Workflow Scheduling in Grids: An Ant Colony Optimization Approach

Wei-Neng CHEN, Jun ZHANG, *IEEE Member* and Yang Yu

***Abstract*—Grid applications in virtue of open service grid architecture (OGSA) are promising next-generation computation techniques. One of the most important and challenging problems about grid application is the workflow scheduling problem to achieve the users' QoS (quality of service) requirements as well as to minimize the cost. This paper proposes an ant colony optimization (ACO) algorithm to tackle this problem. Several new features are introduced to the algorithm. First, we define two kinds of pheromone and three kinds of heuristic information to guide the search direction of ants for this bi-criteria problem. Each ant uses either one from these heuristic types and pheromone types in each iteration based on the probabilities controlled by two parameters. These two parameters are adaptively adjusted in the process of the algorithm. Second, we use the information of partial solutions to modify the bias of ants so that inferior choices will be ignored. Moreover, the experimental results in 3 workflow applications under different deadline constraints show that the performance of our algorithm is very promising, for it outperforms the Deadline-MDP algorithm in most cases.**

I. INTRODUCTION

Grid technologies are a promising next-generation computation platform that supports the sharing and coordinated use of diverse resource from geographically distributed components [1]. In the recent years, grid applications have been reinforced by the open grid services architecture (OGSA) [2]. OGSA introduces Web services into grid interoperability model and develops the idea of service-oriented grid computation. It builds an infrastructure that enables grid application users to share a reliable, secure, scaleable and distributed network grid environment.

Many grid applications can be described as workflows. The problem of workflow scheduling, which aims to map the services of a workflow to different resources in the grid environment, is crucial and challenging for grid computation. By now, some workflow application management systems with workflow scheduling algorithms have been proposed. For example, Pegasus [3][4], ASKALON [5], Condor [6][7], and the min-min [8], max-min, sufferage, HEFT, and random heuristics [9]. However, these algorithms mainly aim at minimizing the makespan (the completion time) of the workflow. Based on OGSA, a number of grid service providers (GSPs) emerge to provide different services in the grid market. Different GSPs may implement the same service by different policies, mechanisms, or structures, with different QoS (quality of service) levels, and charge the service users different price. The overall grid environment becomes an economic-driven [1][10] or market-driven [11] gird market. In general, GSPs charge higher price for high QoS services, and charge lower price for low QoS services. In this case, not only the QoS but also the cost must be considered by users. Therefore, the workflow scheduling algorithms considering both the makespan (which is one of the most important QoS parameters) and the cost are attracted for new-generation workflow management systems.

In this area, Yuan *et al*. [12] develops an extended critical activity (ECA) based dynamic workflow scheduling algorithm. The problem they considered is to minimize the cost for the schedule with minimum makespan. Their deterministic algorithm manages to find the best solution for this problem within polynomial-time. Yu *et al*. [13] propose a Deadline-MDP algorithm for a much more complex problem. The objective of their algorithm is to complete the workflow within deadline and minimize the execution cost of the workflow. As the problem is NP-hard with bi-criteria, Deadline-MDP is only a deterministic algorithm to approximate the best solution to the problem. The algorithm works by partitioning the workflow (described by a directed acyclic graph (DAG)) and assigning a sub-deadline to each partition. Moreover, a Markov Decision Process (MDP) is applied to yield the best choices for each partition.

In this paper, we tackle the same problem as [13], but develop a mataheuristic approach. We take advantage of ant colony optimization (ACO) [14][15] to solve this problem. ACO is proposed by Dorigo in the light of the foraging behavior of ants. ACO works by simulating the pheromone-depositing and pheromone-following behavior of ants, and has been successfully applied to various intractable combinatorial optimization problems[16][18].The algorithm proposed in this paper follows the rules of ant colony system (ACS) [17], which is one of the best ACO algorithms so far.

New features are proposed in our algorithm. First, in order to give attention to both objectives, namely cost and makespan, we define two kinds of pheromone and three kinds of heuristic information to guide the search direction of ants. Each ant selects either one from these heuristic types and pheromone types to guide its search behavior in each iteration based on the probabilities controlled by two parameters. An adaptive scheme is integrated in the algorithm to adjust the value of these two parameters. Second, we estimate the earliest start time and earliest end time of services in the partial solution constructed by ants. In terms of this information, the selection preference of ants controlled by pheromone and heuristic information is

This work was supported in part by NSF of China Project No.60573066 and NSF of Guangdong Project No. 5003346

Wei-Neng Chen, Jun ZHANG and Yang Yu are with Department of Computer Science, Sun-yat Sen University, Guangzhou, China

Jun Zhang is Corresponding author, e-mail: junzhang@ieee.org.

1-4244-1340-0/07/$25.00 ©2007 IEEE

modified, so that inferior choices will be ignored by ants. Moreover, we implement numerical experiments on three workflows: e-Economic, neuro-science [19], and e-Protein [20]. Experimental results under different deadline constraints show that the performance of our algorithm is promising. It outperforms the Deadline-MDP algorithm [13] in most cases. This demonstrates the effectiveness of our algorithm.

The paper is organized as follows. Section II describes the background of workflow application, workflow management, and workflow scheduling in grids. Section III formulates the workflow scheduling problem considered in this paper. Section IV proposes the ACS algorithm for workflow scheduling. Section V gives the instances we used in our experiments and the results of our experiment. The conclusion finally comes in section VI.

II. Architecture for Workflow Application and Management in Grids

The architecture for workflow application and management in grids based on the concept of service-oriented grid computation can be illustrated by Fig. 1. The general idea of the architecture is to map the tasks of a workflow into services. Services are defined as self-describing, open components which support rapid and low-cost distributed applications [21]. In virtue of service, different organizations, although geographically distributed, are able to provide these services based on different implementation policies, mechanism, and structures. These organizations are referred to as grid service providers (GSPs). GSPs charge different services by their QoS. Users only execute their services by GSPs that satisfy their QoS requirements, and only pay for what they use. The whole grid environment becomes an economic-driven market [1][10] that supports reliable, secure, scalable, and distributed applications.

More specific, a workflow application and management cycle in grids can be viewed as the interplay of the following procedures (Fig. 1).

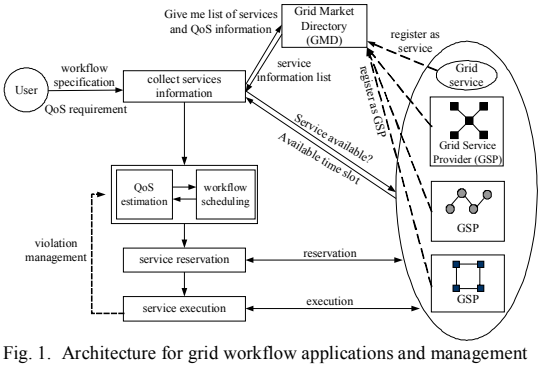

Fig. 1. Architecture for grid workflow applications and management

Step 1) GSPs register services to grid market directory (GMD). In grid market, GMD [22] is used to manage the information of all services. Usually, GMD records the type, the provider, the QoS parameters, the cost and other information of each service. Once an organization tends to promote services to the grid market, it first registers itself to GMD as a GSP. New services also have to be registered before they enter the market.

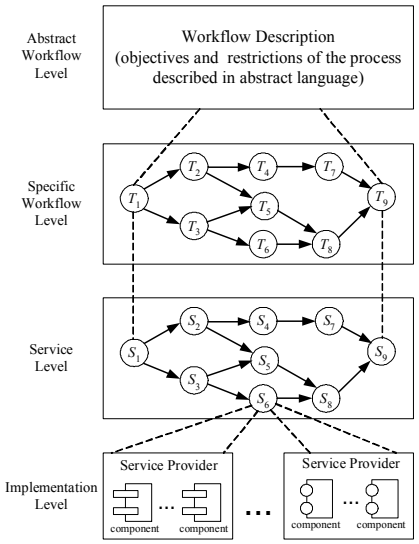

Fig. 2. Mapping of grid workflows to services under SOA: a four-level model

Step 2) Users define and submit the workflow application to workflow management system (WMS). Generally, definition and implementation of workflow applications are processed in four levels [3], which are shown in Fig. 2. Firstly in the abstract workflow level, the objective and requirements of the work are described in abstract language. This abstract description is further specified into a sequence of organized tasks in the specific workflow level. As only services, not tasks, are provided in market, in the service level, tasks of the workflow have to be mapped to corresponding service types. Then users are able to assign these services to GSPs and GSPs execute the services in the implementation level. According to this four-level model, users carry out the first three levels in this step. They submit the workflow with all tasks having been mapped to corresponding service types to WMS. They also submit their QoS requirements to WMS.

Step 3) As soon as WMS accepts the workflow applications, it enquires GMD for service information. Typically, it collects the type, provider, access directory, QoS parameters, and cost of each related services.

Step 4) WMS enquires each related GSPs to know whether the services will be available. The services without available acknowledgements will not be adopted.

Step 5) WMS executes a scheduling algorithm to decide the optimal allocation scheme for the workflow. The scheduler needs to assign each service to the right GSP and determines the execution time slot for each service. The goal of scheduling is to achieve the users' QoS requirements, typically the deadline constraints, as well as to minimize the execution cost. Fig. 1 reveals that the scheduler is composed of three modules. Scheduling module maintains a scheduling algorithm to generate feasible solutions. These solutions are sent to QoS estimation module, where the makespan, cost and other parameters of solutions are evaluated. QoS

parameter recorder module records some priori information, such as the service success rate of each GSP. This information is the summary of previous experience, recording the historical performance of each GSP, and is updated every time when an application is complete. These historical statistics are adopted as parameters in QoS estimation module to evaluate the expected performance of a schedule more reasonably and accurately.

Step 6) WMS accesses related GSPs to make reservations for all services according to the solution returned by step 5.

Step 7) The workflow is executed by GSPs in terms of the prearranged time. Information exchange between WMS and GSPs may occur.

Step 8) The contract between users and GSPs may be violated at times due to many reasons such as the failure of GSPs' resources. In this case, a service may be delayed or even become unavailable, which makes the following services fail to be executed in terms of the priori schedule. Therefore, a rescheduling mechanism is required for violation management. Rescheduling mechanism updates the schedule dynamically to adapt to run-time violations.

Step 9) After the completion of each service, the actual QoS performance of service is fed back to the QoS parameter recorder module so that the historical QoS statistics are updated.

III. Problem Formulation

We focus on the scheduling module in this paper. The scheduling problem involves a workflow application the services of which can be implemented by different GSPs. For the same service, GSPs charge higher price for short-makespan implementation and lower price for long-makespan implementation. The scheduling problem is to allocate each service to a GSP so that the workflow can be done within the users' deadline requirements and the cost is minimized. The scheduling model is formulated in this section in detail.

Generally, workflow applications can be modeled as a directed acyclic graph (DAG) $G=(V,A)$. Let n be the number of services in the workflow. The set of nodes $V=\{S_1,S_2,\cdots,S_n\}$ corresponds to the services of the workflow. The set of arcs A represents precedence relations. An arc is in the form of (S_i,S_j), where S_i is called the parent service of S_j, and S_j is the child service of S_i. Typically in a workflow, a child service cannot be executed until all of its parent services have been completed. The set of parent services of S_i is denoted as $Pred(S_i)$, and the set of child services is $Succ(S_i)$. Examples of workflow described by DAG are given by Fig. 3.

For the sake of convenience, we add a start node S_{start} and an end node S_{end} to the DAG. For all $S_i(1 \le i \le n)$, if $Pred(S_i)$ is empty, we add S_i to $Succ(S_{start})$, so that $Pred(S_i)=\{S_{start}\}$. Similarly, if $Succ(S_i)$ is empty, we add S_i to $Pred(S_{end})$, so that $\mathrm{Succ}(S_i)=\{S_{end}\}$.

Each service $S_i(1 \le i \le n)$ has an implementation domain $SP_i = \{sp_i^1, sp_i^2, \cdots, sp_i^{m_i}\}$, where sp_i^j $(1 \le j \le m_i)$ represents the service implementations provided by different GSPs, and m_i is the total number of available service implantations for S_i. Additionally, we denote the total processing time (duration) of sp_i^j as d_i^j, and the cost of sp_i^j is c_i^j.

The objective function of the scheduling problem is to find an optimal schedule $\{K_1,\cdots,K_n\}$, which means S_i is executed by $sp_i^{K_i}$ $(1 \le i \le n)$, so that the total cost of the workflow is minimized, as described by (1).

$$\text{minimize } cost = \sum_{i=1}^{n} c_i^{K_i} \tag{1}$$

Moreover, the end time of the whole workflow must be not later than D. D is the deadline constraint required by users.

IV. ACO Algorithm for the Scheduling Problem

The general idea of ant colony optimization (ACO) is to simulate the foraging behavior of ant colonies. When a group of ants set out from their nest to search for food source, they use a special kind of chemical to communicate with each other. The chemical is referred to as the pheromone. Once the ants discover a path to food source, they deposit pheromone on the path. By sensing pheromone on the ground, ants can follow the path to food source discovered by other ants. As this process continues, most of the ants tend to choose the shortest path to food as there have been a huge amount of pheromones accumulated on this path. This collective pheromone- depositing and pheromone-following behavior of ants becomes the inspiring source of ACO.

In this paper, we apply the ant colony system (ACS) algorithm [17], which is one of the best ACO algorithms by now, to tackle the workflow scheduling problem in grid applications. Informally, the algorithm can be viewed as the interplay of the following procedures:

1) *Initialization of the algorithm.* All pheromone values and parameters are initialized.

2) *Initialization of ants.* Assume that a group of M ants are used in the algorithm. At the beginning of each iteration, all ants are set to initial state. Each ant chooses a constructive type (forward or backward) and a heuristic type (duration-greedy, cost-greedy, or overall-greedy). Based on the constructive type, each ant builds its tackling sequence of services.

3) *Solution construction.* M ants set out to build M solutions to the problem. The construction procedure includes n steps. n is the number of services in the workflow. In each step, each ant picks up the next service in its tackling sequence and maps it to one implementation out of the service's implementation domain using pheromone and heuristic information. The algorithm also estimates the earliest start time and earliest end time of services in terms of the information of partial solution built by each ant. This information is helpful to guide the search behavior of ants.

4) *Local updating.* Soon after an ant maps a service S_i to sp_i^j, the corresponding pheromone value is updated by a local pheromone updating rule.

5) *Global updating.* After all ants have completed their constructions, global pheromone updating is applied to the best-so-far solution. The cost and makespan of all solutions are evaluated. The pheromone values related to the best-so-far solution is significantly increased. Moreover,

some parameters of the algorithm are adaptively adjusted in this procedure.

6) *Terminal test*. If the test is passed, the algorithm is end. Otherwise, goto step 2) to begin a new iteration.

The flowchart of the algorithm is given by Fig. 4. These procedures are described in detail below.

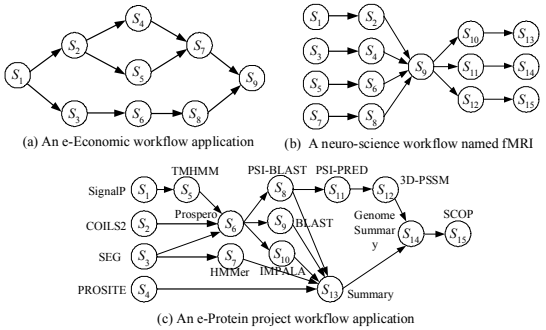

Fig. 3. Workflow applications used in our experiment： Fig. (a), a simple application of workflow with 9 tasks in e-Economic; Fig. (b), a neuro-science workflow with 15 tasks named fMRI [19]; Fig. (c), an e-Protein workflow with 15 tasks [20]

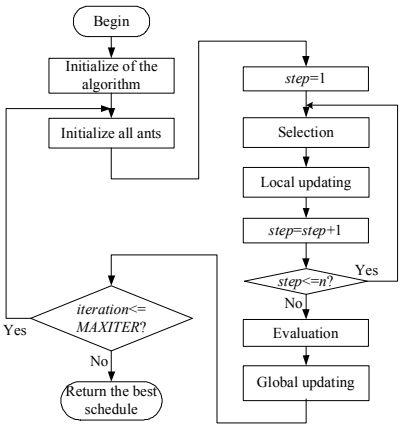

Fig. 4. Flowchart of the ACS algorithm

A. Definition of pheromone and heuristic

The considered problem is a bi-criteria problem to limit the makespan within deadline and to minimize the cost. Therefore, two types of pheromone are used. One represents the desirability from the view of makespan, and the other represents the desirability from the view of cost. We denote these two types of pheromone as $c\tau_{ij}$ and $d\tau_{ij}$ ($1 \le i \le n, 1 \le j \le m_i$). $c\tau_{ij}$ is the desirability of mapping service S_i to sp_i^j from the perspective of cost, and $d\tau_{ij}$ is desirability of the same mapping from the perspective of duration.

While initializing the algorithm, all pheromone values are initialized. That is,

$$c\tau_{ij} = c\tau_0, d\tau_{ij} = d\tau_0, \quad (1 \le i \le n, 1 \le j \le m_i), \qquad (2)$$

where $c\tau_0$ and $d\tau_0$ are two parameters, representing the initial values for $c\tau_{ij}$ and $d\tau_{ij}$ respectively. Similar to the ACS algorithm for TSP [17], we set $c\tau_0 = 1/(n \cdot c^{lb})$ and $d\tau_0 = 1/(n \cdot d^{lb})$. c^{lb} and d^{lb} are the lower bound estimation for the total cost and makespan respectively. Typically, c^{lb} can be set to the total cost when every service is mapped to its lowest-cost implementation, and d^{lb} can be set to the deadline D. So we have

$$c\tau_0 = 1/(n \cdot \sum_{i=1}^{n} \min_{1 \le j \le m_i} c_i^j) \qquad (3)$$

and

$$d\tau_0 = 1/(n \cdot D). \qquad (4)$$

The heuristic information for mapping S_i to sp_i^j is denoted as η_{ij}. We also use three kinds of different heuristic information to guide the search direction of ants, namely *duration-greedy*, *cost-greedy*, and *overall-greedy*. The definition is given by (5).

$$\eta_{ij} = \begin{cases} 1/d_i^j, & \text{if } selection\ type = duration\text{-}greedy \\ 1/c_i^j, & \text{if } selection\ type = cost\text{-}greedy \\ 1/(c_i^j \cdot d_i^j), & \text{if } selection\ type = overall\text{-}greedy \end{cases} \qquad (5)$$

In terms of this definition, *duration-greedy* heuristic bias the implementations with the shortest execution time, *cost-greedy* heuristic prefers the low-cost ones, and *overall-greedy* considers both factors.

B. Initialization of ants

At the beginning of each iteration, all ants are initialized. Each ant chooses a selection type from duration-greedy, cost-greedy, or overall-greedy according to (6):

$$selection\ type = \begin{cases} duration\text{-}greedy, & 0 \le ran < p_1 \\ cost\text{-}greedy, & p_1 \le ran < p_2 \\ overall\text{-}greedy, & p_2 \le ran \le 1 \end{cases} \qquad (6)$$

$p1$ and $p2$ ($0<p1<p2<1$) are two parameters and $ran \in [0,1]$ is a random number. It is apparent that the probabilities of choosing duration-greedy, cost-greedy, and overall-greedy are $p1$, ($p2$-$p1$), and (1-$p2$) respectively. We can see later that the values of p1 and $p2$ are adapted dynamically in our algorithm. The selection type of an ant is corresponding to the type of heuristic information it used while constructing solutions.

Each ant also has to select its constructive type (forward or backward) randomly and builds its tackling sequence of services. The tackling sequence is built as follows. A forward ant begins from the start service S_{start} and applies a random depth-first search to order all services. For example, the possible sequences built by a forward ant in the e-Economic workflow given by Fig. 3 (a) are (S_1•S_2•S_4•S_7•S_9•S_5• S_3•S_6•S_8), (S_1•S_2•S_5•S_7•S_9•S_4•S_3•S_6•S_8), (S_1• S_3•S_6•S_8•S_9•S_2•S_4•S_7•S_5), and (S_1•S_3•S_6•S_8 •S_9•S_2•S_5•S_7•S_4). Similarly, a backward ant begins from the end service S_{end} and uses a random backward depth-first search to order the services. The possible sequences built by a backward ant in the above example are (S_9•S_7•S_4•S_2•S_1•S_5•S_8•S_6•S_3), (S_9•S_7•S_5•S_2• S_1•S_4•S_8•S_6•S_3), (S_9•S_8•S_6•S_3•S_1•S_7•S_5•S_2 •S_4), and (S_9•S_8•S_6•S_3•S_1•S_7•S_4• S_2•S_5). The reason for using a depth-first search scheme is that the information of partial solutions (i.e., the earliest start time and earliest end time of services)

can be estimated, so that we can use this information to rule out the probabilities of selecting inferior components, which can be seen later. The reason for constructing the tackling sequences from both sides (forward and backward) is to diminish the influences exerted by the relative orders of services.

C. Solution Construction

After initialization, M ants set out to build solutions to the problem in parallel according to their tackling sequences. In step k ($1 \bullet k \bullet n$), each ant picks up the k^{th} service from its tackling sequence and maps it to an implementation out of the service's implementation domain. Assume that an ant is choosing one out of $SP_i = \{sp_i^1, sp_i^2, \cdots, sp_i^{m_i}\}$ to map to S_i, the selection rule is as follows:

Step 1: Evaluate the overall bias desirability of all implementations in terms of (7).

$$B_{ij} = \begin{cases} (d\tau_{ij})^{\alpha}(\eta_{ij})^{\beta},\ \eta_{ij} = 1/d_i^j, \\ \text{if the selection type of ant is } duration-greedy; \\ (c\tau_{ij})^{\alpha}(\eta_{ij})^{\beta},\ \eta_{ij} = 1/c_i^j, \\ \text{if the selection type of ant is } cost-greedy; \\ (c\tau_{ij})^{\alpha}(\eta_{ij})^{\beta},\ \eta_{ij} = 1/(c_i^j \cdot d_i^j), \\ \text{if the selection type of ant is } overall-greedy; \end{cases} \tag{7}$$

B_{ij} represents the bias of mapping S_i to $sp_i^j (1 \le j \le m_i)$. and are two parameters determining the weight of pheromone and heuristic information respectively.

Step 2: Adapt the values of B_{ij} in terms of the information from partial solution. The earliest start time and the earliest end time of services can be estimated for the current partial solution built by an ant. We denote the estimated earliest start time of S_i as $S_i.est$ and the earliest end time of S_i as $S_i.eet$. As the tackling sequence is built by depth-first search, it guarantees that a service will only be considered by a forward ant until at least one of its parent services has been considered. (Similarly, a service will only be considered by a backward ant until at least one of its child services has been considered. In the following text, we only discuss the situation for forward ants. The situation for backward ants comes when regarding all parent services as child services and regarding all child services as parent services.) So, when a forward ant is considering S_i, $S_i.est$ can be estimated as

$$S_i.est = \max_{S_k \in pred(S_i)} S_k.eet\ . \tag{8}$$

For example, a forward ant uses the sequence of ($S_1 \bullet S_2 \bullet S_4 \bullet S_7 \bullet S_9 \bullet S_5 \bullet S_3 \bullet S_6 \bullet S_8$) to build a solution for the e-Economic workflow (Fig. 3(a)). After mapping all services of the first branch ($S_1 \bullet S_2 \bullet S_4 \bullet S_7 \bullet S_9$) to corresponding implementations, we can estimate that $S_2.est=S_1.eet$, $S_4.est=S_2.eet$, $S_7.est=S_4.eet$, $S_9.est=S_7.eet$. When considering the next service S_5, we have $S_5.est=S_2.eet$.

On the other hand, sometimes the earliest start time for the child services of S_i may also have been estimated. For instance, when we are considering S_5 in the same example as the last paragraph, $S_7.est$ (S_7 is the son of S_5) has already been estimated. In this case, the available time slot for S_5 is limited by $S_2.eet$ and $S_7.est$. We define

$$slot_i = \begin{cases} undefined, \text{ if } \forall S_k \in succ(S_i), \\ \quad S_k.est \text{ has not been evaluated}; \\ (\min_{S_k \in succ(S_i) \text{ and } S_k.est \text{ has been evaluated}} S_k.est) - S_i.est, \\ \quad \text{otherwise}. \end{cases} \tag{9}$$

Based on this definition, if S_i is mapped to sp_i^j that satisfies $d_i^j > slot_i$, then $S_i.eet = S_i.est + d_i^j$ will be larger then at least one of its child's estimated earliest start time. In this situation, the estimated earliest start time for all children of S_i must be updated to be at least not smaller than $S_i.eet$. Otherwise, for all implementations that satisfy $d_i^j \le slot_i$, only the one with the lowest cost is useful, because all other choices will result in a higher-cost solution with the same makespan. So the ants will ignore these inferior choices by modifying the preferences B_{ij} using (10):

$$B_{ij} = \begin{cases} B_{ij}, \\ \text{if } d_i^j > slot_i \text{ or } slot_i = undefined; \\ \sum_{\forall sp_i^k (d_i^k \le slot_i)} B_{ik}, \\ \text{if } d_i^j \le slot_i \text{ and } c_i^j = \min_{\forall sp_i^k (d_i^k \le slot_i)} c_i^k; \\ 0, \\ \text{if } d_i^j \le slot_i \text{ and } c_i^j > \min_{\forall sp_i^k (d_i^k \le slot_i)} c_i^k. \end{cases} \tag{10}$$

Step 3: An ant selects one implementation out of $SP_i = \{sp_i^1, sp_i^2, \cdots, sp_i^{m_i}\}$ to map to S_i in terms of the following selection rule:

$$S_i \leftarrow \begin{cases} \arg\max_{\forall sp_i^j (1 \le j \le m_i)} B_{ij}, & \text{if } q \le q_0 \\ \text{roulette wheel scheme}, & \text{otherwise} \end{cases} \tag{11}$$

$$p_i^j = \frac{B_{ij}}{\sum_{k=1}^{m_i} B_{ik}}\ . \tag{12}$$

Equation (11) shows the pseudo random proportion selection rule. In this rule, a random number $q \in [0,1]$ is generated and is compared to a parameter q_0 ($q_0 \in [0,1]$). If $q \le q_0$, the implementation sp_i^j with the largest value of B_{ij} is chosen. Otherwise, a roulette wheel scheme is used. The probability of mapping S_i to sp_i^j is given by (12). In other words, the probability of selecting sp_i^j is in direct proportion to the value of B_{ij}.

D. Local updating

Immediately after an ant maps sp_i^j to S_i, local pheromone updating procedure is implemented. The updating rule is given by (13).

$$\begin{cases} d\tau_{ij} = (1-\xi) \cdot d\tau_{ij} + \xi \cdot d\tau_0, \\ \text{if selection type is } duration\text{-}greedy; \\ c\tau_{ij} = (1-\xi) \cdot c\tau_{ij} + \xi \cdot c\tau_0, \\ \text{otherwise}, \end{cases} \tag{13}$$

where $\xi \in [0,1]$ is a parameter. The function of local pheromone updating is to decrease the pheromone value corresponding to sp_i^j so that the following ants have higher probability to choose other implementations. Local pheromone updating procedure enhances the diversity of the algorithm.

E. Global Updating

Global updating takes place after all ants have built their solutions. Global pheromone updating only applies to the components on the best-so-far solution. Assume the best-so-far solution is $\{K_1,\cdots,K_n\}$, which means S_i is executed by $sp_i^{K_i}$ $(1 \le i \le n)$. The cost and makespan of the best-so-far solution are denoted as $cost^{bs}$ and $makespan^{bs}$. Then the global pheromone updating rule is given by (14).

$$\begin{cases} d\tau_{ij} = (1-\rho)\cdot d\tau_{ij} + \rho \cdot 1/makespan^{bs}, \\ \quad \text{if } makespan^{bs} > D; \\ c\tau_{ij} = (1-\rho)\cdot c\tau_{ij} + \rho \cdot 1/cost^{bs} \\ \quad \text{otherwise,} \end{cases} \quad , \tag{14}$$

where $\rho \in [0,1]$ is a parameter. The function of global pheromone updating is to reinforce the components on the best-so-far solution to speedup the convergence of the algorithm.

Additionally, the values of parameters p_1 and p_2 are adaptively tuned in this stage according to the best-so-far solution. The adaptive scheme is given by (15):

$$\begin{cases} p_1 = p_1 + 0.02, p_2 = p_2 + 0.01, \\ \text{if } makespan^{bs} > D \text{ and } p_2 + 0.1 < 0.99; \\ p_1 = 0.98, p_2 = 0.99, \\ \text{if } makespan^{bs} > D \text{ and } p_2 + 0.1 \ge 0.99; \\ p_1 = p_1 - 0.08, p_2 = p_2 - 0.04, \\ \text{if } makespan^{bs} \le D \text{ and } p_1 - 0.08 > 0.02; \\ p_1 = 0.02, p_2 = 0.49, \\ \text{if } makespan^{bs} \le D \text{ and } p_1 - 0.08 \le 0.02. \end{cases} \tag{15}$$

As has been mentioned before, the probabilities of choosing *duration-greedy*, *cost-greedy* and *overall-greedy* are p_1, $(p_2\text{-}p_1)$, and $(1\text{-}p_2)$ respectively. According to the adaptive scheme, if $makespan^{bs} > D$, the probability of choosing *duration-greedy* type (p_1) is increase, so that more ants use shorter duration implementations in their solutions. As a result, these ants tend to find some short makespan schedules to fit the deadline constraints. On the other hand, if $makespan^{bs} \le D$, it means the deadline constraint has been met. In this case, we increase the probability of using *cost-greedy* and *overall-greedy* so that the ants tend to construct low-cost solutions.

V. ACO Algorithm for the Scheduling Problem

A. Test instances

We test the ACS algorithm in the three workflow applications given in Fig. 3. Fig. 3(a) is an e-Economic workflow application with 9 services. Each service of this workflow has 5 different elements in its implementation domain provided by different GSPs. The cost and duration of each implementation are given by Table 1. Fig. 3(b) is a neuro-science workflow application provided by [19] with 15 services. It is called the functional MRI (fMRI) workflow. Fig. 3(c) is an e-Protein workflow application with 15 services derived from [20]. The name of each service in this application is also given in Fig. 3(c). This application aims at testing the issues in building annotation of the proteins in the major genomes using grid technologies. In our simulation, we randomly assign 2 to 10 different implementation modes to each service in the neuro-science workflow or the e-Protein workflow. The cost and duration of all implementations are also randomly set, but they follow the rule that for the same service a short duration implementation is corresponding to a high price and vice versa.

TABLE I
DURATION AND COST CORRESPONDING TO DIFFERENT IMPLEMENTATION MODES FOR THE SERVICES IN THE E-ECONOMIC WORKFLOW APPLICATION

	provider1		Provider2		Provider3		Provider4		Provider5	
	d	*c*	*d*	*c*	*d*	*c*	*d*	*c*	*d*	*c*
S_1	4	2000	8	1200	12	800	16	400	20	200
S_2	6	1000	7	800	8	500	9	300	10	200
S_3	3	500	4	350	5	250	6	150	7	100
S_4	6	500	7	400	8	300	9	200	10	100
S_5	4	400	5	350	6	300	7	250	8	220
S_6	2	1000	3	600	4	400	5	300	6	200
S_7	4	1000	5	850	6	700	7	650	8	500
S_8	4	350	5	300	6	250	7	200	8	150
S_9	12	2000	14	1800	16	1600	18	1300	20	1000

In the table, *d* means duration, and *c* means the cost.

B. Parameters and characteristics of the algorithm

Parameters of the algorithm are set as follows. The weights of pheromone and heuristic information in equation (7) are set to $\alpha = 1, \beta = 2$. The probability of selecting the implementation with the largest value of B_{ij} in equation (11) is q_0=0.8. Local pheromone updating rate in equation (13) is $\xi = 0.1$. Global pheromone updating rate in equation (14) is $\rho = 0.1$. In all experiments, the total iteration number is set to 1000, and the number of ants *M*=10. As the above parameters are regulars in the ACS algorithm, we configure these parameters basically according to the ACS algorithm for TSP [17]. Experimental results prove that this configuration still has good performance.

An interesting characteristic of this algorithm is that two types of pheromone values and three types of heuristic information are adopted to guide the ants towards two objectives: compressing the makespan within deadline and minimize the cost. We applied two parameters p_1 and p_2 in equation (6) to determine the probabilities of using duration-greedy, cost-greedy, and overall-greedy. In our algorithm, we set $p_1 = 0.8$ and $p_2 = 0.9$ initially, which means that the probability of using duration-greedy is 0.8, and the probability of using cost-greedy or overall-greedy is 0.1. The reason for assigning a large probability for duration-greedy initially is that ants are expected to find

solutions that are subject to the deadline constraint as soon as possible at the beginning of the algorithm.

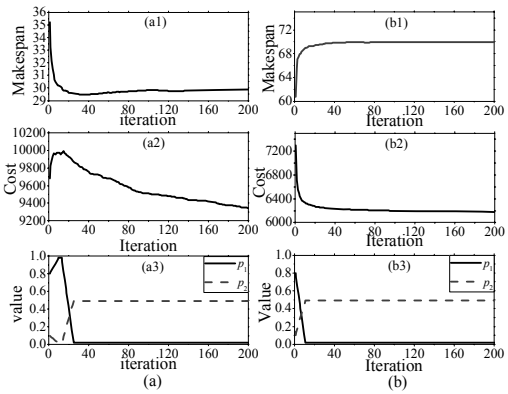

Fig. 5. The effect of adjusting the values of p_1 and p_2 according to equation (15): the three plots (a1), (a2), and (a3) in Fig. 5(a) illustrate the effect on e-Protein workflow application with deadline D=30. (a1) plots the makespan of the best-so-far solution as a function of the iteration number. (a2) plots the cost of the best-so-far solution as a function of the iteration number. (a3) plots the values of p_1 and p_2 as a function of the iteration number. Similarly, the three plots (b1), (b2), and (b3) in Fig. 5(b) illustrate the effect on e-Protein workflow application with deadline D=70. All data are averaged over 100 independent runs.

TABLE II
COMPARISON BETWEEN THE SCHEME OF ADAPTIVELY ADJUSTING AND THE NON-ADAPTIVE SCHEME WITHOUT ADJUSTING: THE COST OF THE BEST SOLUTION FOUND BY THE ALGORITHM IN EACH INSTANCE AVERAGED OVER 100 INDEPENDENT RUNS IS GIVEN THE TABLE.

E-ECONOMIC			FMRI			E-PROTEIN		
DL	AD	NA	DL	AD	NA	DL	AD	NA
36	**6466**	6527	35	**8056**	8180	35	**8106**	8177
40	**5676**	5741	40	**7174**	7494	40	**7759**	7774
44	**5082**	5201	45	**6323**	6642	45	**7460**	7491
48	**4626**	4713	50	**5977**	6102	50	**7177**	7229
52	**4072**	4169	55	**5441**	5503	55	**6914**	6959
56	**3670**	3736	60	**5171**	5206	60	**6636**	6686
60	**3270**	3296	65	**5127**	5154	65	**6392**	6424
64	**2870**	2876	70	**5132**	5161	70	**6145**	6169

DL=Deadline, AD=Adaptive adjusting scheme, NA=Without adaptive adjusting scheme

Additionally, in the process of the algorithm, we adapt the values of p_1 and p_2 using (15). The effect of this adaptive adjusting scheme can be illustrated by Fig. 5. Fig. 5(a) is the application of e-Protein (Fig. 3(c)) workflow with deadline D=30. As D=30 is a very tight constraint for this application, it is not easy for ants in the first few iterations to find solutions finished within deadline. In this case, the value of p_1 is increased so that ants bias short-execution-time implementations. This strategy is helpful for ants to find feasible solutions that are subject to deadline constraints. As a result, the makespan of the best-so-far solution drops to smaller than 30 within fifteen iterations, which is shown in Fig.5 (a1). Later, we decrease the value of p_1 so that ants tend to use low-cost implementations and the cost of the best-so-far solution decreases (Fig.5 (a2)).

Fig. 5(b) gives another case that the deadline constraint is loose (D=70). In this situation, ants manage to find feasible solutions within deadline with ease (Fig. 5(b1)). So the value of p_1 decreases immediately after the beginning of the algorithm and low-cost solutions can be found very quickly (Fig. 5(b2)).

The comparison between the algorithms with and without adaptive adjusting scheme is given by Table 2. It is apparent that the adaptive scheme manages to yield better performance in most cases. This proves that the adaptive scheme is able to improve efficiency of the algorithm.

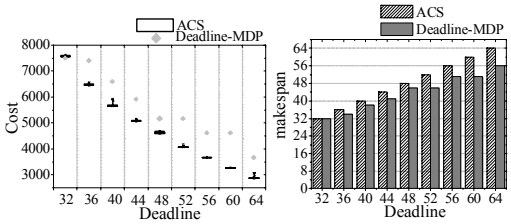

Fig. 6. Comparison between ACS and Deadline-MDP in the e- Economic workflow application given by Fig. 3(a): the left plot shows the cost of the best solution under different deadline constraints. The results derived from Deadline-MDP are marked by a big grey node. The results from ACS is recorded using a box chart, where the best 5% and the worse 5% out of 100 independent runs are marked by small nodes out of the box, and all other results are located in the box. The right plot shows the makespan of the best solution under different deadline constraints. The data of ACS are averaged over 100 runs.

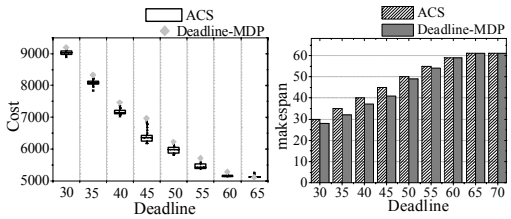

Fig. 7. Comparison between ACS and Deadline-MDP in the fMRI workflow application given by Fig. 3(b)

C. Performance comparison

We compare our ACS approach with the Deadline-MDP algorithm proposed by [13]. Deadline-MDP is a deterministic algorithm to tackle the same problem. This algorithm works as dividing the DAG into several partitions and distributing sub-deadline to each partition. Moreover, a Markov Decision Process (MDP) is applied to find the best solutions for pipeline partition branches in Deadline-MDP.

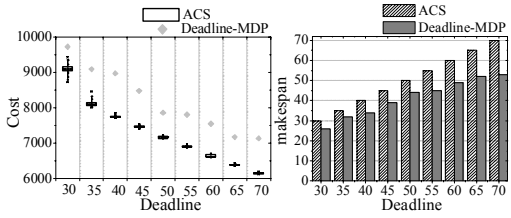

Fig. 8. Comparison between ACS and Deadline-MDP in the e-protein workflow application given by Fig. 3(c)

Experimental results in the three workflow applications are illustrated by Fig. 6, Fig. 7, and Fig. 8. It can be seen that in most cases even the worst solution found by ACS outperforms the one found by Deadline-MDP, especially in

the e-Protein workflow application. The results from both algorithms are able to meet all deadline constraints. However, the ACS approach tends to make full use of the time to minimize the cost. The makespans found by ACS are always equal to the deadline. On the other hand, although the makespan found by Deadline-MDP are shorter than ACS, the costs found by Deadline-MDP are much higher than ACS. This demonstrates the effectiveness of the ACS approach.

REFERENCES

[1] 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.

[2] 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.

[3] E. Deelman et al. "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, Vol. 1, pp. 25-39, 2003.

[4] E. Deelman et al. "Pegasus: planning for execution in grids", *Technical report, GRIPHYN 2002-20*, 2002.

[5] T. Fahringer, et al. "ASKALON: a tool set for cluster and Grid computing", *Concurrency and Computation: Practice and Experience*, Vol. 17, pp. 143-169, Wiley InterScience, 2005.

[6] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler", *Beowulf Cluster Computing with Linux*, the MIT Press, October, 2001

[7] G. Singh, C. Kesselman, and E. Deelman, "Optimizing grid-based workflow execution", *Journal of Grid Computing*, 3 (2006), pp. 201-219.

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

[9] M.M. López, E. Heymann, and M.A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems", *Proceedings of the fifth international symposium on parallel and distributed computing (ISPDC' 06)*, 2006.

[10] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker", *Future Generation Computer Systems*, Vol. 18, pp. 1061-1074, 2002.

[11] C.-H. Chien, P.H.-M. Chang, and V.-W. Soo, "Market-oriented multiple resource scheduling in grid computing environments", *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA' 05)*, 2005.

[12] 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.

[13] 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.

[14] 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 1991.

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

[16] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Transactions on Data and Knowledge Engineering*, vol. 11, No. 5, 1999, pp. 769-778.

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

[18] K. K. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions", *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 33, No. 5, 2003.

[19] Y. Zhao, et al. "Grid middleware services for virtual data discovery Composition, and integration", *In 2nd Workshop on Middleware for Grid Computing*, October 18, 2004, Toronto, Ontario, Canada.

[20] A. O'Brien, S. Newhouse and J. Darlington, "Mapping of scientific workflow within the e-Protein project to distributed resources", *In UK e-Science All Hands Meeting*, Nottingham, UK, Sep. 2004.

[21] M.P. Papazoglou and D. Georgakopoulos, "Service-oriented computing", *Communications of the ACM*, Vol. 46, No. 10, pp. 25-28, October, 2003.

[22] J. Yu, S. Venugopal, and R. Buyya, "Grid Market Directory: A Web Services based Grid Service Publication Directory", *Technical Report, Grid Computing and Distributed Systems Lab*, University of Melbourne, 2003.