An Ant Colony Optimization Algorithm for the Time-varying Workflow Scheduling Problem in Grids

Wei-neng Chen, *Student Member IEEE,* Yuan Shi and Jun Zhang, *Senior Member IEEE*
Deparment of Computer Science, SUN Yat-sen Universiy

***Abstract*—Grid workflow scheduling problem has been a research focus in grid computing in recent years. Various deterministic or meta-heuristic scheduling approaches have been proposed to solve this NP-complete problem. These existing algorithms, however, are not suitable to tackle a class of workflows, namely the time-varying workflow, in which the topologies change over time. In this paper, we propose an ant colony optimization (ACO) approach to tackle such kind of scheduling problems. The algorithm evaluates the overall performance of a schedule by tracing the sequence of its topologies in a period. Moreover, integrated pheromone information is designed to balance the workflow's cost and makespan. In the case study, a 9-task grid workflow with four topologies is used to test our approach. Experimental results demonstrate the effectiveness and robustness of the proposed algorithm.**

Index terms: grid computing, time-varying workflow, scheduling problem, ant colony optimization (ACO)

I. INTRODUCTION

Grid computing, analogous to the pervasive electrical power grid [1], enables resource sharing and cooperative work among distributed computational sites. It has been increasingly regarded as a promising computing platform which satisfies the need of various computation-intensive problems in science and business [2][3]. Recently, the open grid services architecture (OSGA) [4] introduces web services into grids, making service-oriented computing a popular application model [1][5]. The OSGA reinforces the grid technologies by bringing more abundant computational resources to grids.

In grid environment, applications are often described as workflows. A workflow is composed of atomic tasks that are processed in specific order to fulfill a complicated goal. Generally, grid workflows require more intensive computing and process larger data, compared with traditional workflows. Therefore, the performance of grid workflows becomes a critical issue of the workflow management systems. The development of the OSGA, however, makes the workflow management more intractable. In the new architecture of OSGA, a task can be executed by any one of a set of service instances provided by different grid service providers (GSPs). One of the most challenging problems is to map each task to a corresponding service instance to achieve the customers' quality of service (QoS) requirements as well as to accomplish high performance of the workflow. This problem is found to be NP-complete. Under the OSGA, the workflow scheduler has to balance several QoS requirements, including makespan and cost. Consequently, many traditional workflow scheduling algorithms, such as Opportunistic Load Balancing, Minimum Completion Time [6], Min-min [6]-[10], Max-min [6]-[10] and Duplex [6], are not suitable since they only tackle the makespan requirement.

In recent years, a number of researches have been focused on scheduling problem involved more than one QoS requirements. Literature [11] proposed a grid workflow scheduling algorithm in which cost is optimized with the expectation to minimize the makespan. Literature [12] presented a scheduling approach for the economics-driven grids to optimize the cost under the deadline constraint. In [13][14], a mixed-integer non-linear programming algorithm was introduced to optimize the cost with the consideration of other QoS requirements. As the scale of workflow applications becomes larger and larger, conventional deterministic approaches may fail to give a satisfying solution. Therefore, meta-heuristic algorithms have been receiving growing interests due to their powerful global search capability. In [15], a hybrid particle swarm optimization algorithm was developed to efficiently schedule the grid workflow. We also proposed an ant colony optimization approach to handle various QoS requirements and obtained very promising results [16]. The above algorithms are effective yet limited for a class of workflows whose topology is unchangeable. In real-world applications, there exists another class of workflows, named the time-varying workflow, in which the topology changes over time in a period. For example, some scientific workflows in grids apply different topologies in different computational phases. Also, plenty of real-world business workflows have time-varying topologies to perform long-term business processes. To tackle the time-varying workflow, a scheduler needs to give an optimal schedule considering its overall performance in a period.

This paper addresses the time-varying workflow scheduling problem in grids, aiming to minimize the total cost in a period as well as to meet the deadline constraint. To solve this problem, we propose an effective approach based on the ant colony optimization (ACO) [17][18]. ACO, inspired by the cooperative foraging behavior of real ants, is

This work was supported by NSFC Joint Fund with Guangdong, Key Project,No. U0835002, NSF of China Project No.60573066 and the Scientific Research,Foundation for the Returned Overseas Chinese Scholars, State Education Ministry,P.R. China.

Authors are with the Department of Computer Science, SUN Yat-sen University, Guangzhou, P.R.China, (Jun Zhang is the corresponding author, Email:junzhang@ieee.org).

978-1-4244-2959-2/09/$25.00 ©

a meta-heuristic approach for combinatorial optimization problems. ACO algorithms have been successfully applied to many real world application problems [19]-[23]. In this paper, we use one of the best ACO algorithms - the ant colony system (ACS) [24].

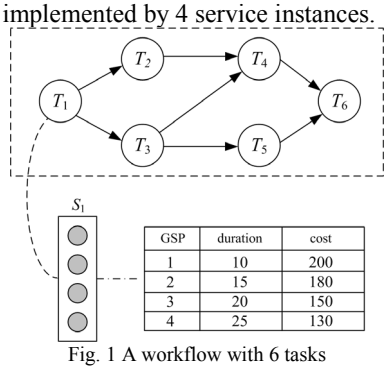
The proposed algorithm evaluates the quality of a schedule by tracing its possible topologies in a period. In addition, integrated heuristic information is designed to balance the objective and the constraint. Furthermore, the algorithm is tested on a time-varying workflow application to verify its effectiveness.

This paper is organized as follows. Section II describes the time-varying grid workflow scheduling problem. Section III gives a brief introduction of ACO. Section IV proposes ACO for the addressed problem. A case study is provided in Section V and the conclusion is finally presented in Section VI.

II. PROBLEM DESCRIPTION

A grid workflow can be modeled as a directed acyclic graph (DAG): $G = (V,A)$. The set of nodes $V = \{T_1,T_2,\ldots,T_n\}$ represents the tasks in the workflow while the set of arcs denotes precedence constraints between tasks. An arc is in the form of (T_i, T_j) where T_i is called the parent task of T_j, and T_j is the child task of T_i. We assume that a child task cannot be executed until all of its parent tasks have been completed.

For each task T_i ($1 \le i \le n$) in the workflow, a set of candidate service instances constitute its implementation domain $S_i = \{s_i^1, s_i^2, \cdots, s_i^{m_i}\}$, where s_i^j $(1 \le j \le m_i)$ represents a service instance provided by a GSP, and m_i is the total number of available service instances for T_i. Every service instance has its own QoS parameters: the cost and duration. Therefore, the properties of a service instance s_i^j can be represented as a group of three variables ($s_i^j.g$, $s_i^j.t$, $s_i^j.c$), in which $s_i^j.g$ stands for the GSP of s_i^j s_i^j while $s_i^j.t$ and $s_i^j.c$ denote the duration and cost of s_i^j , respectively. Fig. 1 shows a workflow application with 6 tasks, in which the first task can be implemented by 4 service instances.



GSP	duration	cost
1	10	200
2	15	180
3	20	150
4	25	130

Fig. 1 A workflow with 6 tasks

In the time-varying workflow, a task can have two working statuses: active and inactive. Only when the task is active (i.e. working) can it affect the total cost and makespan of the workflow. A workflow topology is defined as a workflow structure composed of all active tasks. A time unit is a predefined time span in which the whole workflow is executed exactly once. A period is a large time span containing some time units.

Suppose in a period of P time units, the sequence of the topologies of the workflow is denoted as $\{\Phi_1, \Phi_2, \ldots, \Phi_\Gamma\}$, where Γ is the total number of possible topologies and Φ_i lasts p_i time units. Thus, we have:

$$P = \sum_{i=1}^{\Gamma} p_i \tag{1}$$

An illustrative example is given in Fig. 2.

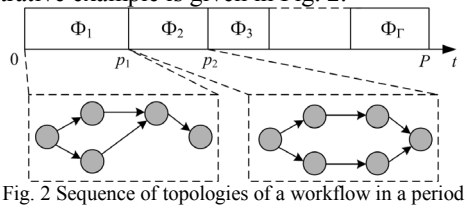


Fig. 2 Sequence of topologies of a workflow in a period

The function of the workflow scheduling algorithm is to map each task to a corresponding service instance to generate an optimal schedule. The goal of the algorithm is to minimize the total cost of the workflow in a period of P time units with its makespan in each time unit no larger than a user-defined variable *Deadline*.

III. ANT COLONY OPTIMIZATION

The basic idea behind the Ant Colony Optimization (ACO) is to simulate the foraging behavior of a colony of real ants. When searching for the food source, ants initially explore the area around their nest in a random way. Once the ants discover a path to food, they deposit a special chemical, called pheromone, on the path. By detecting pheromones on the ground, other ants can follow paths with richer pheromones. As this process continues, the pheromones on the shortest path accumulate quickly, causing most of the ants to choose that path. Apparently, the behavior of a single ant is relatively simple, but the cooperation among an ant colony can result in much more complex intelligence, which inspires the attractive computational paradigm – ACO.

```
Initialize
Loop
    Loop
        Each ant repeatedly applies a selection
          rule to construct a solution;
        Apply the local pheromone updating rule;
    Until all ants have built a complete solution
    Apply the global pheromone updating rule;
Until termination conditon
```

Fig. 3 Framework of the ant colony system (ACS)

In this paper, we apply the ant colony system (ACS) algorithm, one of the best ACO algorithms so far, to handle the time-varying workflow scheduling problem in grid environment. ACS employs a colony of artificial ants to search iteratively through the pheromone-deposit and pheromone-following processes. In every iteration, each ant constructs a solution in an incrementally way by repeatedly selecting one component for the solution based on the pheromones and the heuristic information associated with candidate components. While building a solution, an ant also modifies the amount of pheromones on its selected

components, which is called the local pheromone updating rule. Once all ants finish the solution construction, the pheromones are updated again according to the global pheromone updating rule. The framework of the ACS is shown in Fig. 3.

IV. ACO FOR THE SCHEDULING PROBLEM

The procedures of the ACO algorithm for the scheduling problem are coincident with the framework of ACS (Fig. 3). The details of the proposed algorithm are described as follows.

A. Heuristic Information Definition

In ACO, heuristic information is used to guide the search of artificial ants, which is found to be important for scheduling problems [25]. Here, we represent the heuristic information value of mapping service instance s_i^j to task T_i as η_{ij}.

η_{ij} is an integrated heuristic information value defined as the average value of two parts, aiming at balancing the objective and constraint of the scheduling problem:

$$\eta_{ij} = \frac{1}{2}(\eta_{ij}^{\ c} + \eta_{ij}^{\ t}) \tag{2}$$

where η_{ij}^c is the heuristic information value on cost while η_{ij}^t is the heuristic information value on execution time.

η_{ij}^c is given by:

$$\eta_{ij}^c = \frac{max\_cost_i - s_i^j.c + 1}{max\_cost_i - min\_cost_i + 1} \tag{3}$$

where $min\_cost_i = \min_{1\le j\le m_i}\{s_i^j.c\}$, and $max\_cost_i = \max_{1\le j\le m_i}\{s_i^j.c\}$. Therefore, a service instance with lower cost will be assigned a higher value and $\eta_{ij}^c \in (0,1]$.

η_{ij}^t is calculated by comparing the execution time of a service instance with the estimated sub deadline (ESD) of the corresponding task [16]. The ESD is obtained in the topology where all the tasks are active. Before calculating the ESD, we firstly evaluate the earliest start time (EST) and the backward earliest time (BEST) of each task. The EST is calculated by firstly mapping every task to the service instance with the shortest execution time. The EST of task T_i (denoted by EST_i) equals to the start time of T_i. Under this mapping strategy, the total makespan of the workflow can be viewed as the estimated minimum makespan of the workflow application. We denote this estimated value as *min_Makespan*. To calculate BEST, the DAG should be converted into a backward network by considering the starting node of the DAG as the ending node and vice versa, and reversing the direction of all directed arcs. For every T_i, the value of EST_i in the backward network is $BEST_i$.

Based on these two attributes, we can evaluate the average minimum execution time of task T_i from both forward traverse and backward traverse (denoted as $avg\_min\_time_i$) by

$$avg\_min\_time_i = \frac{\min_{\forall T_j \in pred(T_i)} BEST_j - BEST_i}{2} + \frac{\min_{\forall T_j \in pred(T_i)} BEST_j - BEST}{2} \tag{4}$$

The estimated sub deadline for task T_i (denoted as ESD_i) is obtained by enlarging the value of $avg\_min\_time_i$ on a scale of $Deadline/min\_Makespan$:

$$ESD_i = \left\lfloor avg\_min\_time_i \cdot \frac{Deadline}{min\_Makespan} \right\rfloor \tag{5}$$

η_{ij}^t is then calculated by:

$$\eta_{ij}^t = \begin{cases} 1, & \text{if } s_i^j.t \le ESD_i \\ \dfrac{MAT - (s_i^j.t - ESD_i) + 1}{MAT + 1}, & \text{otherwise} \end{cases} \tag{6}$$

$MAT = \max\{|max\_time_i - ESD_i|, |ESD_i - min\_time_i|\}$.

According to the above formula, service instances whose execution time is no larger than the ESD will be associated with the same value. The longer the execution time of a service instance exceeds the ESD, the less the heuristic value is assigned to the instance. η_{ij}^t is normalized to the interval (0,1]. Hence, the value of η_{ij} is also limited in (0,1].

B. Solution Construction

In every iteration of the algorithm, each ant incrementally builds a complete solution in *N* steps. While in each step, an ant selects a service instance for a task according to the pheromone and the heuristic information. The pheromone value of mapping service instance s_i^j to task T_i is denoted as τ_{ij}. The selection rule of mapping a service instance s_i^j to T_i is given by:

$$s_i^j = \begin{cases} \arg\max_{1\le j\le m_i} \tau_{ij}\beta^{\eta_{ij}}, & q \le q_0 \\ \text{Apply the roulette wheel scheme, othwewise} \end{cases} \tag{7}$$

where q is a random number uniformly distributed in [0,1], $q_0 \in [0,1]$ and $\beta \ge 1$ are two parameters. If $q \le q_0$, the service instance with the largest $\tau_{ij}\beta^{\eta_{ij}}$ is mapped to T_i. Otherwise, the roulette wheel scheme will be used, in which the probability of selecting S_i^j is in direct proportion to the value of $\tau_{ij}\beta^{\eta_{ij}}$. The parameter q_0 determines the relative importance of exploitation versus exploration while β determines the relative influence of pheromone versus heuristic information.

C. Evaluation of solution

The fitness of a schedule *K* is evaluated by considering its performance in the whole period. Since the workflow has several different topologies in the whole period, the total cost of schedule *K* is the sum of the cost in each topology. The cost in the *j*th topology, denoted by $cost_j$, can be calculated by the following equation:

$$cost_j = p_j \sum_{i=1}^{N} act(i,j) \cdot S_i^{K_i}.c \tag{8}$$

where *act*(*i,j*)indicates the working status of the *i*th task in the j^{th} topology. If the task is active, the value of *act*(*i,j*) is 1. Otherwise, the value of *act*(*i,j*) is 0.

Therefore, the cost of the schedule *K* is given by:

$$\begin{aligned} K.cost &= \sum_{j=1}^{\Gamma} cost_j \\ &= \sum_{j=1}^{\Gamma} p_j \cdot \sum_{i=1}^{N} act(i,j) \cdot s_i^{K_i}.c \end{aligned} \tag{9}$$

To satisfy the deadline constraint of the scheduling problem, we have to ensure that the maximum makespan of *K* (denoted by *K.max_makespan*) in all time units is no larger than the user-defined deadline. Mathematically, the constraint is represented by:

$$K.max\_makespan = \max_{1 \le i \le P} K.makespan_i \le Deadline \tag{10}$$

where $K.makespan_i$ is the makespan in i^{th} time unit.

The quality of schedule *K* is presented by its fitness value which contains two parts: deadline part and cost part. If the maximum makespan of *K* in all time slices satisfies the deadline constraint, the value for deadline is set to 1, and the value of the cost part is higher if the cost of *K* is lower. On the other hand, if schedule *K* fails to satisfy the deadline constraint, its value of the deadline part is set according to the degree of satisfaction and the value for cost is limited to a minimum value. Each part is in the interval of (0,1], so the fitness is a value between (0,2] and a better schedule results in a higher fitness. Mathematically, the fitness of *K* is calculated by:

$$K.fitness = \begin{cases} \dfrac{Deadline}{K.max\_makespan} + \dfrac{min\_total\_cost}{max\_total\_cost}, \\ \text{if } K.max\_makespan > Deadline \\ 1 + \dfrac{min\_total\_cost}{K.cost}, \\ \text{if } K.max\_makespan \le Deadline \end{cases} \tag{11}$$

where min_total_cost and max_total_cost are the minimum and maximum cost of the workflow in the period, respectively. They are obtained by the equations below:

$$\begin{aligned} min\_total\_cost &= \sum_{j=1}^{\Gamma} p_j \sum_{i=1}^{N} act(i,j) \cdot min\_cost_i \\ max\_total\_cost &= \sum_{j=1}^{\Gamma} p_j \sum_{i=1}^{N} act(i,j) \cdot max\_cost_i \end{aligned} \tag{12}$$

D. Pheromone Adjustment

1) Pheromone Initialization

All pheromone values are initially set to τ_0 that is the minimum pheromone value. τ_0 is determined according to the following equation:

$$\tau_0 = \frac{min\_total\_cost}{max\_total\_cost} \tag{13}$$

2) Local Pheromone Updating

In the process of solution construction, immediately after an ant has mapped a task T_i to a service instance s_i^j, the pheromone on s_i^j is updated according to the local pheromone updating rule:

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{14}$$

In the above equation, $\rho \in (0,1)$ is a pheromone decay parameter. As τ_0 is the minimum pheromone value, the local updating rule decreases the value of τ_{ij}, which enhances the diversity of the algorithm.

3) Global Pheromone Updating

In the ACS, the global pheromone updating rule is only applied to the best-so-far solution (solution with best fitness value). It increases the pheromone values associated with the best-so-far solution, making these good mappings more attractive in future iterations. If $K(K_1, \cdots, K_n)$ (task T_i is mapped to the service instance $s_i^{K_i}$) is the best-so-far solution, the updating rule is given by

$$\tau_{iK_i} = (1-\rho) \cdot \tau_{iK_i} + \rho \cdot K.fitness, \qquad i = 1,2,\cdots,n \tag{15}$$

In this equation, $\rho \in (0,1)$ is the same parameter as that in the local pheromone updating rule.

E. Implementation ACO for the scheduling problem

The implementation of ACO is illustrated below with the aid of the flowchart shown in Fig. 4.

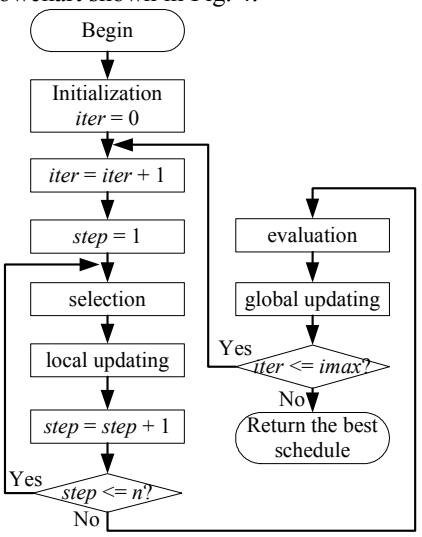


Fig. 4 Flowchart of ACO for time-varying grid workflow scheduling problem

1) Initialization: All pheromone values and parameters are initialized at the beginning of the algorithm.

2) Solution construction: *M* ants consequently build *M* solutions to the problem based on pheromone and heuristic values using the selection rule of the ACS.

3) Local pheromone updating: Soon after an ant maps a service instance s_i^j to task T_i, the corresponding pheromone value is updated by the local pheromone updating rule.

4) Evaluation: Each solution is evaluated according to its overall performance in the whole period.

5) Global pheromone updating: The pheromone values corresponding to the best-so-far solution are updated by the global pheromone updating rule.

6) Terminal test. If the number of iteration is less than a predetermined maximum number of iteration *Imax*, go to step 2) to start a new iteration. Otherwise, the algorithm is terminated.

V. CASE STUDY

We design a workflow case to test the effectiveness of the ACS. Fig. 5 shows the basic structure of the workflow which is based on the e-Economic application with 9 tasks. Assume that the running period of the workflow is 30 time units and the workflow executes exactly once in each time unit. All tasks except T_4 and T_5 are under working during the whole period. However, the working status of T_4 and T_5 are changeable and illustrated in Fig. 6. If the value of the working status of a task is 1, then the task is under work. Otherwise, it doesn't work. In our experiment, we randomly assign 6 to 10 service instances to each task in the workflow. The cost and duration of all service instances are also randomly generated, but they follow the rule that for the same task a service instance with shorter duration probably cost more money, and vice versa.

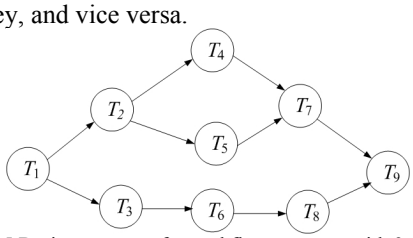


Fig. 5 Basic structure of a workflow test case with 9 tasks

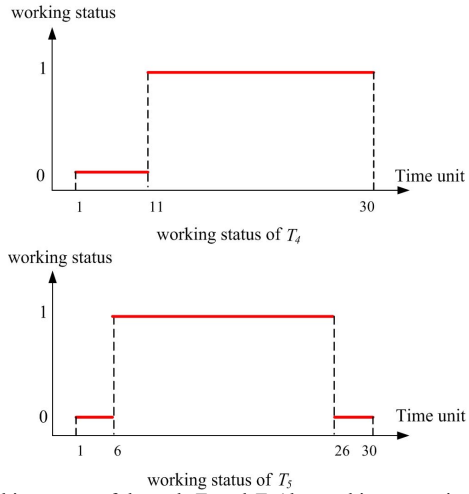


Fig. 6 working status of the task T_4 and T_5 (the working status is 1 when the task is working)

As can be observed from Fig. 6, the workflow goes through 4 topologies in 30 time units. Fig. 7 describes the possible topologies and their time span. The objective of ACO is to generate an optimal schedule which minimizes the total cost of the workflow in the period. In addition, the schedule should satisfy the deadline constraint for each time unit.

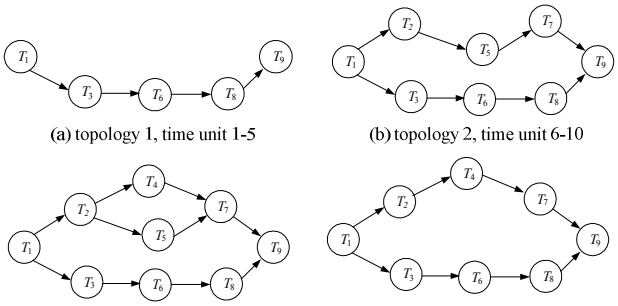


Fig. 7 Possible topologies and their time span

The parameters to be set are β and q_0 in the selection rule, ρ in the pheromone updating rule, the population size *POPSIZE* and the maximum number of iteration *Imax*. Table I lists the parameter settings for ACO.

Experiments are conducted with different levels of deadline constraint. The proposed algorithm is executed 100 times independently for each level of constraint. Table II presents the experimental results. As can be observed, ACO is capable of finding optimal or near-optimal schedules in different constraint conditions, demonstrating the effectiveness and robustness of the algorithm.

Table I Parameter settings of the ACO for the scheduling problem

Parameter	Value
β	1.2
ρ	0.1
q_0	0.5
POPSIZE	30
Imax	250

Table II Experimental Results with different levels of deadline constraint

Deadline Constraint	Total Cost		
	Average	Best	Worst
95	59060.2	59030	59115
115	55186.4	54655	55845
135	50934.7	50790	52125
155	47803.2	47785	48115
175	45280.3	45255	46165

VI. CONCLUSION

To tackle the time-varying workflow scheduling problem in grids, an ACO approach has been proposed. The major advancement of the algorithm is that it evaluates the overall performance of a schedule in a period, which fits the dynamic characteristics of the time-varying workflow. Additionally, integrated pheromone information is designed to balance the cost and makespan, which guides the efficient search of ACO. The algorithm is tested by a time-varying grid workflow with 4 topologies, showing that the proposed algorithm is an effective and robust approach.

REFERENCES

[1] I. Foster and C. Kesselman, *The grid: Blueprint for a future computing infrastructure*, San Mateo, CA: Morgan Kaufmann, 1999.

[2] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service oriented grid computing", *10th Heterogeneous Computing Workshop (HCW' 2001)*, 2001.

[3] F. Neubauer, A. Hoheisel, J. Geiler, "Workflow-based grid applications", *Future Generation Computer Systems*, Vol. 22, 2006, pp. 6-15.

[4] Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the grid – an open grid services architecture for distributed systems integration", *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[5] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Intl J. Supercomputer Applications*, 2001.

[6] T.D Braun, et al., "A comparison eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and distributed computing*, Vol. 61, 2001, pp. 810-837.

[7] H. XiaoShan, S. XiaoHe, "QoS guided min-min heuristic for grid task scheduling", *Journal of Comput. Sci. & Technol.*, Vol. 18, No. 4, 2003, pp. 442-451.

[8] A. Mandal, et al. "Scheduling strategies for mapping application workflows onto the grid", *in Proceedings of the 14th IEEE International Symposium on High Performance and Distributed Computing (HPDC-14)*, 2005, pp. 125-134.

[9] M. Maheswaran, et al., "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems", *Journal of Parallel and Distributed Computing*, Vol. 59, 1999, pp. 107-131.

[10] M.M. López, E. Heymann, M.A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems", *in Proceedings of the Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06), IEEE*, 2006.

[11] Y. Yuan, X. Li, and Q. Wang, "Time-cost trade-off dynamic scheduling algorithm for workflows in grids", *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design*, 2006.

[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] A. Afzal, A.S. McGough, J. Darlington, "Capacity planning and scheduling in grid computing environments", *Future Generation Computer Systems*, 2007, in press.

[14] A. Afzal, J. Darlington, A.S. McGough, "QoS-constrained stochastic workflow scheduling in enterprise and scientific grids", *The 7th IEEE/ACM International Conference on Grid Computing*, 2006, pp. 1-8.

[15] Chun-hua Hu, Min Wu, Guo-ping Liu and Wen Xie, "QoS Scheduling Algorithm Based on Hybrid Particle Swarm Optimization Strategy for Grid Workflow", *The 6th IEEE International Conference on Grid and Cooperative Computing*, 2007.

[16] Wei-neng Chen and Jun Zhang, "An ant colony optimization approach to a Grid workflow scheduling problem with various QoS requirements", *IEEE Transaction on System, Man, and Cybernetics: Part C*, vol. 39, no. 1, pp. 29-43, 2009.

[17] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," appeared in *Proceedings of ECAL91 – European Conference on Artificial Life*, Elsevier publishing, pp. 134-142.

[18] Macro Dorigo, Vittorio Maniezzo and Alberto Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans. on systems man, and cybernetics - part B,* vol. 26, 1996, pp. 29-41.

[19] Yuan Zhang, Hua-Chun Cai, Ying Lin, Jing Xiao, and Jun Zhang, "An ant colony system salgorithm for the multicast routing problem," *Proceeding of the 3rd International Conference of Natural Computation 2007 (ICNC'07)*, Haiko, China, pp. 756-760, Aug. 2007.

[20] Xiao-Min Hu and Jun Zhang, "Orthogonal Methods Based Ant Colony Search for Solving Continuous Optimization Problems," *Journal of Computer Science and Technology*, vol. 23, no. 1, pp. 1-18, 2008.

[21] Xiao-Min Hu, Jun Zhang, Jing Xiao, and Yun Li, "Protein folding in hydrophobic-polar lattice model: a flexible ant-colony optimization approach," *Protein & Peptide Letters*, vol. 15, no. 5, pp. 469-477, 2008.

[22] Xiao-Min Hu, Jun Zhang, and Yun Li, "Flexible protein folding by ant colony optimization," T. G. Smolinski et al. (Eds.): *Comp. Intel. in Biomed. & Bioinform.*, SCI 151, Springer-Verlag Berlin Heidelberg, pp. 317-336, 2008.

[23] Jun Zhang, Xiao-Min Hu, Xuan Tan, Jing-hui Zhong, and Q. Huang, "Implementation of an Ant Colony Optimization Technique for Job Scheduling Problem," *Transactions of the Institute of Measurement and Control*, vol. 28, no. 1, pp. 93-108, 2006.

[24] Macro Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to TSP", *IEEE Trans. Evolutionary Computation*, vol. 1, 1997, pp. 53–66.

[25] Z. Shi, and J.J. Dongarra, "Scheduling workflow applications on processors with different capabilities", *Future Generation Computer Systems*, Vol. 22, 2006, pp. 665-675.