2012 IEEE International Conference on Systems, Man, and Cybernetics
October 14-17, 2012, COEX, Seoul, Korea

A Set-Based Discrete PSO for Cloud Workflow Scheduling with User-Defined QoS Constraints

Wei-Neng Chen, and Jun Zhang (Corresponding Author)
Department of Computer Science, Sun Yat-sen University
Key Laboratory of Machine Intelligence and Sensor Network, Ministry of Education
Key Laboratory of Software Technology, Education Dept. of Guangdong Province, P.R. China
junzhang@ieee.org

***Abstract*—Cloud computing has emerged as a powerful computing paradigm that enables users to access computing services anywhere on demand. It provides a flexible way to implement computation-intensive workflow applications on a pay-per-use basis. Since users are more concerned on the satisfaction of Quality of Service (QoS) in cloud systems, the cloud workflow scheduling problem that addresses different QoS requirements of users has become an important and challenging problem for workflow management in cloud computing. In this paper, we tackle a cloud workflow scheduling problem which enables users to define various QoS constraints like the deadline constraint, the budget constraint, and the reliability constraint. It also enables users to specify one preferred QoS parameter as the optimization objective. A set-based PSO (S-PSO) approach is proposed for this scheduling problem. As the allocation of service instances can be regarded as the selection problem from a set of service instances, it is found the set-based representation scheme in S-PSO is natural for the considered problem. In addition, the S-PSO provides an effective way to take advantage of problem-based heuristics to further accelerate search. We define penalty-based fitness functions to address the multiple QoS constraints and integrate the S-PSO with seven heuristics. A discrete version of the comprehensive learning PSO (CLPSO) algorithm based on the S-PSO method is implemented. Experimental results show that the proposed approach is very competitive especially on the instances with tight QoS constraints.**

Keywords-particle swarm optimization; set-based; workflow scheduling; cloud computing

I. INTRODUCTION

In the last few years, cloud computing has emerged as a powerful and promising next-generation computing paradigm that supports reliable computing services to potentially numerous remote users with diverse requirements [1]. According to Buyya et al. [2], a Cloud is defined as a parallel and distributed system which is composed of a collection of interconnected and virtualized computing resources. These computing resources unify as one or more resource clusters based on service-level agreements. In this way, cloud computing provides a simple and flexible way for users to achieve computing services on-demand.

One unique characteristic of the cloud computing paradigm is that Clouds are usually constructed under a market-oriented architecture [2]. For example, in the Amazon Elastic Compute Cloud (EC2) running in the offering mode of Infrastructure as a Service (IaaS), users can rent virtual computers to run their own computing applications on a "pay-per-use" basis. One momentum of cloud computing is the economies of scale [3]. Compared with the traditional case that each user needs to have his own computing resources to implement his applications, in a cloud system the massive computing resources are maintained by computing resource providers in specially designed data centers. In this way, the costs of managing and operating computing resources can be significantly reduced. In addition, just like the electrical power Grid, users can obtain computing services on-demand and only pay for what they consume. As such, users can flexibly scale up and down the computing infrastructure according to the application's quality of service (QoS) demands and the users' budgets [4].

The development of cloud computing enables scientists to build complex models, manage large data sets and implement computation-intensive numerical and in-silico experiments [5]. Usually, complex scientific computing applications are managed in a workflow model. A workflow is defined as a collection of atomic tasks that are processed in a specific order to accomplish a complicated goal [6]. To manage a scientific workflow, the workflow management system (WfMS) needs to schedule and execute the workflow in an efficient way to satisfy the requirements of scientists. Workflow scheduling is an important and challenging problem for the management of scientific workflows in cloud computing [7].

The research into workflow scheduling has attracted increasing attention in recent years. In general, a workflow is described by a directed acyclic graph (DAG). Many DAG-based heuristics like Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Sufferage, Random, and Heterogeneous Earliest Finish Time [8] have been proposed. There are also tools like the Condor DAGMan for managing workflow applications based on the DAG [9]. However, most of these approaches only consider a single QoS parameter, namely the execution time of the workflow. As the cloud computing paradigm enables the WfMS to consider the different QoS requirements of users, the above traditional workflow scheduling methods become unsuitable for the new paradigm.

To address the various QoS requirements specified by users, Cheng developed a heuristic generic algorithm [10], and Zhi et al. developed a particle swarm optimization (PSO) approach [11]. Both of these approaches aim at minimizing the weight

This work was supported in part by the National Science Fund for Distinguished Young Scholars No.61125205, National Natural Science Foundation of China No.61070004, and NSFC Joint Fund with Guangdong under Key Project U0835002.

978-1-4673-1714-6/12/$31.00 ©2012 IEEE

sum of the workflow cost and makespan. However, users of a cloud system may consider other QoS parameters like the reliability of services. In addition, QoS requirements are usually specified by users as constraints, for example, the deadline and the budget of a workflow. In this case, it is difficult to determine the suitable weights in these approaches. In the economic Grid computing platform which can be viewed as a predecessor model of cloud computing, Yu et al., [12] has developed a workflow scheduling model that can deal with multiple QoS constraints defined by application users. A Markov Decision Procedure (Deadline-MDP) approach has also been developed. In [13], Chen and Zhang further proposed an ant colony optimization (ACO) based approach to the Grid workflow scheduling problem. But these approaches still need to adapt to use in the cloud computing paradigm.

To provide an effective approach to the cloud workflow scheduling problem with various user-defined QoS constraints, this paper intends to develop a set-based particle swarm optimization approach. PSO was initially introduced in 1995 by Kennedy and Eberhart [14]. The basic idea of PSO is to simulate the social intelligent behavior of birds flocking and fish schooling. PSO is simple in concept, easy in implementation, and has a fast converging speed. Thus PSO has become one of the most attractive computational intelligence methods in recent years. However, the traditional PSO cannot be applied to discrete space optimization problems directly. In [15], a set-based PSO (S-PSO) has been developed to extend PSO in the discrete space. The method has been shown to be very promising in solving combinatorial optimization problems. The S-PSO is suitable for the cloud workflow scheduling problem due to the following reasons. First, the service instances available in cloud can be somewhat imaged as a resource set. Hence the problem of allocating services to workflows can be naturally represented using sets and thus the S-PSO can be directly applied. Second, the S-PSO enables the use of heuristics to accelerate search. Thus the existing effective heuristics for workflow scheduling can be integrated with the S-PSO to further improve performance. We apply the discrete version of the comprehensive learning PSO (CLPSO) [16] based on the S-PSO method to the cloud workflow scheduling problem. Experimental results show that the proposed method is promising.

The rest of this paper is organized as follows. In Section, II, the cloud workflow scheduling model is described. In Section III, a review on the PSO and S-PSO is made. In Section IV, the S-CLPSO approach is proposed. Experimental results are shown in Section V. Conclusion is drawn in Section VI.

II. PROBLEM DEFINITION

We model a workflow as a DAG G=(V,A). The set of nodes $V=\{T_1,T_2,\ldots,T_n\}$ corresponds to the tasks, where n is the total number of tasks. The set of arcs A represents precedence relations between the tasks. Each task of the workflow can be implemented by some service instances in the cloud system. We denote the set of available service instances for the task T_i as an implementation domain $S_i=\{s_i^1,s_i^2,\ldots,s_i^{m_i}\}$, where s_i^j $(j=1,2,\ldots,m_i)$ is a service instance available for T_i and m_i is the number of available service instances. For each service instance s_i^j, we consider three types of QoS parameters, i.e., the cost $s_i^j.c$, the execution time $s_i^j.t$, and the service's historical reliability $s_i^j.r$.

To provide a flexible way for users to manage the QoS performance of workflows in the cloud system, the considered cloud workflow scheduling model enables users to define various QoS constraints. With the above three QoS parameters, users can define three types of QoS constraints.

- Deadline constraint: the execution time of the workflow must be not larger than a user-defined variable *Deadline*.
- Budget constraint: the total cost of the service instances consumed by the workflow must be not larger than a user-defined variable *Budget*.
- Reliability constraint: the historical reliability of the service instances reserved for the workflow must be not smaller than a user-define variable *MinReliability*.

In addition, the cloud workflow scheduling model enables users to specify one QoS parameter as the optimization objective. In other words, there are three possible types of optimization objectives:

- Makespan minimization: the objective is to minimize the total execution time subject to the budget and reliability constraints specified by users.
- Cost minimization: the objective is to minimize the total cost subject to the deadline and reliability constraints specified by users.
- Reliability maximization: the objective is to maximize the expected reliability subject to the deadline and budget constraints specified by users.

III. A BRIEF REVIEW ON THE SET-BASED PSO

Particle swarm optimization (PSO) is a population-based stochastic optimization technique proposed by Kennedy and Eberhart in 1995. In PSO, each particle in the population maintains two vectors – a velocity vector and a position vector. During each generation, each particle updates its velocity and position by learning from the particle's own historically best position and the best position found by the entire swarm so far . More specifically, let us suppose that the swarm size is N. Each particle in the swarm maintains two vectors, i.e., the position $\boldsymbol{x_i}(x_i^1,x_i^2,\cdots,x_i^n)$ and the velocity $\boldsymbol{v_i}(v_i^1,v_i^2,\cdots,v_i^n)$. Here i=1, 2, …, N represents the ID of particles. The original PSO works by iteratively running the following two rules, i.e., the velocity updating rule

$$v_i^j \leftarrow \omega v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (gbest^j - x_i^j), \quad j=1,2,\ldots,n \tag{1}$$

and the position updating rule

$$x_i^j \leftarrow v_i^j + x_i^j, \quad j=1,2,\ldots,n \tag{2}$$

In (1), ω is a parameter of PSO named the inertia weight, c_1 and c_2 are acceleration coefficients, and r_1 and r_2 are random numbers uniformly distributed in (0, 1). $\boldsymbol{pbest_i}$

$(pbest_i^1, pbest_i^2, \cdots, pbest_i^n)$ is the historically best position of the *i*-th particle, and ***gbest***(***gbest***^{1}, ***gbest***^{2},…, ***gbest***^{n}) is the best-so-far position of the whole swarm.

Since the updating rules (1) and (2) are all defined on an *n*-dimensional real vector space, they cannot be applied to discrete space optimization problems directly. In order to extend PSO to the discrete space, in our previous work [15], we have developed a set-based PSO (S-PSO) method. According to Lin and Kernighan [17], many combinatorial optimization problems (COPs) can be represented by "find from a set E a subset X that satisfies some constraints Ω and optimizes the objective function *f*". Based on this idea, the S-PSO uses a set-based representation and redefines the operators in the updating rules (1) and (2) of PSO on the set space. In the representation scheme of S-PSO, a COP is described by the following characteristics:

- A universal set E of elements is given. The universal set E can be divided into an *n*-tuple $(E^1, E^2, ..., E^n)$, where $E = E^1 \cup E^2 \cup \cdots \cup E^n$. We can regard $E^1, E^2, ..., E^n$ as the *n* dimension of the problem.
- A candidate solution to the problem $X \in PS$ is an *n*-tuple $(X^1, X^2, ..., X^n)$, where $X^j (j = 1,2,...,n)$ is a set and $X^j \subseteq E^j$. PS is the set of all feasible solutions.
- X is feasible only if X satisfies the constraints Ω.
- The objective of the problem is to find a feasible solution X^* that optimizes the objective function *f*.

Based on this representation, the S-PSO redefines the operators in (1) and (2) in the set space as follows:

- A position X of particle is an *n*-tuple $X = (X^1, X^2, ..., X^n)$ which is just a solution to the problem, i.e., $X \in PS$. We denote the position of the *i*-th position as $X_i = (X_i^1, X_i^2, ..., X_i^n)$.
- A velocity V of particle is an *n*-tuple $V = (V^1, V^2, ..., V^n)$ where V^j is a set with possibilities defined on E^j. That is, $V^j = \{e / p(e) \mid e \in E^j\}$. We denote the velocity of the *i*-th position as $V_i = (V_i^1, V_i^2, ..., V_i^n)$
- The "coefficient×velocity" operator in (1) is defined as

$$cV = \{e / p'(e) \mid e \in E\}, \quad p'(e) = \begin{cases} 1, & \text{if } c \times p(e) > 1 \\ c \times p(e), & \text{otherwise} \end{cases} \tag{3}$$

- The "position-position" operator in (1) is defined as

$$A - B = \{e \mid e \in A \text{ and } e \notin B\} \tag{4}$$

- The "Coefficient×(Position–Position)" operator in (1) is defined as

$$cE' = \{e / p'(e) \mid e \in E\}, \quad p'(e) = \begin{cases} 1, & \text{if } e \in E' \text{ and } c > 1 \\ c, & \text{if } e \in E' \text{ and } 0 \le c \le 1 \\ 0 & \text{if } e \notin E' \end{cases} \tag{5}$$

- The "Velocity+Velocity" operator in (1) is defined as

$$V_1 + V_2 = \{e / \max(p_1(e), p_2(e)) \mid e \in E\} \tag{6}$$

- The position updating rule in (2) is processed by the following steps.

Step 1) Each dimension of the velocity V_i^j is converted into a crisp set $cut_\alpha(V_i^j) = \{e \mid e / p(e) \in V_i^j \text{ and } p(e) \ge \alpha\}$.

Step 2) The *i*-th particle begins to build a new position. Beginning with an empty set, the particle first selects feasible elements from $cut_\alpha(V_i^j)$ to add to the new position.

Step 3) If there is no feasible element in $cut_\alpha(V_i^j)$ and the construction is not complete, the particle reuses the elements in its previous position X_i^j to build the new position.

Step 4) If the construction is still not complete but there is no feasible element in the previous position X_i^j, the particle uses other feasible elements to finally build a complete solution.

In this way, the operators in (1) and (2) can all be redefined in the set space. Thus the S-PSO method can be applied to COPs. For more details of the S-PSO method, please refer to [15]. Based on the S-PSO, different improved PSO variants can also be extended into their discrete versions. Since the CLPSO [16] has been found to be a very promising algorithm for complex multimodal optimization problems, this paper takes advantage of the CLPSO and uses its discrete version based on the S-PSO method to solve the cloud workflow scheduling problem. We denote this discrete version of CLPSO as S-CLPSO. Its velocity updating rule is

$$v_i^j \leftarrow \omega \cdot v_i^j + cr^j (pbest_{f_i(j)}^j - x_i^j), \quad j = 1,2,...,n \tag{7}$$

where c is a parameter, r^j is a random number in [0,1], and $pbest_{f_i(j)}^j$ means the j^{th} dimension of the ***pbest*** position of the particle $f_i(j)$. $f_i(j)$ is given as follows. First a random number $ran \in [0,1]$ is generated. If *ran* is larger than a parameter *Pc*, then $f_i(j)=i$. Otherwise, the algorithm applies the tournament selection to two randomly selected particles. The particle with better fitness value is selected as $f_i(j)$.

IV. THE S-CLPSO APPROACH

In this section, the S-CLPSO is applied to the cloud workflow scheduling problem with various QoS constraints.

A. Representation Scheme

To find a solution to the cloud workflow scheduling problem, we have to find a schedule $K = \{K_1, K_2, ..., K_n\}$, where K_i means the task T_i is mapped to the service instance $s_i^{K_i}$ to implement. According to the representation scheme of the S-PSO method described in the Section III, the search space of the cloud workflow scheduling problem can be described as follows:

- The universal set E is the union of the implementation domains of all tasks, i.e., $E=\bigcup_{i=1}^{n} S_i$. E can be divided into an n-tuple (E^1, E^2,…,E^n)，where $E^i = S_i$.

- The solution to the Cloud workflow scheduling problem is an n-tuple X=(X^1, X^2, …, X^n), where $X^i \subseteq S_i$. In fact, since each task can only be assigned to one service instance, X^i only contains a single element K_i, which means T_i is assigned to $s_i^{K_i}$.
- X is a feasible solution if and only if X satisfies the constraints Ω , that is, X=(X^1, X^2, …, X^n) satisfies the precedence constraints and the reliability, deadline and budget constraints specified by the users.
- The objective of the problem is to find an optimal solution X^* that optimizes the user-preferred QoS criterion.

Based on the above presentation scheme, we define the position of a particle as an n-tuple (X^1, X^2, …, X^n) where $X^i \subseteq S_i$. The velocity of a particle is defined as an n-tuple V=(V^1, V^2, …, V^n). where V^i is a set with possibilities defined on the set S_i . In this way, we can apply the S-PSO to the Cloud workflow scheduling problem.

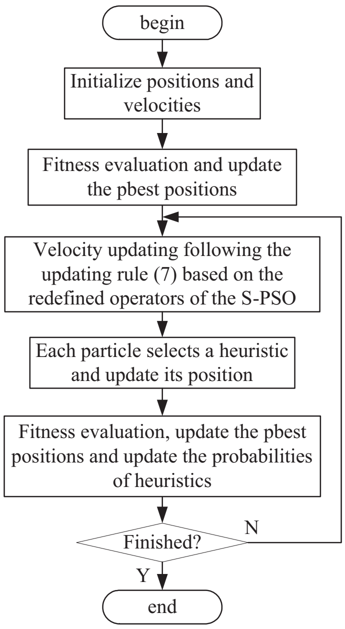

Figure 1. Flowchart of the S-CLPSO algorithm for the cloud workflow scheduling problem

B. Procedure of the Algorithm

Based on the above presentation scheme, we also applied the discrete version of CLPSO, i.e., the S-CLPSO algorithm to solve the cloud workflow scheduling problem. The flowchart of the S-CLPSO is given in Fig. 1.

1) Initialization

At the beginning the initial positions of the M particles are randomly initialized with M feasible solutions. The velocities are initialized as follows. For each dimension V^j of the velocity, the algorithm randomly selects an element from S_j and assigns a random possibility distributed in (0, 1] to the element. The possibilities of all other unselected elements are 0.

2) Velocity Updating

During each iteration of the S-CLPSO algorithm, each particle follows the velocity updating rule of CLPSO (7) to update its velocity. All the operators in the formula redefined based on the methods descried in Section III.

3) Position Updating

After updating velocities, each particle follows the position updating process given in Section III to update the positions. In other words, First, each dimension V_i^j of the velocity is converted into a crisp set $cut_\alpha(V_i^j)$. Then the particle first learns from the set $cut_\alpha(V_i^j)$, then the previous position X_i^j and finally other feasible elements to build a new position.

4) Fitness Evaluation

As the cloud workflow scheduling model takes various QoS constraints into account, in order to evaluate the performance the schedules generated by the particles, we design a fitness function with penalties.

For the reliability optimization problem, the fitness function is

$$f(K)=\begin{cases} 0.5\cdot\dfrac{Deadline}{K.makespan}+0.5\cdot\dfrac{Budget}{K.cost}+\dfrac{min\_Reliability}{max\_Reliability}, & \text{if } K.cost > Budget \text{ and } K.makespan > Deadline \\ 0.5+0.5\cdot\dfrac{Budget}{K.cost}+\dfrac{min\_Reliability}{max\_Reliability}, & \text{if } K.cost > Budget \text{ and } K.makespan \le Deadline \\ 0.5\cdot\dfrac{Deadline}{K.makespan}+0.5+\dfrac{min\_Reliability}{max\_Reliability}, & \text{if } K.cost \le Budget \text{ and } K.makespan > Deadline \\ 1+\dfrac{K.reliability}{max\_Reliability}, & \text{if } K.cost \le Budget \text{ and } K.makespan \le Deadline \end{cases} \tag{8}$$

In (8), $min\_Reliability$ ($max\_Reliability$) is the minimal (maximal) reliability of all service instances. The fitness of K is composed of two parts: penalties of QoS constraints and quality of the preferred QoS parameter. The score for each part is a value between (0,1], so the value of $f(K)$ is limited to the interval of (0,2]. If K satisfies all the QoS constraints, its fitness for the QoS constraints is 1, and the fitness for the user-preferred QoS parameter is set according to the reliability of K. On the other hand, if K fails to satisfy all the QoS constraints, its fitness for the QoS constraints is set according to the degree of satisfaction, and the score for the preferred QoS parameter is set to a minimum value.

In the makespan optimization problem, the fitness function is

$$f(K)=\begin{cases} 1+\dfrac{min\_Makespan}{K.makespan}, & \text{if } K.cost \le Budget \\ \dfrac{Budget}{K.cost}+\dfrac{min\_Makespan}{max\_Makespan}, & \text{if } K.cost > Budget \end{cases} \tag{9}$$

Here *min_Makespan* (*max_Makespan*) is the minimal (maximal) makespan of the workflow which can be estimated by assigning each task to the service instance with the shortest (longest) execution time.

In the cost optimization problem, the fitness function is

$$f(K)=\begin{cases}1+\dfrac{min\_Cost}{K.cost}, & \text{if } K.makespan \le Deadline\\ \dfrac{Deadline}{K.makespan}+\dfrac{min\_Cost}{max\_Cost}, & \text{if } K.makespan > Deadline\end{cases} \quad (10)$$

Here *min_Cost* (*max_Cost*) is the minimal (maximal) cost of the workflow which can be estimated by assigning each task to the service instance with the highest (lowest) cost.

Using the above fitness functions, the S-CLPSO can be used to deal with the cloud workflow scheduling problems with different QoS constraints.

C. Heuristic Information

To address different QoS factors like reliability, time and cost, seven heuristics are applied to integrate with the S-CLPSO approach. The integration is based on heuristic-based selection operators. In the steps 2), 3) and 4) in the position updating rule, the particle first selects from the set $cut_\alpha(V_i^j)$, then the previous position X_i^j, and finally the other feasible elements to build a new position. By using heuristic-based selection operators in this three steps, the S-CLPSO algorithm can further use problem-based information to accelerate the search. The seven heuristics used in the proposed approach is the reliability greedy (RG) heuristic, the time greedy (TG) heuristic, the cost greedy (CG) heuristic, the suggested deadline (SD) heuristic, the suggested budget (SB) heuristic, the time/cost (TC) heuristic, and the overall performance (OP) heuristic. For the details of these heuristics, please refer to the reference [13].

In order to further improve the utility of the heuristics, we design an adaptive strategy for selecting suitable heuristics. In the adaptive strategy, each heuristic is associated with a probability. We denote these probabilities as *P*(RG), *P*(TG), …, *P*(OP). At the beginning, the probabilities of the heuristics are the same, i.e., *P*(RG)=*P*(TG)=…=*P*(OP)=1/7. During each iteration of the algorithm, if a particle using a certain heuristic (e.g., the RG heuristic) successfully improves the best-so-far solution, we set *P*(RG)←*P*(RG)+0.02. Otherwise, we set *P*(RG) ←*P*(RG)−0.02. After the probabilities of all the heuristics are updated, we normalized the probabilities to guarantee that the sum of the probabilities equals to one. In this way, particles in the algorithm can choose suitable heuristics adaptively so that the efficiency of the heuristic-based selection operator can be improved.

V. EXPERIMENTAL RESULTS

A. Test Instances

We test the S-CLPSO algorithm on 10 workflow applications. The basic information of these workflow applications is shown in Table I. The first three workflows, including the e-Economic application, the neuro-science application (fMRI) [18], and the e-protein workflow [19], are derived from real-life applications. The other 7 workflows are generated based on the networks in PSPLIB [20]. These networks include j301_1 with 30 tasks, j601_1 and j601_2 with 60 tasks, j901_1 and j901_2 with 90 tasks, and j1201_1 and j1201_2 with 120 tasks. In our experiment, we randomly assign 6 to 10 service instances to each task in the workflows. The QoS parameters (reliability, time, and costs) of all service instances are randomly generated, but they follow the rule that for the same task a service instance with higher reliability or shorter execution time may cost more money, and vice versa.

TABLE I. TEST INSTANCES

Instance Name	Number of Tasks	Network
e-Economic	9	Fig. 5-3(a)
fMRI	15	Fig. 5-3(b)
e-Protein	15	Fig. 5-3(c)
j301_1	30	j301_1 (PSPLIB)
j 601_1	60	j 601_1 (PSPLIB)
j 601_2	60	j 601_2 (PSPLIB)
j 901_1	90	j 901_1 (PSPLIB)
j 901_2	90	j 901_2 (PSPLIB)
j 1201_1	120	j 1201_1 (PSPLIB)
j 1201_2	120	j 1201_2 (PSPLIB)

B. Comparison Results

We compare the S-CLPSO with the existing approaches including the deadline-based Markov Decision Process (Deadline-MDP) [12] and the ant colony system (ACS) algorithm [13]. The Deadline-MDP and the ACS approaches are by now the only algorithms that can deal with various QoS constraints. Thereinto, the Deadline-MDP can only deal with the problem with the deadline constraint and the objective of cost minimization.

As the Deadline-MDP is a deterministic algorithm, it yields the same result in every run. The results obtained by the Deadline-MDP are shown in the column of "MDP" in Table II. The ACS and the proposed S-CLPSO are stochastic algorithms. Therefore, we execute the algorithms for 100 independent runs on each instance. The three rows of data given in the columns "ACS" and "S-CLPSO" in each instance are the mean, best and worst results found by the ACS and the S-CLPSO respectively. In order to compare these algorithms systematically, we set five different levels of constraint environments on each workflow application. For example, on the e-Economic instance, we set five different deadline constraints {95, 115, 135, 155, 175} seconds. By setting the deadline constraint to 95, the deadline constraint becomes very tight. In a tight constraint environment, the problem of finding a feasible solution to meet the user-defined QoS constraints is already very difficult. In this case, the algorithm needs to address both the issues of makespan minimization and cost minimization. On the other hand, by setting the deadline constraint to 175, the constraint is loose and thus the algorithm can only focus on optimizing the preferred QoS parameter.

According to the results reported in Table II, even the worst solutions found by the S-CLPSO are better than those found by the Deadline-MDP algorithm in most instances. Compared with the ACS, S-CLPSO finds better mean results on 34 cases out of the 50 test cases. In addition, in the more difficult test cases with tight constraints (i.e., the left two columns of cases in Table II), S-CLPSO obtains better mean results on 19 out of the 20 cases. In contrast, in the test cases with soft constraints (i.e., the right two columns of cases in Table II), ACS finds better mean results on 12 out of the 20 cases. Overall, as the proposed S-CLPSO outperforms the ACS on the more difficult cases with tight constraints, these results demonstrate that the proposed S-CLPSO is promising.

TABLE II. COMPARISON RESULTS

		ACS	S-CLPSO	ACS	S-CLPSO	ACS	S-CLPSO	ACS	S-CLPSO	ACS	S-CLPSO
e-Economic 200 iterations	constraint	95		115		135		155		175	
	MDP	2296		2152		2081		1986		1864	
	result	2269.1	**2236.1**	2122.9	**2092.55**	1951.7	**1943.2**	1845.8	**1816.4**	1739.1	**1719.6**
		2233	**2233**	**2079**	**2079**	1933	**1925**	**1803**	**1803**	**1704**	**1704**
		2355	**2247**	2188	**2103**	2045	**1960**	1895	**1842**	1806	**1752**
fMRI 200 iterations	constraint	140		160		180		200		220	
	MDP	3966		3907		3787		3506		3340	
	result	3964.6	**3933.2**	3778.8	**3722.3**	3628.1	**3556.6**	3456.2	**3420.3**	3318.3	**3294.7**
		3922	**3922**	**3705**	**3705**	**3537**	**3537**	**3409**	**3409**	**3287**	**3287**
		4031	**3965**	3916	**3762**	3711	**3587**	3683	**3447**	3403	**3313**
e-Protein 200 iterations	constraint	160		180		200		220		240	
	MDP	4275		4129		3973		3769		3714	
	result	3983.6	**3940.9**	3809.5	**3780.45**	3678.8	**3634.7**	**3522.2**	3532.6	3460.9	**3460.3**
		3909	**3909**	**3727**	3751	**3605**	**3605**	**3514**	**3514**	**3445**	**3445**
		4119	**3983**	3909	**3814**	3841	**3687**	3616	**3586**	3514	**3481**
j301_1 300 iterations	constraint	245		275		305		335		365	
	MDP	9348		8953		8742		8425		8219	
	result	8394.9	**8381.9**	7872.5	**7842.5**	**7499.5**	7503.4	7291.2	**7282.8**	**7124.22**	7158.3
		8244	**8217**	**7748**	**7748**	**7387**	7425	**7190**	**7190**	**7061**	7109
		8606	**8442**	7983	**7919**	7727	**7568**	7426	**7346**	**7203**	7208
j601_1 600 iterations	constraint	280		310		340		370		400	
	MDP	16531		16134		15849		15621		15805	
	result	**14561.1**	14635.4	14341.0	**14292.8**	**14019.3**	14090.5	**13817.9**	13906	13620.0	**13599.6**
		14341	14521	14173	**14168**	**13843**	14035	**13642**	13828	13466	**13398**
		14781	**14731**	14573	**14383**	14216	**14165**	14092	**13984**	13778	**13697**
j601_2 600 iterations	constraint	300		340		380		420		460	
	MDP	16694		15904		15415		15165		14963	
	result	15189.2	**15146.7**	14521.5	**14516.9**	14010.8	**14002**	**13636.9**	13709.3	**13338.1**	13485.9
		14929	**14914**	**14273**	14347	**13837**	13945	**13471**	13606	**13193**	13387
		15473	**15357**	14814	**14627**	14332	**14146**	**13822**	13838	**13542**	13579
j901_1 900 iterations	constraint	230		270		310		350		390	
	MDP	24731		23764		22842		21905		20804	
	result	23705.8	**23580.3**	22211.1	**22105.2**	21132.4	**21059.7**	**20198.3**	20474.5	**19619.1**	19934.8
		23343	**23310**	**21791**	21957	**20886**	21028	**19948**	20286	**19401**	19777
		24465	**24094**	22260	**22006**	21491	**21274**	**20519**	**20519**	**19903**	20058
j901_2 900 iterations	constraint	365		415		465		515		565	
	MDP	25629		24614		23763		23098		22389	
	result	23183.4	**23051.4**	22442.7	**22276.3**	21882.9	**21799.8**	**21432.5**	21546.5	**20915.4**	21147.8
		22926	**22872**	**22163**	22176	**21599**	21658	**21186**	21419	**20699**	21086
		23558	**23350**	22792	**22503**	**22198**	22206	**21613**	21670	**21209**	21236
j1201_1 1200 iterations	constraint	410		455		500		545		590	
	MDP	34228		33303		32529		31778		31019	
	result	29396.6	**29288.5**	28888.1	**28821.4**	28499.6	**28445.2**	**28204.9**	28490.6	**27931.6**	28235.4
		29055	**29016**	**28636**	28657	**28167**	28239	**27890**	28339	**27675**	28123
		29756	**29437**	29168	**29152**	28769	**28547**	**28473**	28713	**28268**	28400
j1201_2 1200 iterations	constraint	410		470		530		590		650	
	MDP	34366		33169		31949		31035		29962	
	result	30787.4	**30167.9**	29625.9	**29525**	**28809.2**	28890.9	28125.8	**28122.2**	**27696.8**	27952.4
		30266	**29968**	**29350**	29463	**28343**	29011	**27825**	27926	**27472**	27696
		31319	**31090**	29980	**29937**	**29273**	29339	28455	**28248**	**27941**	28078

VI. CONCLUSION

In this chapter, A S-CLPSO approach has been designed for the cloud workflow scheduling problem. In order to satisfy the need of workflow management in cloud computing systems, the scheduling model considered in this paper provides a flexible way for users to define various QoS constraints and specify a QoS optimization preference. To address these QoS constraints, the proposed S-CLPSO defines penalty-based objective functions and uses an adaptive scheme to control the use of seven heuristics. Experimental results show that the proposed approach is very competitive especially on the instances with tight QoS constraints.

REFERENCES

[1] N. W. Paton, M. A. T. de Aragão, K. Lee, A. A. A. Fernandes, and R. Sakellariou, "Optimizing utility in cloud computing through autonomic workload Execution," *IEEE Data Engineering Bulletin*, vol. 32, no. 1, pp. 51-58, 2009.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities," *The 10th International Conference on High Performance Computing and Communications*, 2008.

[3] A. Aboulnaga, K. Salem, A. A. Soror, W. F. Minhas, P. Kokosielis, and S. Kamath, "Deploying database applications in the cloud," *IEEE Data Engineering Bulletin*, vol. 32, no. 1, pp. 13-20, 2009.

[4] C. Vecchiola, S. Pandey and R. Buyya, "High-performance cloud computing: a view of scientific applications," The 10th International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 4-16, 2009.

[5] D. de Oliveira, F. A. Baião, and M. Mattoso, "Towards a taxonomy for cloud computing from an e-Science perspective," N. Antonopoulos and L. gillam (eds.), *Cloud Computing: Principles, Systems and Applications, Computer Communications and Networks*, pp. 47-62, 2010

[6] D. Kyriazis, et al., "An innovative workflow mapping mechanism for grids in the frame of quality of service", *Future Generation Computer Systems*, vol. 24, no. 6, pp. 498-511, 2008.

[7] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keaheu, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," the 4th IEEE International Conference on eScience, 2008.

[8] H. Topcuoglu, S. Hariri, M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, 2002, pp. 260-274.

[9] G. Malewicz, I. Foster, A. L. Rosenberg and M. Wilde, "A Tool for prioritizing DAGMan Jobs and its Evaluation," Journal of Grid Computing, vol. 5, no. 2, pp. 197-212, 2007.

[10] B. Cheng, "Hierarchical cloud service workflow scheduling optimization schema using heuristic generic algorithm," *Przeglad Elektrotechniczny*, pp. 92-95, 2012.

[11] Z. Wu, Z. Ni, L, Gu, and X. Liu, "A revised discrete particle swarm optimization for cloud workflow scheduling," *International Conference on Computational Intelligence and Security*, 2010.

[12] J. Yu, R. Buyya, and C.K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids", *Proceedings of the 1st International Conference on e-Science and Grid Computing (e-Science' 05)*, pp. 140-147, 2005.

[13] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a Grid workflow scheduling problem with various QoS requirements", *IEEE Transactions on System, Man, and Cybernetics, Part C*, vol. 39, no. 1, pp. 29-43, 2009.

[14] J. Kennedy, R. C. Eberhart, "Particle swarm optimization", *IEEE International Conference on Neural Networks*, 1995, 1942-1948

[15] W.-N. Chen, Jun Zhang, Henry Chung, Wen-liang Zhong, Wei-Gang Wu and Yu-hui Shi, "A novel set-based particle swarm optimization method for discrete optimization problem," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 278-300, 2010

[16] J.J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimization for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281-295, 2006

[17] S. Lin, B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2.m pp. 498-516, 1973

[18] Zhao Z, et al. Grid middleware services for virtual data discovery Composition, and integration (C). //Proc 2nd Workshop on Middleware for Grid Computing, 2004

[19] O'Brien A, Newhouse S, Darlington J. Mapping of scientific workflow within the e-Protein project to distributed resources (C). //Proc UK e-Science All Hands Meeting, 2004, Nottingham, UK

[20] Kolisch R, Sprecher A. PSPLIB – A project scheduling problem library: OR Software – ORSEP Operations Research Software Exchange Program (J). European Journal of Operational Research, 1997, 96(1): 205-216