Deadline Constrained Cloud Computing Resources Scheduling for Cost Optimization Based on Dynamic Objective Genetic Algorithm

Zong-Gan Chen, *Student Member, IEEE*, Ke-Jing Du, Zhi-Hui Zhan (Corresponding Author), *Member, IEEE*, and Jun Zhang, *Senior Member, IEEE*

***Abstract*—Cloud computing resources scheduling is significant for executing the workflows in cloud platform because it relates to both the execution time and execution cost. In order to take both the time and cost into consideration, Rodriguez and Buyya have proposed a cost-minimization and deadline-constrained workflow scheduling model on cloud computing. Their model has great applicability but the solution of their particle swarm optimization (PSO) approach is not good enough and cannot meet a tight deadline condition. In this paper, we propose a genetic algorithm (GA) approach to solve this model. In order to tackle with the tight deadline condition, a *dynamic objective strategy* is further proposed to let GA focus on optimize the execution time objective to meet the deadline constraint when the feasible solution hasn't been obtained. After obtaining a feasible solution, the GA focuses on optimizing the execution cost within the deadline constraint. Therefore, the proposed dynamic objective GA (DOGA) has adaptive ability to the search environment to different objectives. We have conduct extensive experiments based on workflows with different scales and different cloud resources. Experimental results show that DOGA can find better solution with smaller cost than PSO does on different scheduling scales and different deadline conditions. DOGA approach is more applicable to be used in commercial activities.**

Keywords*—*cloud computing; resource; scheduling; genetic algorithm; dynamic objective strategy

I. INTRODUCTION

Cloud computing is the development of distributed computing, parallel computing, and grid computing. It relies on not only the applications delivered as services over the Internet, but also the hardware and software in the data centers that provide those services [1]. NIST (National Institute of Standards and Technology)'s definition of cloud computing [2] is that cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or the interaction of service providers.

In 2006, Eric Schmidt, the CEO of Google, first proposed the concept of "Cloud Computing" in SES San Jose 2006. Although the birth of cloud computing is less than 10 years, the present availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture, and autonomic and utility computing have led to a growth in cloud computing. Cloud vendors are experiencing growth rates of 50% per annum [4].

Cloud Computing has several features as follows.

- The vast scale of data. Google Cloud already has over 100 million servers, Amazon, IBM, Microsoft, Yahoo and other "cloud" all have hundreds of thousands of servers. "Cloud" can give users an unprecedented computing power.
- Hardware virtualization. By using virtualization technology, cloud computing can provide users with computer infrastructure, server platforms, application software, and other services via infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), and other models [2][3]. It enables us to manage different types of resource easily [5]. Developers only need to consider the scheduling logic in application level, without the need to consider the underlying resource scheduling.
- Expansibility. The scale of "cloud" is flexible so that it can meet the need of application and the growth of the quantity of users.
- On-demand service. "Cloud" is a huge resource pool and users can pay money to lease some resources that meet their demands.

In the dawn of "era of big data", cloud computing is faced with those problems that have a huge scale of data. Therefore, workflow scheduling on clouds has become a significant research topic. It is related to the cost and efficiency of cloud computing. But in reality, workflow scheduling is a NP-hard problem so that it is impossible to generate an optimal

Z.-G. Chen, K.-J. Du, Z.-H. Zhan, and J. Zhang are with the Department of Computer Science, Sun Yat-Sen University, Guangzhou, 510275, China, with the School of Advanced Computing, Sun Yat-Sen University, Guangzhou, 510275, China, with the Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China, with the Engineering Research Center of Supercomputing Engineering Software (Sun Yat-sen University), Ministry of Education, China, and also with the Key Laboratory of Software Technology, Education Department of Guangdong Province, China. Zhi-Hui Zhan is the corresponding author, email: zhanzhh@mail.sysu.edu.cn.

This work was supported in part by the, in part by the National Natural Science Foundations of China (NSFC) with No. 61402545, the Natural Science Foundations of Guangdong Province for Distinguished Young Scholars with No. 2014A030306038, the Project for Pearl River New Star in Science and Technology, Guangzhou, China, the Fundamental Research Funds for the Central Universities, the NSFC Key Program with No. 61332002, the NSFC for Distinguished Young Scholars with No. 61125205, and the National High-Technology Research and Development Program (863 Program) of China No.2013AA01A212.

978-1-4799-7492-4/15/$31.00 ©2015 IEEE

solution within polynomial time. What we can do is to find an approximate or near-optimal solution. With the development of intelligent computing algorithm, using intelligent computing algorithm to solve the workflow scheduling problem of cloud computing has been popular.

Many researches concentrate on minimizing the workflow's execution time. For example, Rahman et al. [6] used the workflow's dynamic critical paths to find a solution. Chen and Zhang [7] used ant colony optimization (ACO) to meet various kinds of QoS requirement. But in the business activities, money is definitely one of the factors that cannot be ignored. The pursuit of solution that completes the task fastest often requires a larger investment. As a result, we need to make balance between cost and time in order to find a solution that can maximize the economic profit. Mao and Humphrey [8] proposed a dynamic approach for scheduling workflow. They used the methods of finding the most economical and practical resource and merging several tasks to a single one. However, they can only find the cheap solution but not the near-optimal solution. Malawski et al. [9] used various dynamic and static algorithms to generate a solution that can meet users' QoS. However, in their method, only one type of virtual machine (VM) is used, which ignores the elasticity and heterogeneity of the resource in cloud computing.

In 2014, Rodriguez and Buyya proposed a deadline based resource provisioning and scheduling algorithm [10]. They develop a cost-minimization and deadline-constrained model, which means to meet the deadline constraint and minimize the cost. This model has great progressiveness. It tally with business need that a workflow task should be done within a deadline and what we should do is minimizing the cost in order to gain biggest profit. In order to solve their optimization model, they used particle swarm optimization (PSO) as their approach. Although promising results have been obtained by their PSO approach, there are still rooms for enhancement.

1. In the particle code, the index of the resource is used to represent the position of the particle. However, resource index doesn't carry any feature of the resource. As a result, the process that particle fly to globally best and personal best may not lead to a better solution.
2. If the scale of scheduling problem is large, PSO may easily converge to a local optimal solution.
3. When the deadline is tight, PSO is difficult to find a feasible solution.

In this paper, we propose a genetic algorithm (GA) based approach to solve the deadline based cloud resource scheduling model [10], which is proposed by Rodriguez and Buyya. We still use the index of the resource to code the chromosome. However, the index is with integer value but not floating value like in PSO. Moreover, as GA doesn't has a learning process like PSO, and the crossover of GA can preserve the good combination of different index values, GA would have a good guidance to promising solution by using such code scheme.

We propose a dynamic objective strategy (DOS) to deal with tight deadline constraint. When all the chromosomes in the population cannot find a feasible solution, we set execution time as the optimization objective until one of chromosome find a feasible solution. Once at least a feasible solution has been found, we set execution cost as the optimization objective. We name this DOS based GA approach as DOGA.

In this paper, experimental study show that our proposed DOGA approach is better than the PSO approach in solving the cost-minimization and deadline-constrained model. DOGA can not only generate a solution with smaller cost than PSO does in many test cases, but also has better adaptability than PSO, which mean that DOGA can meet a tighter deadline while PSO sometimes fails to obtain feasible solutions.

The rest of this paper is organized as follows. Section II presents background introduction of the deadline based model and its solution code and fitness evaluation. Section III presents the DOGA approach and the strategy to solve the situation of tight deadline. Section IV presents the experimental results. Finally, Section V presents the conclusion.

II. BACKGROUND

A. Cost-minimization and Deadline-constrained Model

The workflow has a set of tasks, and these tasks have topology structure. An example is shown in Fig. 1. The number on the line represents the time that parents transfer data to its child when they runs on different resources. And there is a set of resource, named R_{ini}, that can be leased to execute the tasks.

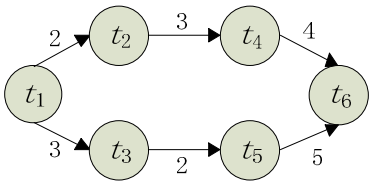

Figure 1. A simple example of workflow

We define the '*total execution cost*' (*TEC*) and the '*total execution time*' (*TET*) as follows:

$$TET = \max\{ET_{t_i} : t_i \subset T\} \quad (1)$$

$$TEC = \sum_{j=1}^{|R|}(C_{r_j} \times (LET_{r_j} - LST_{r_j})) \quad (2)$$

Where ET_{t_i} represents the '*end time*' that the task t_i ends its execution. C_{r_i} represents the cost to lease the resource r_j for a unit of time. LST_{r_i} represents the '*lease start time*' of r_j while LET_{r_i} represents the '*lease end time*' of r_j. *TET* is calculated by the end time of the latest finish task. *TEC* is calculated by accumulating the cost to lease every resource while the cost to lease every resource is calculated by multiplying the cost of the resource by its lease time.

As we want to find a solution that minimizes the cost while meet the deadline constraint, it means that the target is

to minimum *TEC* while its *TET* should meet the deadline constraint, the formulas is shown below:

$$Minimize\ TEC \tag{3}$$

$$TET < deadline \tag{4}$$

B. PSO Framework

PSO is an evolutionary computational algorithm, which is proposed by Eberhart and Kennedy in 1995[11], and has fast developed in recent years [12][13][14]. Every particle has two vectors, x and v. The x represents the position of the particle, which is also a solution to defined problem. The v determines the movement of the particle in the defined problem space. Every particle's best position *pBest* and the best position of the population *gBest* will be stored and updated. For every particle i, the update of the x and v in every generation is shown in formula (5) and (6).

$$x_i^d = x_i^d + v_i^d \tag{5}$$

$$\begin{aligned} v_i^d = {} & \mathrm{w} \times v_i^d + c_1 \times r_1 \times \left(pBest_i^d - x_i^d \right) \\ & + c_2 \times r_2 \times \left(gBest_i^d - x_i^d \right) \end{aligned} \tag{6}$$

Where d represents the dimension of the vector, ω is inertia weight while c_1 and c_2 is the acceleration coefficients [15][16]. r_1 and r_2 are random numbers between 0 and 1. In every generation, evaluate the fitness of every particle and update their *pBest* and *gBest*. Finally, update the x and v of every particle according to formula (5) and (6).

C. Scheduling Scheme and Fitness Evaluation

As our DOGA uses a similar scheduling scheme to encode the solution, we describe the particle code scheme herein according to [10]. In order to programing the PSO approach, the index of the resource is used to encode the particle. The length of the position is the same as the number of tasks, and the value of each dimension represents the corresponding resource that executes this task. Fig. 2 is a simple example with 6 tasks and 3 resources.

Particle's position

Dim.1	Dim.2	Dim.3	Dim.4	Dim.5	Dim.6
1.2	2.3	3.1	1.8	2.5	3.7

Corresponding Task Scheduling

$t_1 \rightarrow r_1$	$t_2 \rightarrow r_2$	$t_3 \rightarrow r_3$	$t_4 \rightarrow r_1$	$t_5 \rightarrow r_2$	$t_6 \rightarrow r_3$

Figure 2. An example of encoding the particle.

As the figure shows, every dimension stands for corresponding task, and its value represents the resource that the task is scheduled to.

In this case, formula (3) and (4) are used to evaluate a particle's fitness. So we need to obtain *TEC* and *TET* from the particle's position, and then calculate a fitness function.

$$transfertime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \begin{bmatrix} 0 & 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad exetime = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 1 & 4 & 6 \\ 5 & 4 & 7 \\ 3 & 6 & 8 \\ 2 & 1 & 4 \\ 5 & 4 & 9 \\ 3 & 2 & 5 \end{bmatrix} \end{array}$$

Figure 3. An example of transfertime and exetime

In order to obtain *TEC* and *TET*, we need to know the time that tasks execute in every resource. The execution time is stored in an array *exetime*, where *exetime*[i][j] represents the execution time that t_i runs on the resource r_j. However, when parents task and its child are executed in different resource, parents may need some time to transfer data to its child. In this case, the transfer time in the array *transfertime* is used, where *transfertime*[i][j] represents the time that t_i transfers data to t_j. An example of *transfertime* and *exetime* corresponding to Fig.1 is shown in Fig. 3.

As we have known these variables, we can obtain the workflow scheduling. The workflow scheduling corresponding to Fig. 2 and Fig. 3 is shown in Fig. 4.

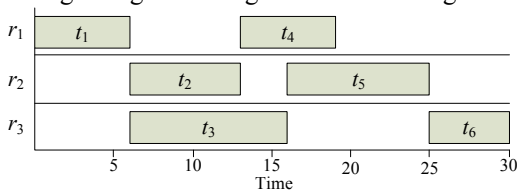

Figure 4. The workflow scheduling

Calculating *TEC* and *TET*

Input: array *pos*[|*T*|] represents a solution
Ouput: *TEC* and *TET*
Void Calculating_*TEC_TET*()
{
  R=∅, *TEC*=0, *TET*=0
  for each i=0 to i=|*T*|-1
    if t_i has no parents
      $ST_{t_i} = LET_{r_{pos[i]}}$
    else
      $ST_{t_i} = \max(\max\{ET_{t_p} : t_p \in \text{parents}(t_i)\}, LET_{r_{pos[i]}})$
    end if
    $exe = exetime[i][pos[i]]$; $transfer = 0$;
    for each child t_c of t_i
      if t_c is not mapped to $r_{pos[i]}$
        $transfer += transfertime[i][c]$
      end if
    end for each
    $PT_{t_i}^{r_{pos[i]}} = exe + transfer$; $ET_{t_i} = ST_{t_i} + PT_{t_i}^{r_{pos[i]}}$;
    if $r_{pos[i]} \notin R$
      $LST_{r_{pos[i]}} = ST_{t_i}$
      $R = R \bigcup \{r_{pos[i]}\}$
    end if
    $LET_{r_{pos[i]}} = ET_{t_i}$
  end for each
  $TET = \max\{ET_{t_i} : t_i \subset \mathrm{T}\}$
  $TEC = \sum_{i=1}^{|R|} \mathrm{C}_i * (LET_{r_i} - LST_{r_i})$
}

Figure 5. The pseudo-code to calculate *TEC* and *TET*

The pseudo-code to calculate *TEC* and *TET* is shown in Fig. 5. At first, we will initialize *TET*, *TEC*, and *R*. *R* is a set of resource that we lease during the execution, initialized as ∅. And then the algorithm will iterate every coordinate i. For every task t_i, we can get the resource it runs according to *pos*[i]. t_i's start time ST_{t_i} is determined by two factors. It

should wait until $r_{pos[i]}$ is available, which is $LET_{r_{pos[i]}}$, and if t_i has parent t_a, t_i should wait until t_a end its execution. Execution time *exe* plus the time that t_i use to transfer data to its child, named *transfer*, is t_i's processing time PT_{t_i}. The end time of t_i, which is ET_{t_i}, is calculated by ST_{t_i} plus PT_{t_i}. If $r_{pos[i]}$ hasn't been leased, the lease start time of $r_{pos[i]}$, which is $LST_{r_{pos[i]}}$, is ST_{t_i}. The lease end time $LET_{r_{pos[i]}}$ is the end time of t_i. Finally, we can calculate *TEC* and *TET* through formula (1) and (2) and then we can obtain *TEC* and *TET* of the solution.

III. DOGA APPROACH

GA [17][18], proposed by Holland, is a search heuristic that mimics the process of natural selection. It can be applied to optimization problems according to selection, crossover, and mutation. In our proposed DOGA approach, it has similar algorithmic structure to traditional GA except that DOGA uses a dynamic objective strategy to adaptively evaluate the chromosomes during the evolutionary process. The whole flowchart of DOGA is illustrated in Fig. 6 and is described in the following subsections.

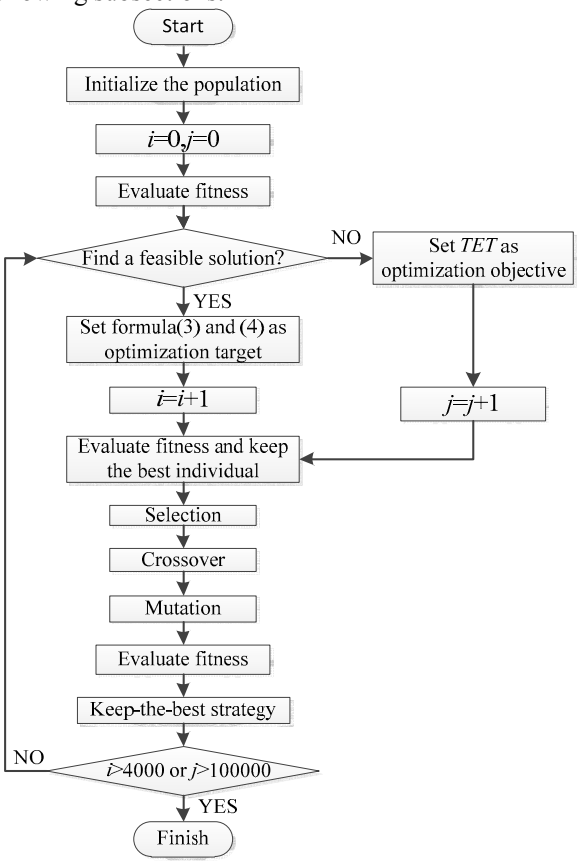

Figure 6. The flowchart of DOGA approach

A. Encoding

The way to encode the chromosomes of population is similar to PSO approach. The only difference is that we use integer rather than floating number. The coordinate i's value represents the resource that t_i runs on. For example, $\dim_i=j$ represents that t_i runs on the resource r_j.

B. Selection

In DOGA, we use roulette wheel selection [19] to select chromosome to next generation. The chromosome with bigger fitness would be more likely to be chosen. As we want to minimize the cost, we use the $1/TEC$ to evaluate the chromosome's fitness. Formulation is shown below.

$$fitness_i = 1/TEC_i \tag{7}$$

$$p_i = \frac{fitness_i}{\sum_{j=0}^{N-1} fitness_j} \tag{8}$$

C. Crossover

Every chromosome has a probability P_c to crossover. If *Random* (0, 1) < P_c, the chromosome can crossover. The crossover is processed by generating a random number n and then two chromosome can exchange the first n coordinate. A simple example of crossover is shown in Fig. 7.

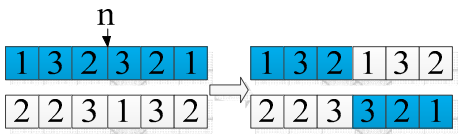

Figure 7. Example of crossover

D. Mutation

Every coordinate of a chromosome has a probability P_m to mutation. The mutation is processed by generating two random number, m and v, and then the value of coordinate m will mutate to *v*. A simple example of mutation is shown in Fig. 8.

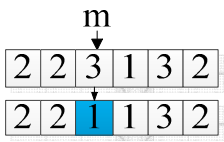

Figure 8. Example of mutation, *v*=1

E. Keep-the-Best Strategy

De Jong proposed "maintain the best solution found over time before selection" [19] to avoid wrecking the best chromosome during mutation or crossover. In our approach, we define a variable to store the global best chromosome. While in every generation, if the best chromosome is better than the global best one, update the global best chromosome. If not, the global best chromosome will replace the worst chromosome of the current generation. This strategy can accelerate the convergence of DOGA.

F. Dynamic Objective Strategy

GA is not designed to solve a constraint problem. So when the deadline is tight, the algorithm cannot even find a feasible solution in the first generation and if we still use formula (3) and (4) to evaluate the fitness, the algorithm may be difficult to find a feasible solution. This problem is more serious in PSO, because in first generation there's no feasible solution, which means that there's no gbest and pbest, so that PSO cannot continue running. Rodriguez and Buyya [10]used the constraint-handling strategy proposed by Deb et al. [20]. That

is, when both solutions are infeasible, the one with smaller overall constraint violation is selected, which means that if two solutions cannot meet deadline constraint, the one with a smaller *TET* is selected.

Differently, in this paper, we proposed a dynamic objective strategy (DOS). In order to programing, if the chromosome cannot meet the deadline constraint, we defined its fitness as 1/1000000, which is far smaller than the feasible chromosome's fitness. And then if all the chromosomes' fitness in the population is 1/1000000, which means no chromosome can find a feasible solution, DOGA sets *TET* as the optimization objective until one of chromosome obtain a feasible solution. After that, DOGA will still use the formula (3) and (4) to evaluate a chromosome's fitness. By using the DOS, when deadline is very tight, DOGA will first try to find a solution with smaller *TET* that can meet the deadline. As a result, DOGA is more likely to meet tight deadline. Moreover, DOGA can use different P_c and P_m values when setting different optimization objective so that the algorithm can be more adaptive.

After finding a feasible solution, the algorithm will run 4000 generations to find the final best solution. We identify that the algorithm cannot find a solution if the algorithm cannot find a feasible solution within 100000 generations.

IV. EXPERIMENTS AND COMPARISONS

In our experiment, for resource r_j, its price C_j is a random value generated in range [1, 5], which is shown in formula (9). For task t_i, the $exetime[i][j]$ is determined by two factor. We know that the resource with high cost may be more likely to have good efficiency to execute tasks. We use $6 - C_j$ to represents this factor. But there are various kinds of tasks and resources with different properties and some resources may have a good efficiency to execute some of the tasks so that cost cannot totally determine the efficiency of the execution of all of the tasks. We use *Random* (1, 5) to represents this factor. We consider that these two factors have the same influences to the execution time. So we use the formula (10) to determine the $exetime[i][j]$. We set $transfertim[i][child(i)]$ as $Random(0,1)$, which is shown in formula (11).

$$C_j = Random(1,5) \tag{9}$$

$$exetime[i][j] = (6 - C_j) \times 0.5 + Ramdom(1,5) \times 0.5 \tag{10}$$

$$transfertime[i][child(i)] = Random(0,1) \tag{11}$$

Generating topology structure of tasks
For i=0 to $|T|$-1
  P_{child}=0.1+0.4*(i/$|T|$)
  for k=i to $|T|$
    if $Random(0,1)<P_{child}$
      Set t_k as the child of t_i
    end if
  end for each
end for each

Figure 9. Algorithm of generating topology structure.

We use algorithm in Fig. 9 to generate topology structure of the tasks, we consider that the tasks can be executed sequentially, which means for t_i, $child(i)>i$. For t_i, we set P_{child} to be the probability that a task t_k to be the child of t_i. In order to generate a balanced structure, P_{child} is increased with the increasing of i.

We use 3 different scales of data to do our experiment. We first compare GA with DOGA to show the effect of the dynamic objective strategy. And then we will compare the results of DOGA and PSO by representing the result of every scale of data via a table and a figure.

DOGA and PSO may need some generations to find a feasible solution, named find generation (*FGEN*). If *FGEN* >100000, we regard that the algorithm cannot find a solution. Otherwise, the algorithm runs more 4000 generations after finding a feasible solution, so as to find the best solution with smallest *TEC*. We define *Time* as the computational time of two algorithms, measured by second. The results will compare the *FGEN* and *TEC* of PSO and DOGA under different deadline constraints. Moreover, we will plot the convergence curves on the *TEC* metric along the 4000 generations of DOGA and PSO. The X-axis means generation, named *GEN* and the Y-axis means *TEC*.

In PSO approach, we follow the proposals in [10] as c_1=c_2=2.0, ω=0.5, and the population is 100. The population of DOGA is also set to be 100. For the crossover and mutation probabilities of DOGA, as we use dynamic objective strategy in different stages, therefore, if the algorithm is in the stage of finding a feasible solution, we set formula (3) and (4) to as optimization target, and set P_c =0.15 and P_m =0.008; otherwise, if the algorithm is in the stage of finding the smallest *TET* during the 4000 generations, we set *TET* as optimization target, we set P_c =0.8 and P_m =0.002. Moreover, because of the randomness, we execute the both DOGA and PSO 30 times on each instance and use the average result to evaluate these two approaches.

A. Compare GA with DOGA

As is mention in Section III, dynamic objective strategy is designed to meeting tight deadline constraint. In this section, we will compare the tightest deadline that DOGA and GA can meet to show the effect of the strategy.

TABLE I. THE TIGHTEST DEADLINE THAT GA AND DOGA CAN MEET

Case	GA	DOGA	(GA-DOGA) /GA
50 tasks 15 resources	115	**60**	47.8%
75 tasks 25 resources	200	**100**	50.0%
100 tasks 30 resources	330	**150**	54.5%

Notice that the results in Table I are measured by 5 seconds as a unit. Table I shows that DOS can help GA more adaptive to deal with the tighter deadline in the cost-minimization and deadline-constrained model. More significantly, as the scheduling scale being increasing, the advantages of DOGA become more obvious, as indicating by the value of (GA-DOGA)/GA.

B. *Compare PSO with DOGA*

1) *Small Scale of Data*

In the small scale of data, the workflow has 50 tasks and there are 15 kinds of resource that can be leased. We set 3 deadlines as 80, 100, and 120. The results are compared in Table II and Fig. 10.

The better results in the Table are marked with **boldface**. The results show that DOGA can always obtain smaller *TEC* values than PSO does under different deadline constraints. Moreover, the computational time of DOGA is always less than PSO. The *FGEN* values tell how many generations PSO and DOGA have to use until they can find at least one feasible solution. When the deadline constraint is large, e.g., both PSO and DOGA can obtain feasible solution at the beginning, with *FGEN*=0. The PSO approach may convergent faster than DOGA sometimes in obtaining the feasible solution, as indicated by the case with deadline as 100. However, when the deadline become tighter, e.g., 80, PSO cannot even find a feasible solution while DOGA can still find feasible solution in about 743.5 generations.

TABLE II. FGEN AND TEC UNDER DIFFERENT DEADLINE CONSTRAINT ON SMALL SCALE OF DATA

Dead line	PSO			DOGA		
	FGEN	*TEC*	*Time*	*FGEN*	*TEC*	*Time*
120	**0**	315.87	16	**0**	**259.92**	**13**
100	**63.3**	484.12	18	188.1	**367.63**	**14**
80	N/A	N/A	N/A	**743.5**	**424.59**	**16**

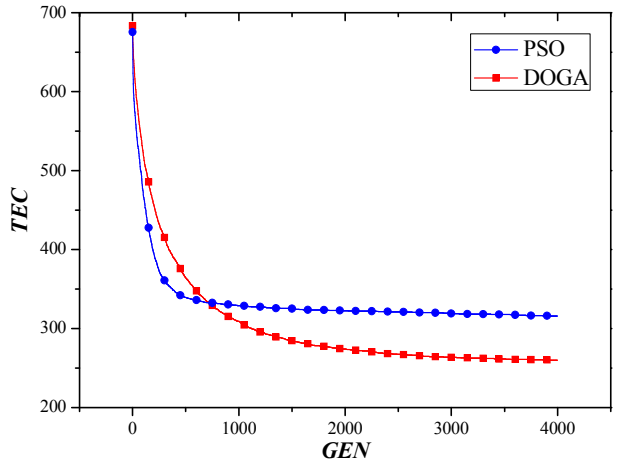

Figure 10. The results on small scale of data with deadline=120

2) *Medium Scale of Data*

In the medium scale of data, the workflow has 75 tasks and there are 25 kinds of resource that can be leased. We set 3 deadline, 120, 200, and 280. The results are compared in Table III and Fig. 11.

TABLE III. FGEN AND TEC UNDER DIFFERENT DEADLINE CONSTRAINT ON MEDIUM SCALE OF DATA

Dead line	PSO			DOGA		
	FGEN	*TEC*	*Time*	*FGEN*	*TEC*	*Time*
280	**0**	776.689	31	**0**	**540.829**	**27**
200	**11.9**	914.840	32	21.1	**663.633**	**28**
120	N/A	N/A	N/A	**3670.1**	**738.395**	**51**

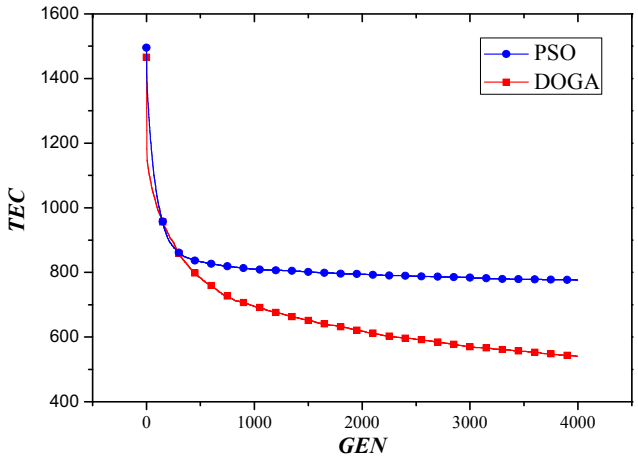

Figure 11. The results on medium scale of data with deadline=280

The results in Table III also show that DOGA can do better than PSO under different deadline constraints, with smaller final *TEC* values and with less CPU computational time. Moreover, the curves in Fig. 11 show that DOGA has faster convergence speed than PSO on medium scale of data with deadline=280.

3) *Large Scale of Data*

In the large scale of data, the workflow has 100 tasks and there are 30 kinds of resource that can be leased. We set 3 deadline constraints as 200, 300, and 400. The results are compared in Table IV and Fig. 12. These compared results further show that DOGA can obtain feasible solutions under different deadline constraints and that DOGA has general better performance than PSO in terms of both solution quality and computational time.

TABLE IV. FGEN AND TEC UNDER DIFFERENT DEADLINE CONSTRAINT ON LARGE SCALE OF DATA

Dead line	PSO			DOGA		
	FGEN	*TEC*	*Time*	*FGEN*	*TEC*	*Time*
400	**0**	1169.462	52	**0**	**756.595**	**45**
300	**93.33**	1647.823	54	245.6	**978.411**	**48**
200	N/A	N/A	N/A	**4490.3**	**1164.298**	**92**

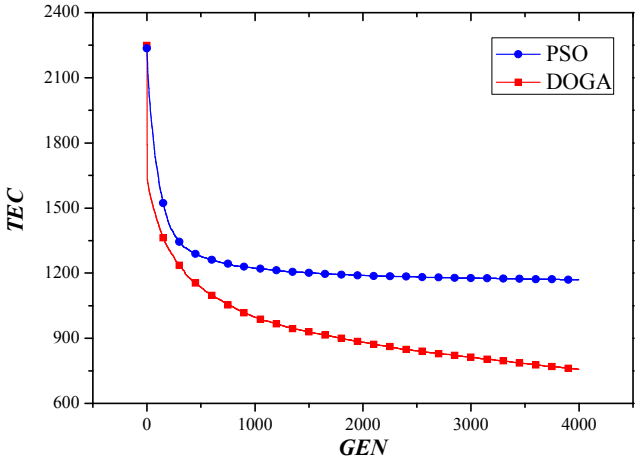

Figure 12. The results on large scale of data with deadline=400

4) Analysis

1. Convergence

In the case of small scale of data, PSO's convergence speed is better than DOGA. With the increasing scale of data, DOGA has better convergence speed than PSO.

2. Result

In 3 different scales of data, DOGA can generate a better solution with smaller *TEC* than PSO does. With the increasing scale of data, the disparity between PSO and DOGA is more and more obvious.

3. Meeting deadline constraint

In 3 different scales of data, DOGA can meet a much tighter deadline than PSO. So DOGA has a higher applicability than PSO, which means that DOGA is more likely to meet the needs of different kinds of users.

V. CONCLUSION

In this paper, we propose a DOGA approach to solve the resource scheduling problem in cloud computing environment under the cost-minimization and deadline-constrained models [10]. The model has great availability and can meet the needs of business organizations. But the PSO approach they proposed [10] has some problems and the result is not good enough. So we use GA to solve this model. During experiment, we found that GA still cannot get the solution in the case of tight deadline so we proposed a GA approach with *dynamic objective strategy (DOS)*, named DOGA. The experiments under different scheduling scales and different deadline constraints show that DOGA is more adaptive to the constraint of various deadlines and can find a better solution with smaller *TEC* than PSO does.

In the future work, other evolutionary computation algorithms such as ant colony optimization [21], differential evolution [22], artificial bee colony [23], and brain storm optimization [24] will be investigated to solve this resource scheduling problem in cloud computing environment. Moreover, dynamic and multi-objective characteristics will be considered and related algorithms would be studied [25][13].

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53 no. 4, pp. 50-58 Apr. 2010.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," *Communications of the ACM*, vol.53, no.6, pp. 50-50, Jun. 2011.

[3] V. William, B. James, and R. Buyya, "Introduction to cloud computing," *Cloud Computing: Principles and Paradigms. New York, USA: Wiley Press*, 2011, pp. 3–37.

[4] "The economy is flat so why are financials Cloud vendors growing at more than 90 percent per annum?". *FSN*. March 5, 2013.

[5] X. F. Liu, Z. H. Zhan, K. J. Du, and W. N. Chen, "Energy aware virtual machine placement scheduling in cloud computing based on ant colony optimization approach," in *Proc. Genetic Evol. Comput. Conf.*, Vancouver, Canada, Jul., 2014, pp. 41-47.

[6] M. Rahman, S. Venugopal, and R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," in *Proc. 3rd IEEE Int. Conf. e-Sci. Grid Comput.*, 2007, pp. 35–42.

[7] W. N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev.*, vol. 39, no. 1, pp. 29–43, Jan. 2009.

[8] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal*., 2011, pp. 1–12.

[9] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, pp. 1–11.

[10] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235 April-June 2014

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. 6th IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[12] M. Shen, Z. H. Zhan, W. N. Chen, Y. J. Gong, J. Zhang, and Y. Li, "Bi-velocity discrete particle swarm optimization and its application to multicast routing problem in communication networks," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 7141-7151, Dec. 2014.

[13] Z. H. Zhan, J. Li, J. Cao, J. Zhang, H. Chung, and Y. H. Shi, "Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 445-463, April. 2013.

[14] Y. H. Li, Z. H. Zhan, S. Lin, J. Zhang, and X. N. Luo, "Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems," *Information Sciences*, vol. 293, no. 1, pp. 370-382, 2015.

[15] Z. H. Zhan, J. Zhang, Y. Li, and H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics--Part B*, vol. 39, no. 6, pp. 1362-1381, Dec. 2009.

[16] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 6, pp. 832-847, Dec. 2011.

[17] Z. H. Zhan, G. Y. Zhang, Y. J. Gong, and J. Zhang, "Load balance aware genetic algorithm for task scheduling in cloud computing," in *Proc. Simulated Evolution And Learning*, 2014, pp. 644-655.

[18] J. Zhang, Z. H. Zhan, Y. Lin, N. Chen, Y. J. Gong, J. H. Zhong, H. S.H. Chung, Y. Li, and Y. H. Shi, "Evolutionary computation meets machine learning: A survey," *IEEE Computational Intelligence Magazine*, vol. 6, no. 4, pp. 68-75, Nov. 2011.

[19] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," *Ph.D. Dissertation*, University of Michigan, 1975.

[20] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[21] Z. H. Zhan, J. Zhang, Y. Li, O. Liu, S. K. Kwok, W. H. Ip, and O. Kaynak, "An efficient ant colony system based on receding horizon control for the aircraft arrival sequencing and scheduling problem," *IEEE Trans. Intell. Transport. Syst.*, vol. 11, no. 2, pp. 399-412, Jun. 2010.

[22] Y. L. Li, Z. H. Zhan, Y. J. Gong, W. N. Chen, J. Zhang, and Y. Li, "Differential evolution with an evolution path: A DEEP evolutionary algorithm," *IEEE Trans. on Cybernetics*, DOI: 10.1109/TCYB.2014.2360752, 2014.

[23] M. D. Zhang, Z. H. Zhan, J. J. Li, and J. Zhang, "Tournament selection based artificial bee colony algorithm with elitist strategy," in *Proc. Conf. Technologies and Applications of Artificial Intelligence*, Taiwan, Nov. 2014, pp. 387-396.

[24] Z. H. Zhan, J. Zhang, Y. H. Shi, and H. L. Liu, "A modified brain storm optimization," in *Proc. IEEE World Congr. Comput. Intell.*, Brisbane, Australia, Jun. 2012, pp. 1-8.

[25] Z. H. Zhan, J. J. Li, and J. Zhang, "Adaptive particle swarm optimization with variable relocation for dynamic optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 1565-1570.