MINIMIZING EXECUTION COST FOR APPLICATIONS WITH DEADLINE AND RELIABILITY CONSTRAINT IN UTILITY GRID

Li Wang, Shangping Ren, Shuhui Li
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, United States
email: {lwang64, ren, sli38}@iit.edu

Gang Quan
Department of Electrical and Computer Engineering
Florida International University
Miami, FL, United States
email: gang.quan@fiu.edu

ABSTRACT
Grid computing has become a widely used approach to solving large-scale complex problem because of its powerful computing capability. For different computing resources in a grid, their price, capability, reliability, and availability may vary. When deploying tasks to computing resources, how to minimize the execution cost, and at the same time complete the task execution without violating the user's constraints (such as deadline, reliability, etc.) is a challenging problem. In fact, it is a NP-hard problem. In our work, we provide an Ant Colony System (ACS) based approach to solving the task deployment problem. The experimental results indicate good performance of our ACS based approach over other traditional heuristic approaches.

KEY WORDS
Utility Grid, Ant Colony System, Execution Cost Minimization, Heterogeneous Computing Resources

1 Introduction

Grid computing is to share, select, and integrate resources that are geographically distributed and have different capacity, reliability, and availability [1]. It has emerged for solving large-scale complex problems in science, engineering, and industry. Recently, grid computing has progressed towards a service-oriented paradigm in which users pay for executing their applications and providers offer their computing resources for making profit [2]. The environment with decoupling of users from providers is generally termed as Utility Grid [3, 4].

Previous work [2, 5, 6] has proposed grid market infrastructures for utility grids. In these grid market infrastructures, resource providers aim to maximize their profit by offering a competitive resource access cost in order to attract consumers. Users have an option of choosing the providers that best meet their requirements. Grid brokers are part of the infrastructures and work as a mediator between users and grid resource providers. They perform resource discovery, negotiate for access costs using trading services, map jobs to resources (scheduling), stage the application and data for processing (deployment), start job execution, and finally gather the results [6].

In utility grids, users interact with grid brokers and express their requirements such as the maximum amount of *budget* that they have for executing an application, the minimum application reliability [7], and a deadline by which the application execution must be completed. The resource providers express their pricing policies, reliability information, and available time slots. The goal of the grid brokers is to find a mapping between the application tasks and resources so that the reliability and deadline of the application are met and at the same time, the cost of the application execution is minimized.

However, this task scheduling problem is not only strongly NP-hard but also nonapproximable, i.e., it cannot be approximated in polynomial time within a good lower bound. This is because a simpler version of the problem, i.e., minimizing the total execution cost of a set of independent tasks within the deadline has been proved to be strongly NP-hard and nonapproximable [8, 9]. Therefore, the focus of our work is to design a heuristic approach to solving this problem.

Ant Colony System (ACS) [10] is a population-based heuristic algorithm which is biologically inspired by the behavior of ants in finding path from the nest to a food source. When solving a given problem, artificial ants probabilistically construct solutions using pheromone trails and heuristic information. The pheromone trails are updated according to the quality of the solutions constructed by the ants. The procedure is repeated until predefined ending conditions are met. In our work, we develop an ACS based approach to solving this task scheduling problem.

The rest of this paper is organized as follows. We discuss related work in Section 2. In Section 3, we define system model and formulate the problem. We present an ACS based approach to solving the task scheduling problem in Section 4. The experimental results are discussed in Section 5. Finally, we conclude our work and point out the future work in Section 6.

2 Related Work

Task scheduling problem has been extensively studied from different aspects. For example, in [11, 12, 13], researchers proposed different heuristic approaches to minimizing the makespan when allocating a set of tasks to heterogeneous

processors. However, these work did not consider the cost when scheduling the tasks.

In [14], Liu et al. presented a general-purpose resource selection framework that provides a resource selection service for different kind of applications. This framework identifies a suitable resource set for applications based on the applications' characteristics and realtime status. He et al. proposed a QoS guided task scheduling algorithm for Grid computing [15]. While Min et al. in [16] proposed an algorithm for scheduling co-reservations on heterogeneous resources. The main objective of these algorithms is to satisfy the given requirements, i.e., meet the execution constraints. The research presented in this paper goes beyond simply meeting application's constraints, it also tries to optimize the applications' objective, such as minimizing the application's execution cost.

In [17], Buyya et al. developed a Nimrod/G scheduler framework which supports deadline and budget constraints. However, they only considered independent tasks in their model. Ju et al. in [18] proposed a genetic algorithm for a grid workflow application to minimize its duration with budget constraints. In [19], Yu et al. proposed a cost-based workflow scheduling algorithm which aims to minimize the cost while meeting the deadline constraint. This approach only works when the number of tasks is small because it involves factorial time to calculate local deadline for the tasks.

The researchers mentioned above are based on the assumptions that at all time there are unlimited resources available and individual resource never goes down. However, the two assumptions may not always hold [7]. In this paper, we address the task deployment problem under the condition where the computing resources in a grid are not always available and may fail with different failure rates. The detailed system model and assumptions are presented in the next section.

3 System Model and Problem Formulation

3.1 Resource Model

We use set $\mathcal{S} = \{s_1, s_2, \cdots, s_k\}$ to denote the group of available computing resources in a grid. For different resources, their computing capability, price, available time slots, and reliability may vary. We use $\overrightarrow{P} = [p_1, p_2, \cdots, p_k]$ to represent the price of using resource s_j $(1 \leq j \leq k)$ for a unit time, and $\overrightarrow{T} = [t_1, t_2, \cdots, t_k]$ refers to the available time slots of the resources before the deadline D of application α, where $t_i = [(b_i^1, u_i^1), (b_i^2, u_i^2), \cdots, (b_i^m, u_i^m)]$, b_i^j and u_i^j refer to the start time and end time of the jth slot of resource s_i. We use $\overrightarrow{\Lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_k]$ to denote the failure rate of each computing resource.

For resource s_j, if its running time is L_j, the reliability of resource s_j is defined as [7]

$$R_{s_j} = e^{-\lambda_j L_j} \tag{1}$$

3.2 Application Model

We assume that an application α has multiple tasks with data dependencies among them. The application is represented by a directed acyclic graph (DAG) $\alpha = (N, E, D)$, where $N = \{\tau_1, \tau_2, \cdots, \tau_n\}$ represents the set of tasks the application α has, E is the set of edges between the tasks. Each edge $e_{i,j} = (\tau_i, \tau_j)$ represents the data dependency constraint such that task τ_j cannot start until task τ_i completes its execution. D is the application's end-to-end deadline.

For task τ_i, we use $pred(\tau_i)$ to denote the set of its immediate predecessors, and $succ(\tau_i)$ the set of its immediate successors. A task having no parent is called an *entry task* and a task having no child is called an *exit task*. If the application has multiple entry tasks, we create a dummy task which is the parent task of the entry tasks. Similarly, if the application has multiple exit tasks, we create a dummy task as the child task of all the exit tasks. This way guarantees that there is only one entry task and exit task in an application.

We further assume that the execution time of tasks on different resources is known, and is represented by the Task-Execution-Time matrix $\mathbf{W}_{n\times k} = (w_{i,j})_{n\times k}$, where $w_{i,j}$ denotes the execution time of completing task τ_i by resource s_j. If resource s_j cannot perform task τ_i, the execution time is $+\infty$. For each task, it can be assigned to only one resource for execution. In addition, as tasks' execution time is usually much larger than the data transmission time even the tasks are not on the same processing element. Therefore, in our work, the data transmission time and data transmission cost are not taken into consideration.

Due to data dependency among tasks, task τ_i cannot start to execute until the execution of all its predecessors have been finished. Furthermore, task τ_i cannot be executed before its assigned resource is ready. In our work, we assume that once the assigned computing resource is ready and its predecessors are finished, task τ_i starts to execute immediately. Therefore, for task τ_i, its start time (i.e., $ST(i,j)$) and finish time (i.e., $FT(i,j)$) on resource s_j are defined as

$$ST(i,j) = \max\{T_{ready}(i,j), \max_{\tau_g \in pred(\tau_i)} FT(g, SI(g))\} \tag{2}$$

$$FT(i,j) = w_{i,j} + ST(i,j) \tag{3}$$

where $T_{ready}(i,j)$ is the ready time of resource s_j to execute task τ_i, $pred(\tau_i)$ is the set of immediate predecessors of task τ_i, and $SI(g)$ refers to the index of resource where the task τ_g is deployed.

3.3 Application Reliability and Execution Cost

We use Task-Deployment matrix $\mathbf{M}_{n\times k} = (x_{i,j})_{n\times k}$ to represent the task deployment scheme, where $x_{i,j}$ $(x_{i,j} \in$

$\{0,1\}$) denotes whether task τ_i is allocated to resource s_j. To be more specific, $x_{i,j} = 1$ indicates that task τ_i is deployed on resource s_j, and $x_{i,j} = 0$ indicates the opposite. As each task can be assigned to only one resource, we have

$$\sum_{j=1}^{k} x_{i,j} = 1, \ \forall\, i = \{1, 2, \cdots, n\} \tag{4}$$

When task τ_i is allocated to resource s_j, the execution cost $c_{i,j}$ is $c_{i,j} = w_{i,j} \times p_j$. Based on the resource reliability model, the probability that task τ_i can be successfully executed by resource s_j, i.e., the reliability of task τ_i is

$$r_{i,j} = e^{-\lambda_j w_{i,j}} \tag{5}$$

As a single task failure will lead to application failure, therefore, the reliability of application is defined as

$$R = \prod_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} \times r_{i,j} = \prod_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} \times e^{-\lambda_j w_{i,j}} \tag{6}$$

and the total execution cost is

$$C = \sum_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} \times w_{i,j} \times p_j \tag{7}$$

where $w_{i,j}$ denotes the execution time of task τ_i by resource s_j, p_j is the price of resource s_j for a unit time, k and n refer to the number of computing resources and tasks.

3.4 Problem Formulation

The problem of minimizing application execution cost under deadline and reliability constraints, i.e., MAEC_DRC problem, can be formulated as following:

Problem 1 *Given k heterogeneous computing resources $(\mathcal{S})$ with different unit price $(\overrightarrow{P})$, available time slots $(\overrightarrow{T})$, and failure rates $(\overrightarrow{\Lambda})$, decide a task deployment $\mathbf{M}_{n\times k}$ for application $\alpha = (N, E, D)$ with objective:*

$$\textit{minimize} \qquad C = \sum_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} \times w_{i,j} \times p_j \tag{8}$$

Subject to:

$$\prod_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} \times e^{-\lambda_j w_{i,j}} \geq R_{\alpha}^{min} \tag{9}$$

$$FT(\tau_{exit}, z) \leq D \tag{10}$$

$$\sum_{j=1}^{k} x_{i,j} = 1, \qquad \forall i = \{1, 2, \cdots, n\} \tag{11}$$

where $w_{i,j}$ is the execution time of task τ_i on resource s_j, $FT(\tau_{exit}, z)$ refers to the finish time of exit task τ_{exit} when it is deployed on resource s_z, and R_{α}^{min} is the minimum reliability of application α. □

4 ACS based Apprpoach to Solving the Task Scheduling Problem

In this section, we will give the detailed implementation of the ACS based approach to solving the MAEC_DRC problem.

4.1 Pheromone Initialization

In the ACS algorithm, pheromone represents historical search experiences of ants, and the pheromone values of component solutions impact the resource selection when deploying tasks. At the beginning, the pheromone values of all solution components are initially set to η_0. In our work, we follow the approach given in [10] to set the initial value of pheromone η_0 as

$$\eta_0 = (\sum_{i=1}^{n} c_i^{max})^{-1} \tag{12}$$

where n is the number of tasks, and $c_i^{max} (c_i^{max} \neq +\infty)$ refers to the maximum execution cost of task τ_i on each resource. η_0 is also the minimum amount of pheromone on each solution component.

4.2 Ants Initialization

Based on the system model, task cannot be executed until the execution of all its predecessors have finished. Therefore, at the beginning, ants are placed at the entry task of the application. In other words, the entry task is the first task to be deployed by the ants.

4.3 Solution Construction

When ants build solutions, they need to deploy each task to a resource. There are two questions to be answered: First, what is the order for deploying tasks? Second, what is the criteria when choosing a resource for a task?

4.3.1 Task Deployment Order

As in the system model, we assume there may be data dependency among tasks, hence, the predecessors of task τ_i should be deployed to resources before task τ_i. In order to reflect the dependency relationship, we use the absolute deadline $AD(\tau_i)$ as deployment priority. The absolute deadline refers to the latest time a task must be finished, or the exit task will miss the application's deadline. The absolute deadline is recursively defined by (13).

$$AD(\tau_i) = \min_{\tau_j \in succ(\tau_i)} \{AD(\tau_j) - w_j^{min}\} \tag{13}$$

where $succ(\tau_i)$ refers to the set of successors of task τ_i, w_j^{min} is the minimum execution time on all resources, and the absolute deadline of the exit task is the application's deadline, i.e., D.

4.3.2 Resource Selection Criteria

When deploying task τ_i, ant y follows the ACS transition rule [10] defined in (14) to select resource.

$$s_j(q) = \left\{ \begin{array}{ll} \max\limits_{s_j \in V_y(i)} \{[\eta(i,j)] \times [\gamma(i,j)]^\beta\} & \text{if } q < q_0 \\ S & \text{otherwise} \end{array} \right\} \quad (14)$$

where $V_y(i)$ is the set of resources that τ_i can be deployed to by ant y, $\eta(i,j)$ and $\gamma(i,j)$ are the pheromone value and heuristic information value of solution component $sc(i,j)$, respectively. β is an adjustable parameter which determines the relative importance of pheromone versus heuristic information. q is a randomly selected number in the interval [0,1], q_0 ($0 \leq q_0 \leq 1$) is a tunable parameter. S refers to the resource which is chosen according to the probability given by (15). Condition $q \leq q_0$ corresponds to an exploitation of the current best local solution, while $q > q_0$ favors more exploration.

$$p_y(i,j) = \frac{[\eta(i,j)] \times [\gamma(i,j)]^\beta}{\sum\limits_{s_j \in V_y(i)} [\eta(i,j)] \times [\gamma(i,j)]^\beta} \quad (15)$$

Heuristic information $\gamma(i,j)$ is a key factor for ACS algorithm. It directly reflects the quality of solution components. Similar to pheromone value $\eta(i,j)$, the heuristic information $\gamma(i,j)$ influences the search direction of the ants. For different problems, the meanings of heuristic information and rules to decide the heuristic information values are different.

Based on the system model, our goal is to minimize the application's total execution cost while meeting application's deadline and its reliability constraints. Therefore, when deploying tasks to the resources, we cannot simply use the execution cost as the only criteria. Rather, we must also consider the task's response time and reliability.

In our work, we use the idea of the effective-gradient based approach [20] to determine the heuristic information value for different solution components. The effective-gradient based approach was proposed by Toyoda in [20] to solve the multidimentional knapsack problem (MKP), and the experimental results show good performance of this approach.

In the MKP, there is a set of n items with values v_j and m knapsacks with capacities $c_i = 1$. Each item j consumes $d_{i,j}$ ($0 \leq d_{i,j} \leq 1$) spaces from each knapsack i. The goal is to select a subset of items to maximize the total values while the space constraint is met.

The idea of effective-gradient based approach is that when selecting the items, each item is evaluated in terms of its effective gradient which is computed by using a penalty vector, and the items are selected according to the calculated values. The effective gradient of item i is defined as

$$G_i = \frac{v_j}{\Gamma_i \times \Gamma_u} \quad (16)$$

where $\Gamma_u = (c_1, c_2, \cdots, c_m)$ is a penalty vector, which is the used space of each knapsack, $\Gamma_i = (d_{i,1}, d_{i,2}, \cdots, d_{i,m})$ is the demand of item i on each knapsack. The product of Γ_i and Γ_u measures the penalty to use the spaces of knapsacks.

The most important characteristic of this approach is that when selecting items, it not only considers the value v_i and space requirement Γ_i on each knapsacks, it also considers the amount of used space Γ_u in each knapsack. According to (16), the less available space in the knapsack, the higher penalty will be added to the items if they require that space. This way, the remaining space in each knapsack will be balanced, and more items can be selected.

In the MAEC_DRC problem, each solution component is associated with execution cost, response time, and task reliability. In addition, for application α, its maximum total execution time (i.e., end-to-end deadline) and minimum reliability are fixed. Therefore, when applying the idea of the effective-gradient based approach in the MAEC_DRC problem, the maximum total execution time and minimum reliability are regarded as the knapsacks in the MKP problem, the solution components are regarded as the items in the MKP problem. In other words, when selecting the solution components, the constraints, i.e., maximum total execution time and minimum reliability, can not be voilated.

In the effective-gradient based method, the space of each knapsack is set to 1, and the space required by each item in each knapsack is normalized to a value between 0 and 1. In our problem, we also normalize both maximum execution time and application's minimum reliability to 1, and for different solution components, their normalized response time $\widehat{rt(i,j)}$ and task reliability $\widehat{r_{i,j}}$ are

$$\widehat{rt(i,j)} = (FT(i,j) - \max_{\tau_x \in pred(\tau_i)} \{FT(x, SI(x))\})/D \quad (17)$$

$$\widehat{r_{i,j}} = (1 - r_{i,j})/(1 - R_\alpha^{min}) \quad (18)$$

where $SI(x)$ refers to the resource index where the task τ_x is deployed, $FT(i,j) - \max\limits_{\tau_x \in pred(\tau_i)} \{FT(x, SI(x)\}$ refers to the response time of resource s_j to execute task τ_i.

We use $\widehat{T_u}$ to denote the normalized used time, and we have

$$\widehat{T_u} = \max_{\tau_x \in pred(\tau_i)} \{FT(x, SI(x))\}/D \quad (19)$$

Similarly, we use $\widehat{R_u}$ to denote the normalized application reliability, and we have

$$\widehat{R_u} = (1 - \prod_{\tau_j \in deployed(\tau_i)} r_{j,SI(j)})/(1 - R_\alpha^{min}) \quad (20)$$

where $deployed(\tau_i)$ refers to the set of tasks which are deployed before task τ_i.

Based on the discussion above, the heuristic information $\gamma(i,j)$ of solution component $sc(i,j)$ is defined as:

$$\gamma(i,j) = \frac{1}{c_{i,j}} \times \frac{1}{\Gamma_{i,j} \times \Gamma_u} \quad (21)$$

where $c_{i,j}$ refers to the execution cost if task τ_i is deployed to resource s_j, $\Gamma_{i,j} = (\widehat{rt(i,j)}, \widehat{r_{i,j}})$, and $\Gamma_u = (\widehat{T_u}, \widehat{R_u})$.

4.4 ACS Local update Rule

While building a solution to the MAEC_DRC problem, ants go through each task and deploy it to a resource according to the rule given in (14). If an ant chooses resource s_j to execute task τ_i, the ant updates the pheromone value of $\eta(i,j)$ by applying the local update rule [10] defined by (22).

$$\eta(i,j) \leftarrow (1-\rho) \times \eta(i,j) + \rho \times \eta_0 \tag{22}$$

where ρ $(0 \leq \rho \leq 1)$ is the evaporation parameter which controls the local trail decay, and $\eta_0 = (\sum_{i=1}^{n} C_i^{max})^{-1}$. As η_0 is the minimum amount of pheromone on each solution component, based on (22), we can see that once the ant deploys task τ_i to resource s_j, the pheromone value $\eta(i,j)$ of solution component $sc(i,j)$ will be decreased. In other words, the local update rule reduces the convergence because less ants will deploy task τ_i to resource s_j, hence solution diversity is increased, and the probability that global best solution can be found increases.

4.5 ACS Global Update Rule

In ACS, global update is performed after all ants have completed their solution construction. In addition, only the best ant is allowed to globally update the pheromone of the solution component it selected. In our work, according to the objective of the problem, the quality of the solution is decided by the total execution cost and whether the solution meets the deadline and reliability constraints. Therefore, if the solution violates the constraints, a penalty should be considered.

We use $sol(y)$ to denote the solution constructed by ant y, if either the deadline or the reliability constraint is violated, the quality of solution is defined as

$$Q(sol(y)) = \frac{1}{sol(y).cost} \times \min\{\frac{D - sol(y).deadline}{D}, \frac{sol(y).reliability - R_\alpha^{min}}{R_\alpha^{min}}\} \tag{23}$$

otherwise,

$$Q(sol(y)) = \frac{1}{sol(y).cost} \tag{24}$$

Once the current best solution is found, the pheromone level of the solution components chosen by the best ant is updated by applying the ACS global update rule [10] which is defined by (25).

$$\eta(i,j) \leftarrow (1-\rho) \times \eta(i,j) + \rho \times \frac{1}{cost_{gb}} \tag{25}$$

where $cost_{gb}$ is the execution cost obtained by the best solution. From (25), we can see that the pheromone levels of all solution components in the globally best solution are reinforced after the global update.

4.6 Termination Condition Check

The ACS based approach stops when the predefined condition is met. In our work, either the total number of iterations exceeds $\mathcal{T}$ or the consecutive value of the execution cost remains the same for $\mathcal{U}$ times, the whole process is terminated.

5 Experimental Evaluation

In this section, we investigate the performance of the ACS based approach when solving the MAEC_DRC problem. In particular, we compare our approach with two other approaches, i.e., Greedy-Cost approach and Deadline-Level approach which are proposed in [17, 19, 21]. To be more specific, Greedy-Cost approach allocates tasks to the available computing resources with minimum cost. While Deadline-Level approach first gets the level (i.e., the depth of task in the DAG) of tasks, then assigns a sub-deadline to the tasks based on their levels. When deploying tasks, it chooses the resource with minimum execution cost which also completes the task execution within task's sub-deadline.

In our experiment, we consider three common application structures presented in [22]: pipeline, parallel, and hybrid structure. For pipeline applications, the tasks are executed in sequential order. For parallel applications , multiple pipelines are executed in parallel. While for hybrid structure, it is a combination of pipeline and parallel strucutre.

5.1 Experiment Settings

When the number of tasks and resources are fixed, based on the system model, the execution cost is affected by the application's end-to-end deadline and reliability constraints. Therefore, in order to compare the performance of different approaches, we run the test cases with different end-to-end deadline and reliability constraints.

In our implementation, in order to reflect heterogeneity among tasks, we use UUnifast algorithm [23] to generate the execution time for each task. To be more specific, assume the total execution time for n tasks is E_{total}, the UUnifast algorithm uniformly distributes E_{total} to task τ_i with $0 < w_i < E_{total}$ $(1 \leq i \leq n)$. The algorithm guarantees $E_{total} = \sum_{i=1}^{n} w_i$.

When there are k computing resources, we first obtain the execution time w_i for task τ_i $(1 \leq i \leq n)$. We then apply the UUnifast algorithm again to choose the execution time of task τ_i on the group of computing resources s_j $(1 \leq j \leq k)$, i.e., $w_{i,j}$. This way, the average execution time of task τ_i on the computing resources remains the same, i.e., w_i.

For all resources, their failure rate λ_j $(1 \leq j \leq k)$ and unit price p_j $(1 \leq j \leq k)$ are uniformly selected in the range $[5 \times 10^{-6}, 10 \times 10^{-6}]$ and $[10, 20]$, respectively. For

each computing resource, the higher unit price, the lower failure rate.

In order to apply the ACS based approach, we have to first decide the following parameters, the number of ants n_{ants}, pheromone evaporation rate ρ, β, and q_0. In our experiment, as suggested in [10], we set $\rho = 0.1$, and the remaining parameters of the ACS based approach for different application structures are given in Table 1. All the values are obtained by using experiments. In addition, when either the total number of iterations exceeds $\mathcal{T} = 1000$ or the consecutive value of the execution cost remains the same for $\mathcal{U} = 100$ times, we stop the whole process.

Table 1: Parameter Values

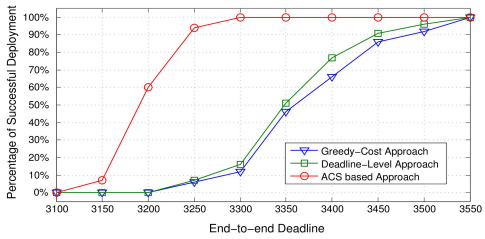
Structure	n_{ants}	ρ	β	q_0
Pipeline	11	0.1	13	0.6
Parallel	9	0.1	18	0.55
Hybrid	11	0.1	9	0.8

5.2 Comparison under Different Application Structures

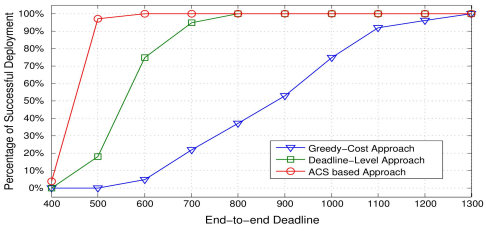
In the first set of experiments, we compare the percentage of successful task deployment of the three approaches under different end-to-end deadlines (application's reliability is not considered). In this experimtal setting, we assume the total number of computing resources and the number of tasks in an application is 20 and 120, respectively. In addition, we set the total execution time for 120 tasks is 6000. The actual execution time on the resources is decided by the UUnifast algorithm.

From Fig. 1, we can see that for all three approaches, the percentage of successful task deployment under different application structures increases with the end-to-end deadline. This is because when more execution time is given, more tasks can be executed. In addition, we can also see that under these three structures, ACS based approach performs better than Deadline-Level approach and Greedy-Cost approach performs the worst. The reason is when deploying tasks, Greedy-Cost approach deploys the tasks to the resource with minimum execution cost without considering the task execution and waiting time, hence leaving limited time for the remaining tasks. Although Deadline-Level approach considers sub-deadline when deploying tasks, the sub-deadlines are calculated based on the tasks' level rather than their execution time. In other words, some tasks may be allocated more execution time, which may lead to deadline miss for the remaining tasks. The ACS based approach performs the best because it applies a dynamic task deployment rule when deploying tasks, i.e., task deployment depends on task's execution time and remaining time. In other words, tasks may be deployed to expensive resources when the remaining time is small, hence, more tasks can be deployed.

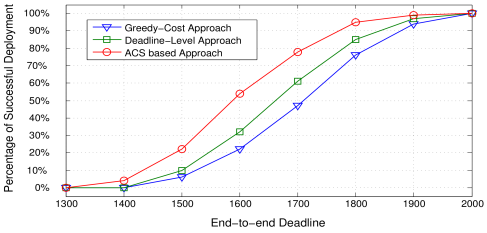
In the second set of experiment, we compare the percentage of successful task deployment of these three approaches under different application's minimum reliability (reliability bound). The experimental setting is the same as given in the first set of experiment except the end-to-end deadline is not considered.



(a) Pipeline Structure



(b) Parallel Structure

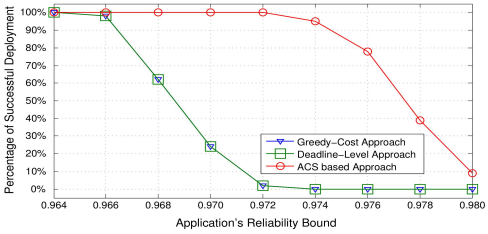


(c) Hybrid Structure

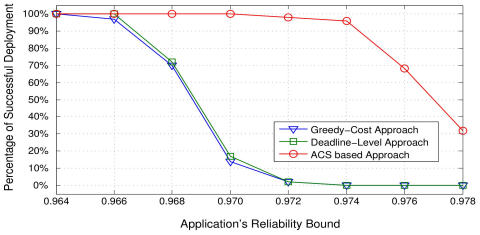
Figure 1: Percentage of successful task deployment under different end-to-end deadlines

Fig. 2 shows how the percentage of successful task deployment changes with application's minimum reliability. From Fig. 2, we can see that when application's minimum reliability increases, the percentage of successful deployment drops. This is because obtaining higher application's reliability usually requires shorter execution time, which may not be achieved when all tasks cannot be finished within the desired execution time even they are deployed to the resources with minimum execution time.

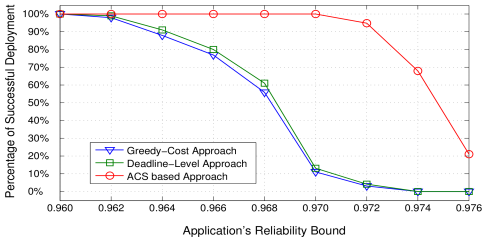
Another observation from Fig. 2 is that the percentage of successful task deployment of ACS based approach is much larger than the other two approaches. The main reason is that for computing resources, their high reliability also comes with high unit price, as both Greedy-Cost approach and Deadline-Level approach deploy tasks to the resources with minimum cost, therefore, the deployment scheme obtained by these two approaches has low application reliability, which may violate the application's minimum reliability.



(a) Pipeline Structure

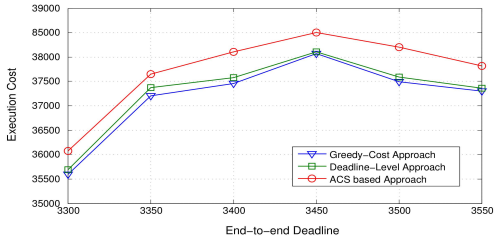


(b) Parallel Structure

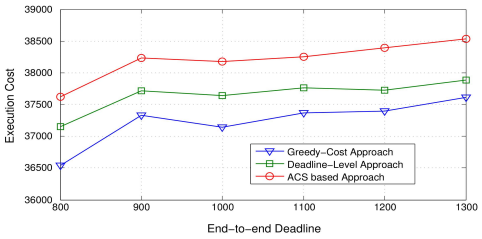


(c) Hybrid Structure

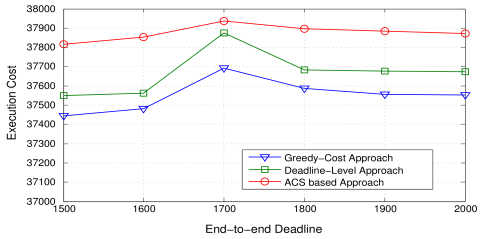
Figure 2: Percentage of successful task deployment under different reliability bound



(a) Pipeline Structure



(b) Parallel Structure



(c) Hybrid Structure

Figure 3: Execution Cost under different end-to-end deadlines

In the third set of experiments, we compare the execution cost obtained by different approaches. For different approaches, their execution cost, finish time, and application reliability may not be the same even under the same experimental setting. In other words, although an approach obtains the smallest execution cost, its finish time may be larger than the maximum allowed execution time. Therefore, in this experiment, we only consider the case in which all approaches obtain the feasible deployment scheme under the same experimental setting.

Fig. 3 shows the execution cost obtained by different approaches under different structures when the application's end-to-end deadline increases. From Fig. 3, we can see that Greedy-Cost approach performs best while our ACS based approach performs worst. This is because we only compare the successful deployments. As Greedy-Cost approach always deploys tasks to the resources with minimum execution cost, therefore, if the deployment is successful, the execution cost obtained by the Greedy-Cost approach is also the minimum execution cost. From Fig. 3, we can see the execution cost obtained by ACS based approach is close to the minimum execution cost (the maximum difference is less than 3% under all three structures.), which indicates good performance of the ACS based approach.

6 Conclusion

Grid computing has emerged for the solution of larger-scale complex problems in different fields. However, deploying dependent tasks to computing resources with the goal of minimizing execution cost is difficult to achieve. In our work, we provide an ACS based approach to solving the task deployment problem, and the results indicate good performance of our ACS based approach over other heuristic approaches.

In our work, we only consider minimizing the execution cost from the perspective of users, how to maximize the profit of resource providers is another interesting and practical problem. This problem will be discussed and solved in our future work.

Acknowledgement

The work is supported in part by NSF CAREER CNS 0746643 and NSF CNS 1018731.

References

[1] I. Foster and C. Kesselman, eds., *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[2] D. Neumann, J. Stoesser, A. Anandasivam, and N. Borissov, "Sorma - building an open grid market for grid resource allocation," in *Proceedings of the 4th international conference on Grid economics and business models*, pp. 194–200, 2007.

[3] S. K. Garg, R. Buyya, and H. J. Siegel, "Scheduling parallel applications on utility grids: time and cost trade-off management," in *Proceedings of the Thirty-Second Australasian Conference on Computer Science*, pp. 151–160, 2009.

[4] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, pp. 217–230, Dec. 2006.

[5] J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen, "Gridecon - the economic-enhanced next-generation internet," in *Proceedings of the 4th international conference on Grid economics and business models*, GECON'07, (Berlin, Heidelberg), pp. 188–193, Springer-Verlag, 2007.

[6] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service oriented grid computing," in *Proceedings of the Heterogeneous Computing Workshop*, 2001.

[7] Y. Dai, M. Xie, and K. Poh, "Reliability of grid service systems," *Computers & Industrial Engineering*, vol. 50, no. 1C2, pp. 130 – 147, 2006.

[8] J. D. Ullman, "Np-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, pp. 384–393, June 1975.

[9] S. Kumar, K. Dutta, and V. Mookerjee, "Maximizing business value by optimal assignment of jobs to resources in grid computing," *European Journal of Operational Research*, vol. 194, no. 3, pp. 856 – 872, 2009.

[10] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, pp. 53 –66, apr 1997.

[11] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel Distrib. Comput.*, vol. 47, pp. 8–22, November 1997.

[12] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *IEEE Heterogeneous Computing Workshop*, pp. 79–87, 1998.

[13] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, pp. 280–289, April 1977.

[14] C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and evaluation of a resource selection framework for grid applications," in *Proceedings of the International Symposium on High Performance Distributed Computing*, pp. 63– 69, 2002.

[15] X. He, X. Sun, and G. von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *J. Comput. Sci. Technol.*, vol. 18, pp. 442–451, July 2003.

[16] R. Min and M. Maheswaran, "Scheduling co-reservations with priorities in grid computing systems," in *Proceedings of the International Symposium on Cluster Computing and the Grid*, pp. 266–267, 2002.

[17] R. Buyya, M. Murshed, and D. Abramson, "A deadline and budget constrained cost-time optimisation algorithm for scheduling task farming applications on global grids," in *Conf. on Parallel and Distributed Processing Techniques and Applications*, 2001.

[18] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Workshop on Workflows in Support of Large-Scale Science*, pp. 1 –10, 2006.

[19] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *Proceedings of the First International Conference on e-Science and Grid Computing*, pp. 140–147, 2005.

[20] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems," *Management Science*, vol. 21, no. 12, pp. 1417–1427, 1975.

[21] R. Buyya, J. Giddy, and D. Abramson, "An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications," in *The Second Workshop on Active Middleware Services*, Kluwer Academic Press, 2000.

[22] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08)*, 2008.

[23] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pp. 196–203, 2004.