflect the characteristic of the parents, the good individual of the parents should be embodied in the next generation of population, while preserving the possibility of the poor fitness value of individual which can be chosen. In the paper, roulette algorithm is used for selection. The first step is to calculate the probability of each individual that is selected, defined as follows:

$$P(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)} \tag{2}$$

where $P(i)$ is the chosen probability of individual. Parameter $f(i)$ refers to the fitness value of individual, while $\sum_{i=1}^{N} f(i)$ represents the fitness value of the population. Figure 2 shows an example of roulette algorithm:

When the roulette stops rotating, the pointer points to an individual, and the pointed individual is selected. Therefore, individuals with higher fitness values are more likely to be selected than those with lower fitness values. Figure 2 is an example of roulette algorithm. In Fig. 2, there are five individuals and each individual owns different fitness

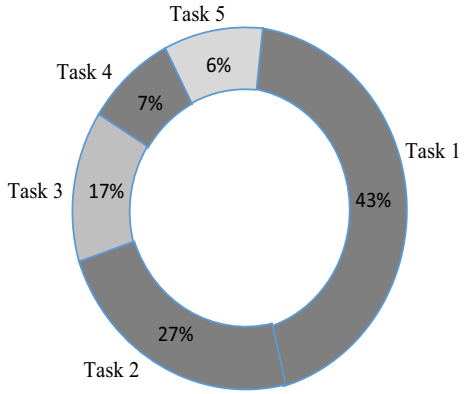

Fig. 2 An example of roulette algorithm

values. The individual with high fitness value will be chosen with high probability.

3.2.4 Crossover

The crossover operator is an important step for GA algorithm. To improve the global search capability of GA algorithm, it is necessary to do the crossover operator for two individuals. We use parameter P-crossover as the crossover probability, and it can be defined as follows:

$$P\text{ - crossover} = k_1 \frac{f_{\max} - f_a}{f_{\max} - f_b} \quad \text{if } f_a \geq f_b \tag{3}$$

$$P\text{ - crossover} = k_2 \quad \text{if } f_a < f_b \tag{4}$$

where parameter $f_{\max}$ is the maximum fitness value in the population, parameter f_a refers to the larger fitness value between the two individuals that need to do crossover operator, parameter f_b is the average fitness value in the entire population. Parameters k_1 and k_2 can be considered as a common value ($0 < k_1 < 1$, $0 < k_2 < 1$). Figure 3 shows the crossover operator.

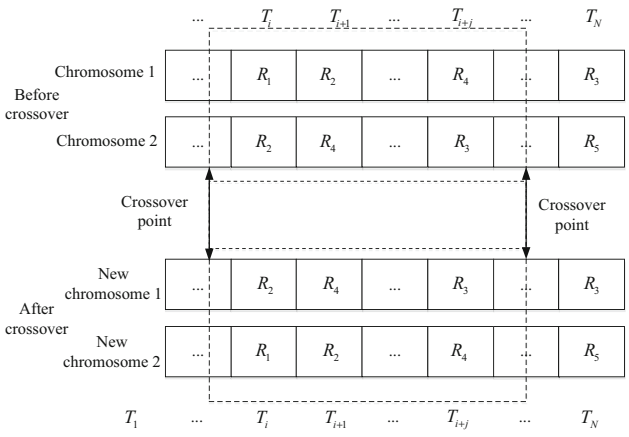

Fig. 3 An example of chromosome crossover operator

3.2.5 Mutate operator

Mutate operator is the same important as crossover. Mutation can improve the local search capability of GA algorithm and maintain the diversity of the population. The mutation probability can be calculated, as shown in Eqs. (5) and (6).

$$P\text{-mutate} = k_3 \frac{f_{\max} - f_c}{f_{\max} - f_b} \quad \text{if } f_c \geq f_b \tag{5}$$

$$P\text{-mutate} = k_4 \quad \text{if } f_c < f_b \tag{6}$$

where parameter P - mutate represents the mutation probability, Parameters k_3 and k_4 can be considered as a common value a ($0 < k_3 < 1$, $0 < k_4 < 1$). Parameter $f_{\max}$ is the maximum fitness value in the population, parameter f_c refers to the fitness value of the individual that will do the mutation operator, and parameter f_b is the average fitness value in the entire population. Figure 4 shows the crossover operator:

As shown in Fig. 4, there are two chromosome genes variation at the location $(i + 1)$ and $(N - 2)$, separately. At the location $(i + 1)$, the value of gene is from the R_2 to R_4. Similarly, the value of gene is from the R_5 to R_2 at the location $(N - 2)$. The mutation of gene may bring the good results. Sometimes, it may bring bad results. Nevertheless, mutation of gene is necessary to maintain the diversity of population.

3.2.6 Greedy selection

The greedy selection in this study occurred in the children coding after each mutation and crossover operator. These tasks assigned to a VM with the maximum total execution time will transfer to another VM with the minimum total execution time, for the purpose of reducing the total execution time, thus improving the efficiency of the task scheduling algorithm. Transferring some tasks to another VM will contribute to the balance of workload. The specific greedy selection is as follows:

1. Translating the coding into a task list for each VM. The estimated total execution time of each VM is calculated;
2. Sorting all VMs in descending order according to the total execution time;
3. Eliminating the minimum tasks in the VM with the maximum execution time, and sorting all VMs in descending order according to the total execution time;
4. Determining whether the VM with the minimum execution time is the VM with the maximum execution time. If so, terminate the algorithm and quit the greedy selection. If not, add the eliminated task into the VM with the minimum execution time;
5. Executing the step from (1) to (4).

The final obtained VM list is the improved coding after greedy selection. Replace the original children with the improved coding.

Using the greedy algorithm can improve the problem of children with low superior probability, and it also contributes to deal with the problem of load balancing.

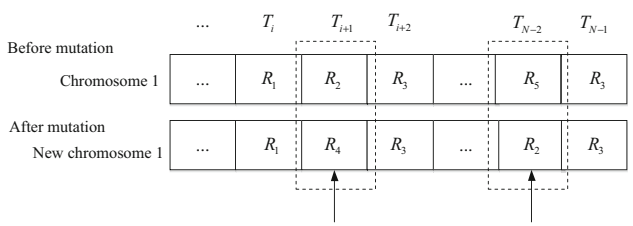

Fig. 4 An example of chromosome gene variation

3.3 The flowchart of MGGS

MGGS algorithm includes the coding, fitness function, selection, crossover, mutation operator, greedy selection. The flowchart of MGGS is as follows:

Figure 5 shows the procedure of the MGGS algorithm, and it includes the following steps:

1. Randomly generate several codes to form a population;
2. Using binary code to coding all tasks;
3. Calculate the fitness value of all coding in the population;
4. Select several pairs of codes to do the crossover operation to form new population (the total number of population remains unchanged);
5. Perform mutation operation through selection operator again;
6. Make greedy choices for all codes of the newly formed population. Form a whole generation of new population;
7. Repeat (2)–(6) until a stable optimal individual fitness is found, that is, the optimal solution of task scheduling.

3.4 The specific implementation of MGGS

For the specific implementation of MGGS algorithm, it mainly includes two classes, that is, the "Gene" class and "GenticAlgorithmWithGreedy" class. The "Gene" class is shown in Fig. 6.

"Gene" class is the basis of all classes, and each "Gene" class is used to save an individual. That is to say, each "Gene" class saves the map from task to VM. During

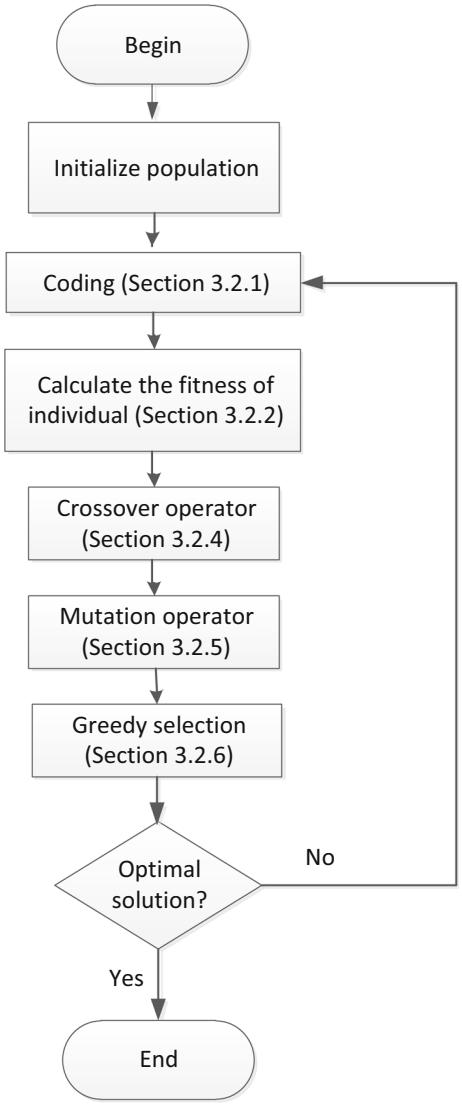

Fig. 5 An example of chromosome gene variation

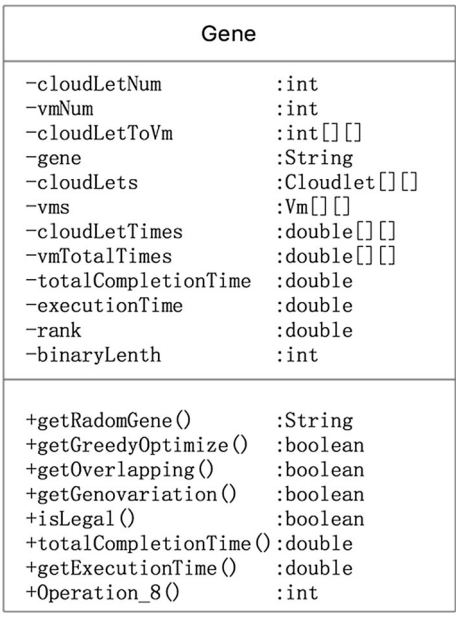

Fig. 6 Gene class

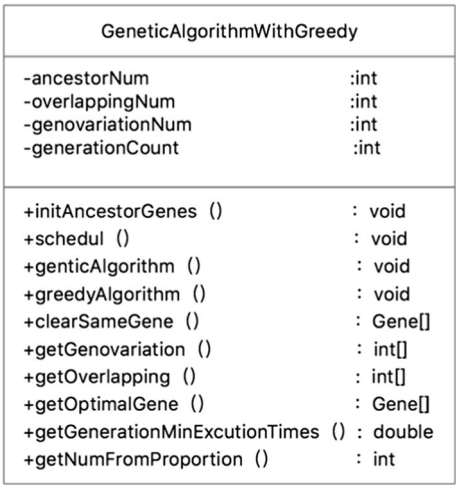

Fig. 7 GenticAlgorithmWithGreedy class

the "Gene" class, it includes variable such as "cloudLetNum," "vmNum," and function "getRadomGene ()."

Except for "Gene" class, there is another important class named "GenticAlgorithmWithGreedy." Figure 7 illustrates the basic structure of "GenticAlgorithmWithGreedy" class.

Figure 7 displays that the "GenticAlgorithmWithGreedy" class includes three important methods, that is, the greedyAlgorithm(), genticAlgorithm(), and schedul().

In the following three paragraphs, we will introduce the three important functions including greedyAlgorithm(), genticAlgorithm(), and schedul().

The first important function in "GenticAlgorithmWithGreedy" class is "greedyAlgorithm()." Its main function is to make a greedy operator for population. The pseudocode of "greedyAlgorithm" method is shown in Fig. 8.

The second important method of "GenticAlgorithmWithGreedy" class is the "genticAlgorithm()" method, and its main function is to make crossover and mutation operator for the population. The pseudocode of "genticAlgorithm" method is shown in Fig. 9.

The third important method of "GenticAlgorithmWithGreedy" class is the "schedul()" method, and its main function is to achieve the optimal task scheduling, and call "greedyAlgorithm" method and "genticAlgorithm" method. The pseudocode of "schedul" method is as follows (Fig. 10).

Algorithm 1: the pseudocode of the "greedyAlgorithm" method

Input: ethnicGroups, vms

Output: maxTime

```
for item in ethnicGroups     //ethnicGroups is the array of population
    is_loop = true;
    while is_loop
        is_loop=false;
        //get the VM with the estimated maximum execute time
        hard_vm = getMaxExecutionTimeVM(item);
        //obtain the total execution time of the VM
        maxTime = hard_vm.getTime();
        flag = false;
        for task in vm.listTasks
                for vm in vms
                //task.length is the length of a task，and vm.mips is
                // execute speed of the VM
                if maxTime > (task.length/vm.mips+vm.getTime())
                        maxTime = (task.length/vm.mips+vm.getTime());
                        flag = true;
                        is_loop = true;
                if flag
                        break;
```

Fig. 8 The pseudocode of greedyAlgorithm method

4 Performance analysis

In this section, we will evaluate the performance of MGGS, and the comparison algorithms include the GA [10], Min–Min [13], and FCFS [24].

4.1 The environment setup

To evaluate the performance of the proposed algorithm MGGS, we choose CloudSim toolkit [25, 26] as the simulation tool due to allowing the modeling the virtualized environments and supporting on-demand resource provisioning. Table 1 displays the task parameter list, and the tasks are generated randomly. The size of these tasks is as follows.

In this experiment, there are 10 VMs (can be considered as computing resource) that have been created to process the tasks to meet the requirement of users. The related parameter of these VMs is as follows.

Regarding the GA algorithm and MGGS algorithm, the number of population can affect the accuracy of the testing. Therefore, the number of population is set through empirical method. (In this paper, the number of population

Algorithm 2: the pseudocode of "genticAlgorithm" method

Input: ethnicGroups, overlapNum

Output: chromosome, mutation

```
//rank is the value of total execution time for each population,
// the more the total execution time, the more the rank value
ranks[];
// the proportion of the "rank" of each population
i=0;
totalRank=getTotalRank();
for item in ethnicGroups
        rank[i]=ItemEvent.rank/totalRank;
        i++;
i=0;
//overlapNum is the preset crossover number
while i < overlapNum
// randomly obtain the task with crossover according to
// the value of ranks, ethnicGroups is the array of population
        father, mother=getParent(ethnicGroups,ranks);
       // randomly obtain the location of the crossover
        start, end=getRandomPostions();
        //obtain the chromosome with the exchange
        f_chromosome=father.getChromosome(start,end);
        m_chromosome=mother.getChromosome(start,end);
        //do the crossover operator
        father.replace(start,end,m_chromosome);
        mother.replace(start,end,f_chromosome);
        i++;
i=0;
//mutationNum is the preset number of mutation
while i<mutationNum
//randomly obtain the task with mutation according to
//the value of ranks, ethnicGroups is the array of population
        mutation=getMutation(ethnicGroups,ranks);
      //randomly obtain the location of the mutation
        postion=getrandomPostion();
        chromosome=mutation.getChromosome(postion,postion+1);
        mutation.replace(postion,postion+1,!chromosome);
        i++;
```

Fig. 9 The pseudocode of genticAlgorithm method

Algorithm 3: the pseudocode of "schedul" method

Input: generationCount, ancestorGenes, generationGene

Output: optimalGene, newGenes

```
int count = 0;
//generationCount the number of iterations
while(count < generationCount) {
    count += 1;
    //ancestorGenes is the array of Gene,
    //getExecutionTime function is to obtain the execute time
   double minExecutionTime = ancestorGenes[0].
                              getExecutionTime();
      for (int i = 0; i < ancestorGenes.length; i++) {
          double executionTime =
                ancestorGenes[i].getExecutionTime();
          if (minExecutionTime > executionTime) {
              minExecutionTime = executionTime;
              generationGene = ancestorGenes[i];
                  }
              }
  //optimalGene is the population with the least execute
 //time from the original generation to this generation
      if (generationGene.getExecutionTime() <
          optimalGene.getExecutionTime()){
          optimalGene = generationGene;
           }
      //clear the same population
      Gene[] newGenes = clearSameGene(ancestorGenes);
      // do an genetic operator
      genticAlgorithm(ancestorGenes);
      // do a greedy selection operator
      greedyAlgorithm(ancestorGenes);
}
```

Fig. 10 The pseudocode of "schedul" method

is set as 50.) To ensure the accuracy of the experiments, we have repeated the experiment for the four algorithms (MGGS, Min–Min, GA, and FCFS) 50 times, and we use the average of the 50 repeated times as the final results.

4.2 Discussion of results

4.2.1 Analysis of total execution time

The total execution time is vital for any task scheduling algorithms. In this paper, we first evaluate the performance of the total execution time for the four algorithms. The size of tasks and processing speed of VMs are shown in Tables 1 and 2, respectively. As the benchmark algorithms are GA, Min–Min, and FCFS, Fig. 11 shows the comparison for the four algorithms, as shown below:

Figure 11 shows the total execution time of the four algorithms (GA, MGGS, Min–Min, and FCFS) under different tasks. The experimental results illustrate that MGGS is the best (leads to the least time), Min–Min is the second, GA is the third, and FCFS is the worst. MGGS is better than GA algorithm in terms of the total execution time, and the reason is that, MGGS algorithm chooses the most efficient individuals to make greedy operator in each generation, for the purpose of obtaining the optimal efficiency for the individual at the present stage. MGGS can largely compensate for the uncertainty caused by crossover and mutation of GA, and improve the probability of excellent individuals in each generation. MGGS algorithm can obtain better individuals with relatively fewer iterations.

Min–Min is better than GA algorithm; the reason can be explained by the fact that Min–Min schedules small tasks to fast VMs to execute each time. GA is better than FCFS, and the reason is that GA optimizes the task scheduling, while FCFS algorithm has not optimized the task scheduling. Figure 11 also displays that the total execution time grows with the increase in the number of tasks.

4.2.2 Analysis of workload balancing

The workload balance is an important indicator which can be used to evaluate task scheduling algorithm. In this paper, the degree of workload balance can be considered as the variance between total execution time and average time. The definition of workload balance (denoted by parameter σ) can be shown as follows:

$$\sigma = \frac{\sum_{i=1}^{N} (X_i - \bar{X})^2}{N} \tag{7}$$

where parameter σ is the degree of workload balance, parameter N is the number of tasks, parameter X_i refers to the execution time of task i, parameter $\bar{X}$ is the average execution time of tasks.

In this paper, the second experiment is to evaluate the σ performance for the four algorithms (GA, MGGS, Min–Min, and FCFS) under different tasks. Figure 12 shows the comparison of the workload balance degree for the four algorithms, as shown:

Figure 12 shows the degree of workload balance of the four algorithms (GA, MGGS, Min–Min, and FCFS) under different tasks. The experimental results illustrate that MGGS is the best (with the least σ value), Min–Min is the second, GA is the third, and FCFS is the worst. The reason is as follows: The essence of greedy algorithm is to get the local optimal solution under the current condition. Therefore, it is inevitable to transfer some workload in VMs with heavy workload into other VMs with lightly workload, so as to make the whole system more balanced. MGGS inherits this characteristic and gives a good performance in the degree of workload balance. On the other hand, Min–

Table 1 Task parameter list

Number	Task length (MIPS)	Number	Task length (MIPS)	Number	Task length (MIPS)
1	45,795	11	10,659	21	52,127
2	56,485	12	41,340	22	44,005
3	16,607	13	54,756	23	57,880
4	49,345	14	21,790	24	48,777
5	40,573	15	21,710	25	37,604
6	44,102	16	42,966	26	12,477
7	44,722	17	34,434	27	12,216
8	33,505	18	38,571	28	42,616
9	36,744	19	17,154	29	26,851
10	53,222	20	58,818	30	45,652

Table 2 The parameter list of VMs

Number	The processing speed of VMs (MIPS/S)
1	261
2	522
3	286
4	693
5	654
6	532
7	288
8	420
9	872
10	408

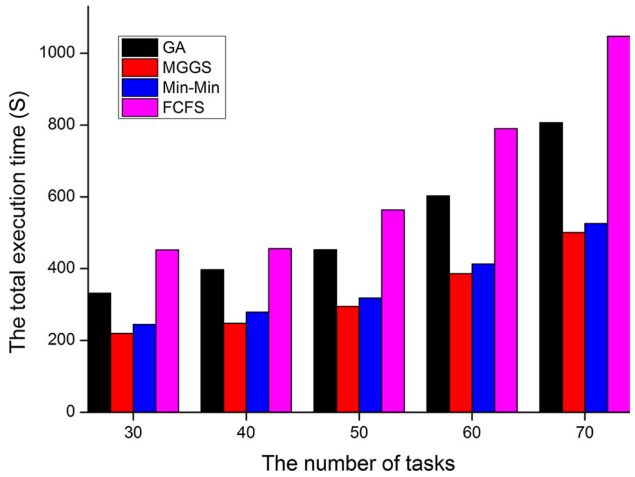

Fig. 11 The total execution time of the four algorithms

Min prefers scheduling small tasks lead to unbalance of workload. GA is better than FCFS, and the reason is that FCFS does not consider the characteristic (such as big tasks and small tasks).

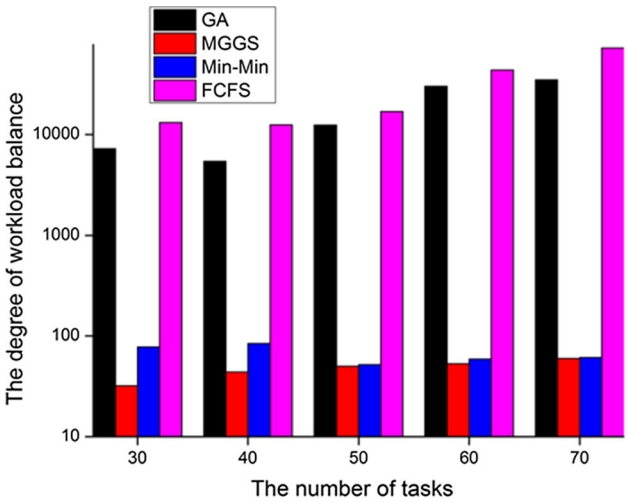

Fig. 12 The degree of workload balance

4.2.3 Analysis of average response time

The average response time is another essential indicator which can be used to measure the performance of the task scheduling algorithm. The average response time of the four algorithms is as follows:

Figure 13 illustrates the average response time of the four algorithms, and the experimental results show that Min–Min is the best (leads to the least time), MGGS is the second, GA is the third, and FCFS is the worst. Min–Min is better than other three algorithms, and the reason is that it prefers scheduling small tasks to be run first, which leads to the shorter wait time of big tasks. Therefore, the average response time of Min–Min is the best.

MGGS is better than GA, and it can be explained that MGGS inherits this characteristic of greedy algorithm. GA outperforms the FCFS, the reason is that GA optimizes the task scheduling, while FCFS algorithm has not optimized the task scheduling. Figure 13 also displays that the average response time grows with the increase in the number of tasks.

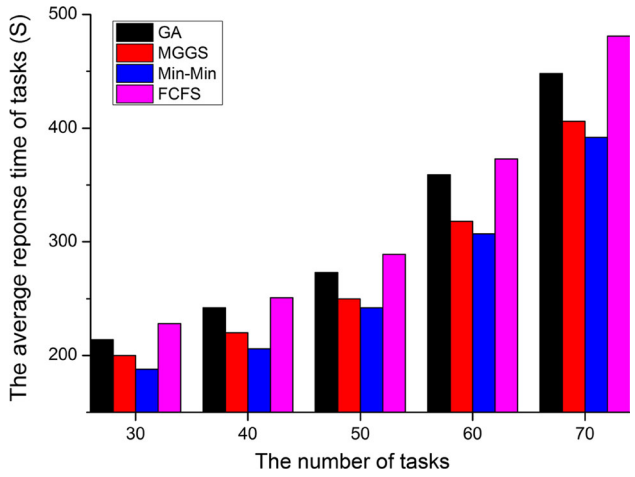

Fig. 13 The average response time of the four algorithms

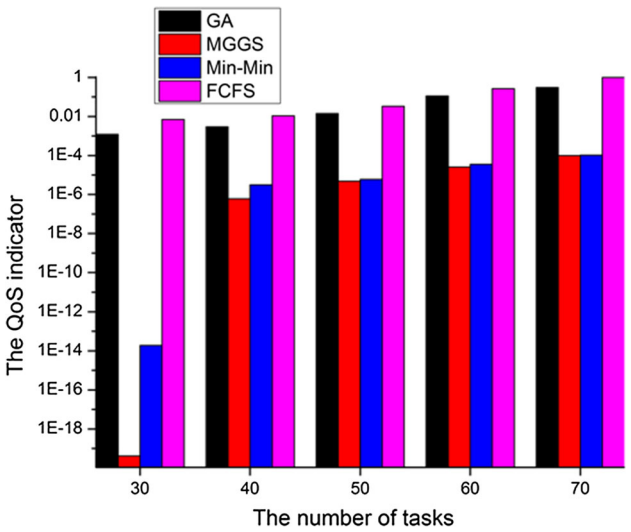

Fig. 14 The QoS delivered by the four algorithms

4.2.4 Analysis of QoS delivered

The QoS is vital for any task scheduling algorithm. Regarding the QoS indicator, different algorithms have different definitions. At present, most task scheduling algorithms only consider one or two indications such as the total execution time or the average response time. Unlike other algorithms, in this paper, we give a QoS definition under considering total execution time, the degree of workload balance, and the average response time. As the total execution time, the degree of workload balance, and the average response time are the same important for task scheduling algorithm. Therefore, the definition of QoS (denoted T_{QoS}) is as follows:

$$T_{QoS} = X_1 \times X_2 \times X_3 \tag{8}$$

where parameter T_{QoS} represents the QoS performance delivered by related algorithms. Parameters X_1, X_2, X_3 refer to the total execution time, the degree of workload balance, and the average response time, respectively. As the total execution time, the degree of workload balance and the average response time are with different units, we adopt "Min–Max" method to do normalization processing.

Figure 14 displays the QoS of the four algorithms (GA, MGGS, Min–Min, and FCFS) under different tasks. The experimental results illustrate that MGGS is the best (leads to the least time), Min–Min is the second, GA is the third, and FCFS is the worst. MGGS is better than other three algorithms, and the reason is that MGGS has advantages in the total execution time and degree of workload balance. Min–Min outperforms GA and FCFS (or GA is better than FCFS), it can be explained by the same reason. Figure 14 also displays that the QoS grows with the increase in the number of tasks.

5 Conclusion

This paper puts forward a novel algorithm (named MGGS) based on the combination of GA algorithm and greedy strategy to optimize the task scheduling. The experimental results show that the proposed algorithm performs well as compared to other task scheduling algorithms. MGGS is expected to be used in cloud-based platforms and improves the return on investment.

In future research plan, we will introduce stochastic service system theory to improve cloud computing for resource management.

Acknowledgement This work was supported by the National Natural Science Foundation of China (Nos. 61572525 and 61772088), the China Postdoctoral Science Foundation (No. 2018M642974), the Scientific research project of education department of hunan province, the Natural Science Foundation of Hunan Province (No. 2019JJ50689) and the Science and Technology Plan Project of Changsha city (No. k1705036).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Liu XF, Zhan ZH, Deng JD et al (2018) An energy efficient ant colony system for virtual machine placement in cloud computing. IEEE Trans Evol Comput 22(1):113–128
2. Zhou Z, Abawajy J, Chowdhury M, Hu Z et al (2017) Minimizing SLA violation and power consumption in cloud data centers using adaptive energy-aware algorithms. Future Gener Comput Syst (FGCS) 86(6):836–850
3. Zhou Z, Abawajy J, Chowdhury M, Li F, Hu Z, Li K et al (2018) Fine-grained energy consumption model of servers based on task

characteristics in cloud data center. IEEE Access 6(1):27080–27090
4. Xie G, Zeng G, Jiang J, Fan C, Li R, Li K (2016) Energy management for multiple real-time workflows on cyber-physical cloud systems. Future Gener Comput Syst 27(7):1915–1928
5. Wu KC, Liu WY, Wu SY (2018) Dynamic deployment and cost-sensitive provisioning for elastic mobile cloud services. IEEE Trans Mob Comput 17(6):1326–1338
6. Hao L, Kenli L, Jiyao A, Keqin L (2018) MSGD: A novel matrix factorization approach for large-scale collaborative filtering recommender systems on GPUs. IEEE Trans Parallel Distrib Syst 29(7):1530–1544
7. Kumar M, Mao YH, Wang YH, Qiu TR, Yang C, Zhang WP (2017) Fuzzy theoretic approach to signals and systems: static systems. Inf Sci 418:668–702
8. Huang SC, Jiau MK, Lin CH (2015) A genetic-algorithm-based approach to solve carpool service problems in cloud computing. IEEE Trans Intell Transp Syst 16(1):352–364
9. Karimi MB, Isazadeh A, Rahmani AM (2017) QoS-aware service composition in cloud computing using data mining techniques and genetic algorithm. J Supercomput 73(4):1387–1415
10. Pizzuti C (2012) A multiobjective genetic algorithm to find communities in complex networks. IEEE Trans Evol Comput 16(3):418–430
11. Manasrah AM, Ali HB (2018) workflow scheduling using hybrid GA-PSO algorithm in cloud computing. Wireless Commun Mobile Comput 2018(3):1–16
12. Chitra OL, Venkatesh P et al (2011) Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system. Int J Inf Technol Commun Converg 2(1):146–158
13. Etminani K, Naghibzadeh M (2007) A min-min max-min selective algorithm for grid task scheduling. In: 3rd IEEE/IFIP International conference in central asia on internet, pp 1–7
14. Moreno R, Alonso-Conde AB (2004) Job scheduling and resource management techniques in economic grid environments. In: Grid computing, Santiago de Compostela, pp 25–32
15. Zhou Z, Hu Z (2014) Task scheduling algorithm based on greedy strategy in cloud computing. Open Cybern Syst J 8(1):111–114
16. Mao Y, Chen X, Li X (2014) Max–Min task scheduling algorithm for load balance in cloud computing. Proc Int Conf Comput Sci Inf Technol 225:457–465
17. Masdari M, Salehi F, Jalali M et al (2016) A survey of PSO-based scheduling algorithms in cloud computing. J Netw Syst Manag 25(1):122–158
18. Almaamari A, Omara FA (2015) Task scheduling using PSO algorithm in cloud computing environments. Int J Grid Distrib Comput 8(5):245–256
19. Yang Z, Qin X, Li W et al (2013) Optimized task scheduling and resource allocation in cloud computing using PSO based fitness function. Inf Technol J 12(23):7090–7095
20. Otero FEB, Freitas AA, Johnson CG (2012) Inducing decision trees with an ant colony optimization algorithm. Appl Soft Comput 12(11):3615–3626
21. Liu X, Yi H, Ni Z (2013) Application of ant colony optimization algorithm in process planning optimization. J Intell Manuf 24(1):1–13
22. Dai Y, Lou Y, Lu X (2015) A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi-QoS constraints in cloud computing. In: IEEE international conference on intelligent human-machine systems and cybernetics, pp 428–431
23. Yuan H, Bi J, Zhou MC et al (2017) WARM: workload-aware multi-application task scheduling for revenue maximization in SDN-based cloud data center. IEEE Access 6(1):645–657
24. Yahyaoui H, Maamar Z, Lim E et al (2013) Towards a community-based, social network-driven framework for Web services management. Future Gener Comput Syst 29(6):1363–1377
25. Calheiros RN, Ranjan R, Beloglazov A et al (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50
26. Zhou Z, Hu Z, Yu J, Abawajy J, Chowdhury M (2017) Energy-efficient virtual machine consolidation algorithm in cloud data centers. J Cent South Univ 24(10):2331–2341