Cost-based Scheduling of Scientific Workflow Applications on Utility Grids

Jia Yu[†], Rajkumar Buyya[†] and Chen Khong Tham[‡]

[†] Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, VIC 3010 Australia
{jiayu, raj}@csse.unimelb.edu.au

[‡] Dept. of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260
eletck@nus.edu.sg

***Abstract*—over the last few years, Grid technologies have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on utility computing models. Users consume these services based on their QoS (Quality of Service) requirements. In such "pay-per-use" Grids, workflow execution cost must be considered during scheduling based on users' QoS constraints. In this paper, we propose a cost-based workflow scheduling algorithm that minimizes execution cost while meeting the deadline for delivering results. It can also adapt to the delays of service executions by rescheduling unexecuted tasks. We also attempt to optimally solve the task scheduling problem in branches with several sequential tasks by modeling the branch as a Markov Decision Process and using the value iteration method.**

I. INTRODUCTION

Utility computing [23] has emerged as a new service provision model and its services [10] are capable of supporting diverse applications including e-Business and e-Science over a global network. Users consume the services when they need to, and pay only for what they use. In the recent past, providing utility computing services has been reinforced by service-oriented Grid computing [2][12] that creates an infrastructure enabling users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment.

Many Grid applications such as bioinformatics and astronomy require workflow processing in which tasks are executed based on their control or data dependencies. As a result, a number of Grid workflow management systems with scheduling algorithms have been developed by several projects such as Condor DAGMan [22], Askalon [11], GrADS [8], ICENI [16], APST [3], and Pegasus [4][9]. They facilitate the execution of workflow applications and minimize their execution time on Grids. However, for imposing workflow paradigm on utility Grids, execution cost must also be considered when scheduling tasks on resources. For a utility service, pricing is dependent on the level of QoS offered such as the processing speed of the service. Typically, service providers charge higher prices for higher QoS. Therefore, users may not always need to complete workflows earlier than they require. Instead, they prefer to use cheaper services with lower QoS that are sufficient to meet their requirements.

Given this motivation, we present a cost-based workflow scheduling algorithm for time-critical workflow applications. The objective function of the proposed scheduling algorithm is to develop a workflow schedule such that it minimizes the execution cost and yet meets the time constraints imposed by the user. In order to solve scheduling problems efficiently for large-scale workflows, we partition workflow tasks and generate a workflow execution schedule based on the optimal schedules of the task partitions. Scheduling based on workflow partitions also allows the scheduler to re-compute some partial workflows during the workflow execution, when their initial schedules are violated. A deadline assignment strategy is also developed to distribute the overall deadline over each task partition. We also attempt to solve optimally the scheduling problem for sequential tasks by modeling the branch partition as a Markov Decision Process (MDP) [20], which has proven to be effective for modeling decision problems.

Several works have been proposed to address scheduling problems based on users' deadline constraint. Nimrod-G [6] schedules independent tasks for parameter-sweep applications to meet users' deadline. In contrast, the scheduling algorithm developed in the paper aims to schedule tasks with certain dependencies. A market-based workflow management system [13] locates an optimal bid based on the assigned deadline of each individual task. However, in most situations users may only want to specify a deadline for the entire workflow execution. Therefore, we focus on how to assign sub-deadlines of tasks to meet the overall deadline.

Proposed workflow scheduling approach can be used by both end-users and utility providers. End users can use the approach to orchestrate Grid services, whereas utility providers can outsource computing resources to meet customers' service-level requirements.

The remainder of the paper is organized as follows. Section II provides an overview of the workflow management system. We describe our workflow scheduling approach in Section III. Experimental details and simulation results are presented in Section IV. Finally, we conclude the paper with directions for further work in Section V.

II. WORKFLOW MANAGEMENT SYSTEM

Figure 1 shows the architecture of the workflow management system. Users first submit workflow

specifications with their QoS requirements. The system then discovers appropriate services for processing the workflow tasks and schedules the tasks on the services. There are three major steps in workflow scheduling: performance estimation, workflow planning and workflow execution with run-time rescheduling.

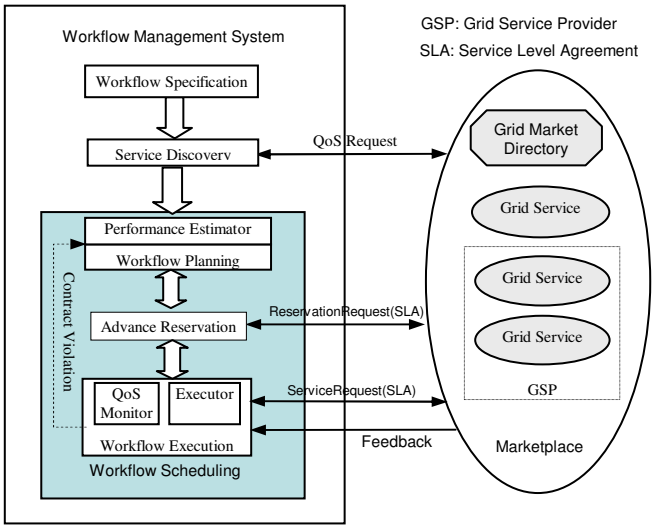


Fig. 1. Workflow management system architecture.

In order to plan in advance, the scheduler needs to predict task execution time on the available services. Different performance estimation approaches can be applied to different types of utility service. We classify existing utility services as either reservation-enabled resource or application services. Resource services provide proportions of hardware resources, such as computing processors, network resources, storage and memory, as a service for remote client access. Application services allow remote clients to use their specialized applications. To submit tasks to resource services, the scheduler needs to determine the number of resources and duration required to run tasks on the discovered services. The performance estimation for resource services can be achieved by using existing performance estimation techniques (e.g. analytical modeling [18], empirical [8] and historical data [14][19]) to predict task execution time on every discovered resource service. Unlike resource services, a reservation-enabled application service is capable of providing estimated service times based on the metadata of users' service requests [1]. As a result, the task execution time can be obtained by the application providers.

Workflow planning is to select a service and execution time slot for every task in the workflows based on users' QoS constraints, capability and availability of the services. The reservation manager makes reservations in advance to ensure the availability of the desired time slots for task execution. Candidate time slots are also generated during planning time for alternative reservations when a desired time slot is not available at the time of reservation.

At workflow execution time, the contract between a service provider and the workflow management system may be violated by many reasons such as resource failure. Therefore, rescheduling is deployed in the system to adapt to service dynamics and update the schedule. For example, if the desired start time of a task is delayed, the scheduler will adjust the reservation schedule for its unexecuted child tasks to compensate the delay.

III. A Cost-based Workflow Scheduling

The processing time and execution cost are two typical QoS constraints for executing workflows on "pay-per-use" services. The users normally would like to get the execution done at lowest possible cost within their required timeframe. In this section we present a cost-based workflow scheduling methodology and algorithm that allows the workflow management system to minimize the execution cost while delivering results within a certain deadline.

1) Problem Description and Methodology

We model workflow applications as a Directed Acyclic Graph (DAG). Let Γ be the finite set of tasks T_i $(1 \le i \le n)$. Let Λ be the set of directed arcs of the form (T_i, T_j) where T_i is called a parent task of T_j, and T_j the child task of T_i. We assume that a child task cannot be executed until all of its parent tasks are completed. Let D be the time constraint (deadline) specified by the users for workflow execution. Then, the workflow application can be described as a tuple $\Omega(\Gamma, \Lambda, D)$.

In a workflow graph, we call a task which does not have any parent task an *entry task* denoted as T_{entry} and a task which does not have any child task an *exit task* denoted as T_{exit}.

Let m be the total number of services available. There are a set of services S_i^j:cond $\equiv (1 \le i \le n,\ 1 \le j \le m_i,\ m_i \le m)$ is capable of executing the task T_i, but only one service can be assigned for the execution of a task. Services have varied processing capability delivered at different prices. We denote t_i^j as the sum of the processing time and data transmission time, and $\boldsymbol{c}_i^j$ as the sum of the service price and data transmission cost for processing T_i on service S_i^j.

The scheduling problem is to map every T_i onto some S_i^j to achieve minimum execution cost and complete the workflow execution within the deadline D. We solve the scheduling problem by following the divide-and-conquer technique and the methodology is listed below:

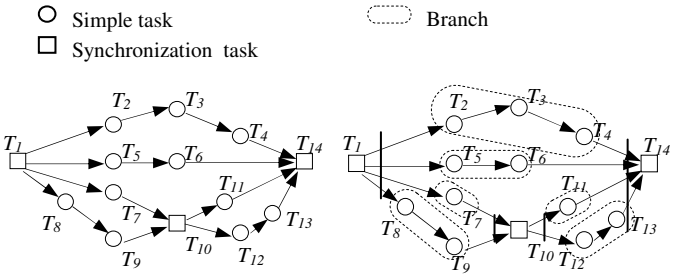
Step 1. Discover available services and predict execution time for every task.
Step 2. Group workflow tasks into task partitions.
Step 3. Distribute users' overall deadline into every task partition.
Step 4. Query available time slots, generate optimized schedule plan and make advance reservations based on the local optimal solution of every task partition.

Step 5. Start workflow execution and reschedule when the initial schedule is violated at run-time.

We provide details of steps 2-5 in the following sub-sections. The service discovery can be done by querying a directory service such as the Grid market directory [24] .

B. Workflow Task Partitioning

We categorize workflow tasks to be either a *synchronization task* or a *simple task*. A synchronization task is defined as a task which has more than one parent or child task. In Figure 7a, T_1 , T_{10} and T_{14} are synchronization tasks. Other tasks which have only one parent task and child task are simple tasks. In the example, $T_2 - T_9$ and $T_{11} - T_{13}$ are simple tasks.



(a) Before partitioning. (b) After partitioning.

Fig. 2. Workflow task partition.

Let a *branch* be a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. For example, the branches in Figure 2b are $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$, $\{T_7\}$, $\{T_8, T_9\}$, $\{T_{11}\}$ and $\{T_{12}, T_{13}\}$. We then partition workflow tasks Γ into independent branches $B_i (1 \le i \le k)$ and synchronization tasks $Y_i (1 \le i \le l)$, such that k and l are the total number of branches and synchronization tasks in the workflow respectively.

Let V be a set of nodes in a DAG corresponding to a set of task partitions $V_i (1 \le i \le k + l)$. Let E be the set of directed edges of the form (V_i, V_j) where V_i is a parent task partition of V_j and V_j is a child task partition of V_i . Then, a task partition graph is denoted as $G(V, E, D)$. A *simple path* (referred to as path) in G is a sequence of task partitions such that there is a directed edge from every task partition (in the path) to its child, where none of the vertices (task partitions) in the path is repeated.

A task partition V_i has four attributes: ready time ($rt[V_i]$), deadline ($dl[V_i]$), expected execution time ($eet[V_i]$). The earliest ready time of V_i is the earliest time the first task in it can be executed and it can be computed according to its parent partitions, $rt[V_i] = \max_{V_j \in P_i} dl[V_j]$, where P_i is the set of parent task partitions of V_i . The attributes are related as: $eet[V_i] = dl[V_i] - rt[V_i]$.

C. Deadline Assignment

After workflow task partitioning, we distribute the overall deadline between each V_i in G . The deadline $dl[V_i]$ assigned to any V_i is a *sub-deadline* of the overall deadline D . In this paper, we consider the following deadline assignment policies:

P1. *The cumulative sub-deadline of any independent path between two synchronization tasks must be same.*

A synchronization task cannot be executed until all tasks in its parent task partitions are completed. Thus, instead of waiting for other independent paths to be completed, a path capable of being finished earlier can be executed on slower but cheaper services. For example, the deadline assigned to $\{T_8, T_9\}$ is the same as $\{T_7\}$ in Figure 7. Similarly, deadlines assigned to $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$, and $\{\{T_7\}, \{T_{10}\}, \{T_{12}, T_{13}\}\}$ are same.

P2. *The cumulative sub-deadline of any path from $V_i (T_{entry} \in V_i)$ to $V_j (T_{exit} \in V_j)$ is equal to the overall deadline D .*

P2 assures that once every task partition is computed within its assigned deadline, the whole workflow execution can satisfy the user's required deadline.

P3. *Any assigned sub-deadline must be greater than or equal to the minimum processing time of the corresponding task partition.*

If the assigned sub-deadline is less than the minimum processing time of a task partition, its expected execution time will exceed the capability that its execution services can handle.

P4. *The overall deadline is divided over task partitions in proportion to their minimum processing time.*

The execution times of tasks in workflows vary; some tasks may only need 20 minutes to be completed, and some others may need at least one hour. Thus, the deadline distribution for a task partition should be based on its execution time. Since there are multiple possible processing times for every task, we use the minimum processing time to distribute the deadline.

We implemented deadline assignment policies on the task partition graph by combining Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms with critical path analysis to compute start times, proportion and sub-deadlines of every task partition.

D. Planning

The planning stage generates an optimized schedule for advance reservation and run-time execution. The scheduler allocates every workflow task to a selected service such that they can meet users' deadline at the lowest possible execution cost.

In general, mapping tasks on distributed services is an NP-hard problem. To model the entire workflow as an optimization problem will produce large scheduling overhead,

especially for the problem with two dimension constraints such as time and cost. Therefore, we solve the workflow scheduling problem by dividing the entire problem into several task partition scheduling problems. After deadline distribution, we can find a local optimal schedule for each partition based on its sub-deadline. If each local schedule guarantees that their task execution can be completed within the sub-deadline, the whole workflow execution will be completed within the overall deadline. Similarly, the result of the cost minimization solution for each task partition leads to an optimized cost solution for the entire workflow. Therefore, an optimized workflow schedule can be easily constructed by all local optimal schedules.

There are two types of task partitions: synchronization task and branch partition. The scheduling solutions for each type of partition and the overall algorithm are described in following sub-sections.

1) Synchronization Task Scheduling (STS)

For STS, the scheduler only considers one task to decide the service for executing that task. The objective function for scheduling of a synchronization task Y_i is:

$$\min c_i^j \text{, where } 1 \le j \le m_i \text{ and } t_i^j \le eet(Y_i)$$

The solution to a single task scheduling problem is simple. The optimal decision is to select the cheapest service that can process the task within the assigned sub-deadline.

2) Branch Task Scheduling (BTS)

If there is only one simple task in a branch, the solution for BTS is the same as STS. However, if there are multiple tasks, the scheduler needs to make a decision on which service to execute each task after the completion of its parent task. The optimal decision is to minimize the total execution cost of the branch and complete branch tasks within the assigned sub-deadline. The objective function for scheduling branch B_j is:

$$\min \sum_{T_i \in B_j} c_i^k \text{, where } 1 \le k \le m_i \text{ and } \sum_{T_i \in B_j} t_i^k \le eet(B_j)$$

BTS can be achieved by modeling the problem as a Markov Decision Process (MDP) [20], which has been shown to be effective for solving sequential decision problems.

3) MDP Model for Sequential Branch Tasks

The definition of our MDP model for scheduling branch B_i is described below:

States:

A Markov decision process is a state space S such that:

Definition 1: *A state $s \in S$ consists of current execution task, ready time RT and current location.*

Actions and transitions:

For every state s, there is a set of actions A_s. Actions incur immediate utility and affect the MDP to transit from one state to another.

Definition 2: *An action in the MDP is to allocate a time slot on a service to a task. There are two variables associated with each action a: input data transmission time plus the processing time of the service denoted as t and transmission cost plus the service cost denoted as c.*

Definition 3: *$u(s,a,s')$ is the immediate utility obtained from taking action a at state s and transitioning to state s'.*

$$u(s,a,s') = \begin{cases} \infty, & s'.RT > \text{sub-deadline} \\ a.c, & \text{otherwise} \end{cases}$$

Definition 4: *A transition incurred by an action from one state to another is deterministic, as services are utility services and can be reserved in advance.*

The MDP problem is to find an optimal policy π^* for all possible states. A policy is a mapping from s to a. Decision making for finding an optimal action for each state is not based on the immediate utility of the action but its expected utility, which is the sum of all the immediate utilities obtained as a result of decisions made for transiting from this state to a terminal state.

The value associated to each state represents the expected utility of this state in the MDP. This value is calculated recursively by using the value of successor states. The value of one state s is:

$$U(s) = \min_{a \in A_s} \{u(s,a,s') + U(s')\}$$

The best action for state s is:

$$\pi^*(s) = \arg\min_{a \in A_s} \{u(s,a,s') + U(s')\}$$

The optimal policy indicates the best services that should be assigned to execute branch tasks under a specific sub-deadline. The computation of the optimal policy can be solved by using a standard dynamic programming algorithm such as policy iteration and value iteration [20] (we have used value iteration here). Value iteration computes a new value function for each state based on the current value of its next state. Value iteration proceeds in an iterative fashion and can converge to the optimal solution quickly. The complexity analysis of value iteration can be found in [15]. By using dynamic programming, we can also record a number of candidate solutions while finding the optimal policy. Therefore, once the optimal time slot is rejected or not available, the scheduler can make another reservation immediately by using second optimal slot.

4) Scheduling Algorithm

Algorithm 1 shows the pseudo-code of the algorithm for planning an execution schedule. After acquiring the information about available services for each task, a task partition graph G is generated from the application graph Ω and overall deadline D is distributed over every partition in it. Then optimal schedules are computed for every partition in G level-by-level using either STS or BTS. We also found that after the optimization of one partition, there is an idle time between its expected completion time and assigned sub-

deadline. Instead of waiting, we adjust the assigned sub-deadlines of optimized partitions and the ready times of their child partitions.

E. Rescheduling

Run-time rescheduling is developed to adapt to dynamic situations such as delays of services and variations in availability of services due to failures, in order to complete workflows and satisfy users' requirements. The key idea of our rescheduling policy for handling an unexpected situation is to re-adjust sub-deadlines and re-compute optimal schedules for unexecuted task partitions. We also consider rescheduling the minimum number of tasks, since the scheduler need to cancel earlier reservations for tasks that need to be rescheduled to other services. Therefore, the scheduler re-computes unexecuted task partitions level-by-level. For example, if the execution of one task partition is delayed, the scheduler looks at its child task partitions. If the delay time can be accommodated by the child task partitions, rescheduling will not impact on its lower levels. Otherwise, the rest of the outstanding delay time is distributed further to successive task partitions of the child partitions.

In addition to handling task execution delay, the level-by-level task partition based approach can also be applied for managing other dynamic situations such as service unavailability and service policy change.

Algorithm 1 . Scheduling algorithm for cost optimization within users' deadline

Input: *A workflow graph* $\Omega(\Gamma,\Lambda,D)$

Output: *A schedule for all workflow tasks*

```
request processing time and price from available services for ∀T_i ∈ Γ
convert Ω into task partition graph G(V,E,D)
distribute deadline D over ∀V_i ∈ G
Repeat
    S ← get unscheduled task partitions whose parent task partitions
    have been scheduled
    for all i ∈ S do
        compute ready time of i
        query available time slots during ready time and sub-deadline
        on available services
        if i is a branch then
            compute an optimal schedule for i using BTS
        Else
            compute an optimal schedule for i using STS
        end if
        make advance reservations with desired services for all tasks in i
        adjust sub-deadline of i
    end for
until all partitions have been scheduled
```

IV. PERFORMANCE EVALUATION

We use GridSim [7][20] to simulate a Grid testbed for our experiments. Simulation facilitates evaluation as the same testbed environment can be repeated for different approaches. Figure 3 shows the simulation environment in which simulated services are discovered by querying GridSim Index Service (GIS) and every service is able to handle free slot query, reservation request and commitment.

We compare our proposed scheduling algorithm denoted as *Deadline-MDP* with two other scheduling approaches: *Greedy- Cost* and *Deadline-Level*. These two approaches are derived from the cost optimization algorithm in Nimrod-G, which is initially designed for scheduling independent tasks on Grids. Greedy-Cost sorts services by their prices and assigns as many tasks as possible to services without exceeding the deadline. Deadline-Level first divides workflow tasks into levels (based on their depth in the workflow graph), then divides the deadline by the number of levels and distribute the divided sub-deadlines over task levels. In Greedy-Cost, the deadlines of all tasks are the same as the overall deadline, whereas in Deadline-Level, tasks on the same level have the same sub-deadline.

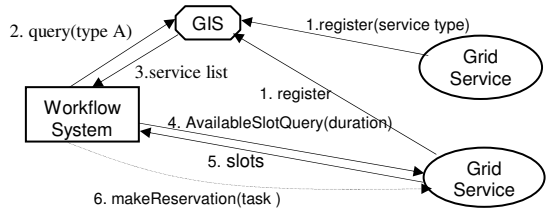
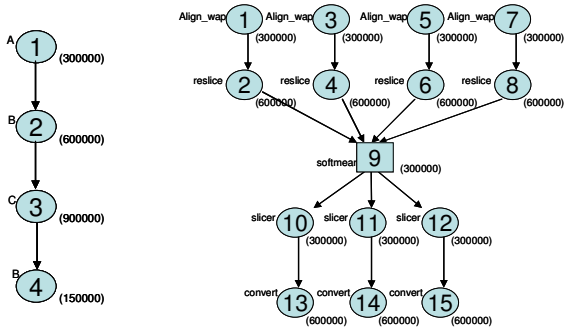
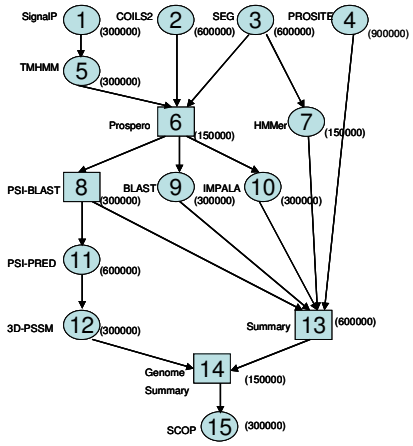


Fig. 3. Simulation environment.



a. Pipeline b. Parallel application (fMRI workflow [26])



c. Hybrid structure (protein annotation workflow [5])

Fig. 4. Workflow applications. The label on the left of a task denotes the required service type. The number in brackets represents the length of the task in MI.

We simulate three common workflow structures in scientific workflow applications for our experiments: pipeline, parallel and hybrid. A pipeline application (see Fig. 4a) executes a number of tasks in a single sequential order. A parallel application (see Fig. 4b) requires multiple pipelines to be executed in parallel. For example, in Fig. 4b, there are 4 pipelines (1-2, 3-4, 5-6 and 7-8) before task 9. A hybrid structure application (see Fig. 4c) is a combination of both parallel and sequential applications. In our experiments, we select a neuro-science workflow [26] for our parallel application and a protein annotation workflow [5] developed by London e-Science Centre for our hybrid workflow structure application.

As execution requirements for tasks in scientific workflows are heterogeneous, we use service type to represent different type of service. Every task in our experimental workflow applications requires a certain type of service. For example, task 1, 3, 5, 7 in parallel application requires service type *Align_wap* and task 2, 4, 6 and 8 requires *reslice*. In the simulation, we use MI (million instructions) to represent the length of tasks and use MIPS (Million Instructions per Second) to represent the processing capability of services. We simulate 15 types of services, each supported by 10 different service providers with different processing capability. The values of MIPS for services range from 100 to 5000 and the value of MI for each task is indicated in bracket next to the task in Figure 4.

In the experiments, every task in the workflows generates output data required by its child tasks as inputs. The data need to be staged out from the task processing node and staged into the processing node of its child tasks. The I/O data of the workflows range from 10MB to 1024 MB. The available network bandwidths between services are 100Mbps, 200Mbps, 512Mbps and 1024Mbps.

For our experiments, the cost that a user needs to pay for a workflow execution comprises of two parts: processing cost and data transmission cost. Table I shows an example of processing cost, while Table II shows an example of data transmission cost. It can be seen that the processing cost and transmission cost are inversely proportional to its processing time and transmission time respectively.

Table I. Service speed and corresponding price for executing a task.

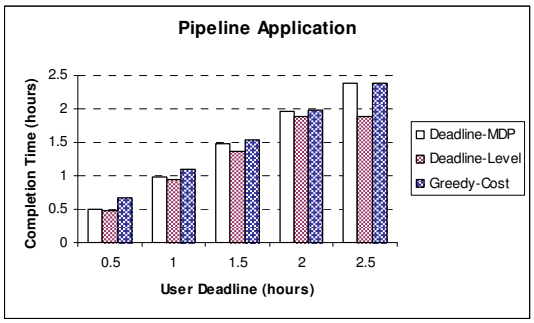
Service ID	Processing Time (sec)	Cost (G$)
1	1200	300
2	600	600
3	400	900
4	300	1200

Table II. Transmission bandwidth and corresponding price.

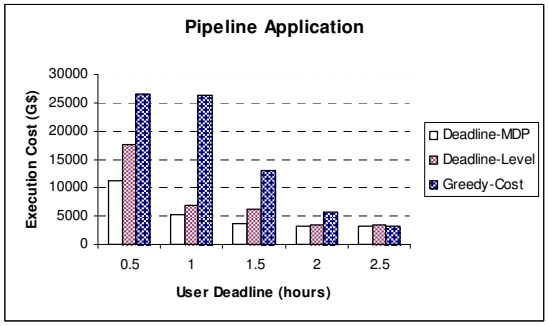
Bandwidth (Mbps)	Cost/sec (G$/sec)
100	1
200	2
512	5.12
1024	10.24

The two metrics used to evaluate the scheduling approaches are time constraint and execution cost. The former indicates whether the schedule produced by the scheduling approach meets the required deadline, while the latter indicates how much it costs to schedule the workflow tasks on the testbed.

Figure 5 to 7 compare the execution time and cost of using Deadline-MDP, Deadline-Level and Greedy-Cost for scheduling pipeline, parallel and hybrid structure applications with deadline 0.5, 1, 1.5, 2, and 2.5 hours respectively. It can be seen that Greedy-Cost does not guarantee that users' deadlines can be met, whereas both Deadline-MDP and Deadline-Level can meet deadlines. Greedy-Cost also incurs significantly higher execution costs even though it takes longer time to complete executions. This is because it attempts to meet the deadline by employing faster but more expensive services as the deadline approaches.

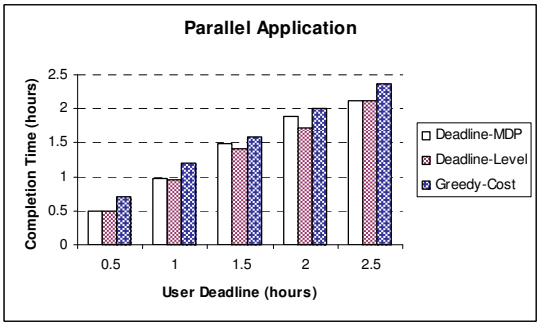


a . Execution time of three approaches.

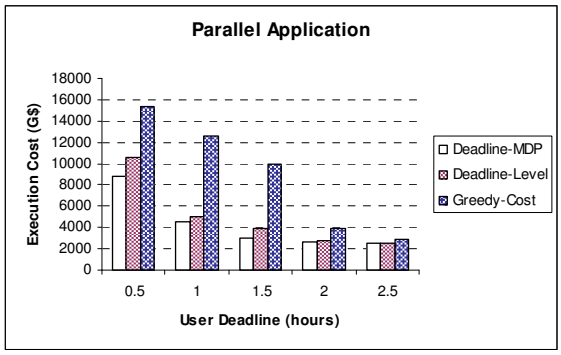


b. Execution cost of three approaches.

Fig. 5 Execution time and cost using three approaches for scheduling the pipeline application.

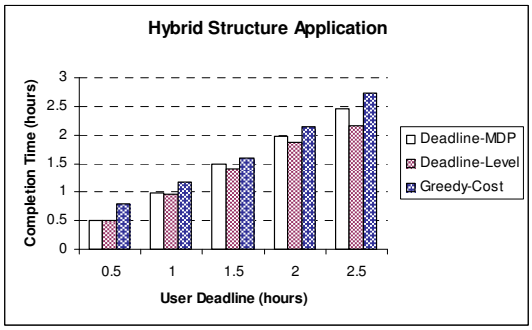


a. Execution time of three approaches.

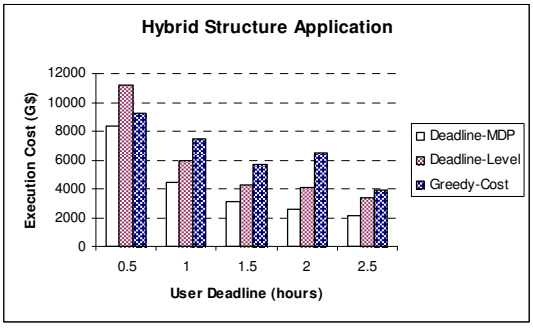


b. Execution time of three approaches.

Fig. 6. Execution time and cost using three approaches for scheduling the parallel application.

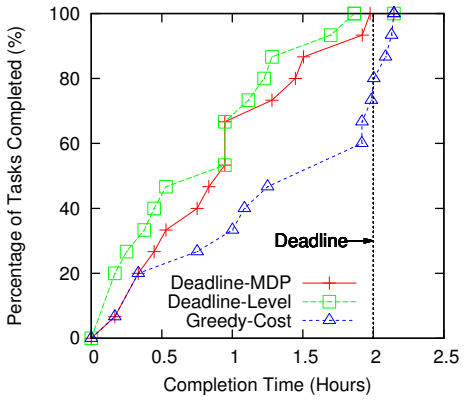


a. Execution time of three approaches.

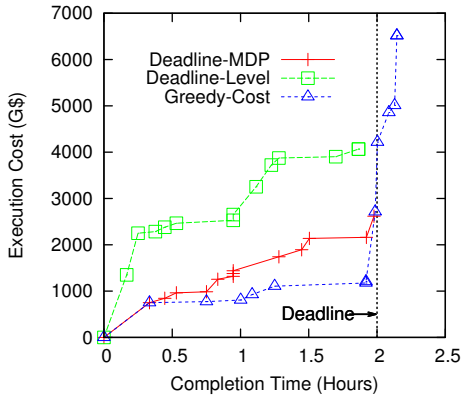


b. Execution cost of three approaches.

Fig. 7. Execution time and cost using three approaches for scheduling the hybrid structure application.



a. Percentage of tasks completed.

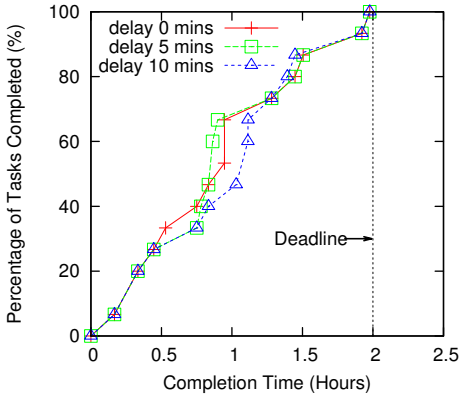


b. Execution Cost.

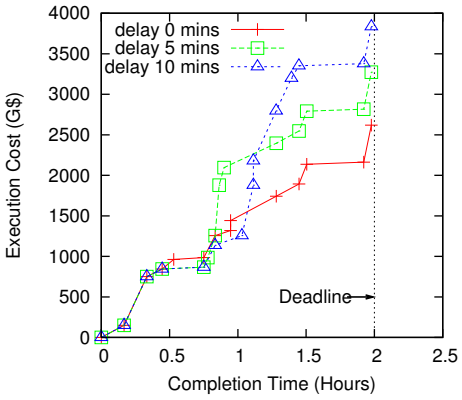
Fig.8. Deadline-MDP, Deadline-Level, and Greedy-Cost scheduling for Hybrid Structure Application.

Both Deadline-Level and Deadline-MDP can meet the deadlines, but Deadline-MDP spends much less cost for shorter deadlines. Deadline-MDP can achieve that for the pipeline application by modeling the entire pipeline application as one MDP process, such that it can find an optimal path among the cheapest services to execute tasks and transfer input/output data. Similarly for the parallel application, it also optimizes the cost for branches within the parallel application.

For the hybrid structure application, Deadline-MDP performs better as it assigns sub-deadlines to tasks based on their dependencies and estimated execution times. Deadline-Level assigns sub-deadlines only based on the task level and thus incurs unnecessary cost. This is because parent tasks that are completed earlier using faster but more expensive services still need to wait for other slower parent tasks to be completed before their child tasks can start execution. This shows that it is important to consider task dependencies as it is pointless to employ expensive services if not required.



a. Percentage of tasks completed.



b. Execution cost.

Fig. 9. Deadline-MDP scheduling with delay of 0, 5, and 10 minutes for task 6 in Hybrid Structure Application.

Figure 8 shows the percentage of tasks completed and execution cost for the hybrid structure application at various times. We can see that Deadline-Level attempts to complete tasks earlier by using more expensive but faster services. However its final completion time is only marginally lower than that of Deadline-MDP. On the other hand, Deadline-MDP selects much cheaper services than Deadline-Level but can still meet the final deadline. In contrast, Greedy-Cost chooses cheaper services and completes tasks slowly at the early stages of the execution such that it does not have enough time to complete the partially remaining tasks before the deadline, even though it selects more expensive services to speed up execution during the final stage.

We now evaluate the rescheduling approach in Deadline-MDP. Figure 9 shows how Deadline-MDP performs with delays of 0, 5 and 10 minutes for the execution of task 6 in the hybrid structure application. For the delays of 5 minutes and 10 minutes, Deadline-MDP can still meet the deadline by rescheduling the partially remaining unexecuted tasks. However, the execution cost increases for longer delays since the scheduler switches the remaining tasks to more expensive services in order to complete the remaining execution within the deadline.

VI. Conclusion And Future Work

Utility Grids enable users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment. Users are required to pay for access services based on their usage and the level of QoS provided. Therefore, workflow execution cost must be considered during scheduling. In this paper, we proposed a cost-based workflow scheduling algorithm that minimizes the cost of execution while meeting the deadline. We also described task partitioning and overall deadline assignment for optimized execution planning and efficient run-time rescheduling. We have used a Markov Decision Process approach to schedule sequential workflow task execution, such that it can find the optimal path among services to execute tasks and transfer input/output data. The experimental results demonstrate that the proposed scheduling approach can meet users' deadline whilst spending less cost. It can also adapt to the delays of service executions by rescheduling unexecuted tasks to meet users' deadlines. In future, we will further enhance our scheduling method to support multiple service negotiation models.

Acknowledgments

We would like to thank Hussein Gibbins, Chee Shin Yeo, Srikumar Venugopal, Sushant Goel, Tianchi Ma and Arun Konagurthu for their comments on this paper. We also want to thank Anthony Sulistio for providing reservation infrastructure in GridSim. This work is partially supported through StorageTek Fellowship and Australian Research Council (ARC) Discovery Project grant.

References

[1] S. Benkner et al., "GEMSS: Grid-infrastructure for Medical Service Provision", In *HealthGrid 2004 Conference*, 29th-30th Jan. 2004, Clermont-Ferrand, France.

[2] S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt, "VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing", In *the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, PA, USA, November 2004.

[3] A. Birnbaum et al., "Grid workflow software for High-Throughput Proteome Annotation Pipeline", In 1st *International Workshop on Life Science Grid (LSGRID2004)*, Ishikawa, Japan, June 2004.

[4] J. Blythe et al., "Task Scheduling Strategies for Workflow-based Applications in Grids", In *IEEE International Symposium on Cluster Computing and Grid (CCGrid)*, 2005.

[5] 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.

[6] 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 *2nd Workshop on Active Middleware Services (AMS 2000)*, Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.

[7] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", *Concurrency and Computation: Practice and Experience,* Vol. 14(13-15):1175-1220, Wiley Press, USA, 2002.

[8] K. Cooper et al., "New Grid Scheduling and Rescheduling Methods in the GrADS Project", In *NSF Next Generation Software Workshop*, International Parallel and Distributed Processing Symposium, Santa Fe, IEEE CS Press, Los Alamitos, CA, USA, April 2004.

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

[10] T. Eilam et al., "A utility computing framework to develop utility systems", *IBM System Journal*, Vol. 43(1):97-120, 2004.

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

[12] I. Foster et al., "The Physiology of the Grid", Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[13] A. Geppert, M. Kradolfer, and D. Tombros, "Market-based Workflow Management", *International Journal of Cooperative Information Systems,* World Scientific Publishing Co., NJ, USA, 1998.

[14] S. Jang et al., "Using Performance Prediction to Allocate Grid Resources". Technical Report 2004-25, GriPhyN Project, USA.

[15] O. Madani, "Polynomial Value Iteration Algorithms for Deterministic MDPs", In *18th Conference on Uncertainty in Artificial Intelligence*, August, 2002.

[16] A. Mayer et al., "ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time", In *UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, September 2003.

[17] S. Newhouse, "Grid Economy Services Architecture (GESA)", *Grid Economic Services Architecture WG*, Global Grid Forum, 2003.

[18] G. R. Nudd et al., "PACE- A Toolset for the performance Prediction of Parallel and Distributed Systems", *International Journal of High Performance Computing Applications (JHPCA)*, Special Issues on Performance Modelling- Part I, 14(3): 228-251, SAGE Publications Inc., London, UK, 2000.

[19] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information", In *Workshop on Job Scheduling Strategies for Parallel Processing*, 12th International Parallel Processing Symposium & 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98), IEEE Computer Society Press, Los Alamitos, CA, USA, 1998.

[20] A. Sulistio and R. Buyya, "A Grid Simulation Infrastructure Supporting Advance Reservation", In *16th International Conference on Parallel and Distributed Computing and Systems* (*PDCS 2004),* November 9-11, 2004, MIT Cambridge, Boston, USA.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[22] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor - A Distributed Job Scheduler", *Beowulf Cluster Computing with Linux*, The MIT Press, MA, USA, 2002.

[23] G. Thickins, "Utility Computing: The Next New IT Model", *Darwin Magazine*, April 2003.

[24] J. Yu, S. Venugopal, and R. Buyya, "A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services", *Journal of Supercomputing*, Kluwer Academic Publishers, USA, 2005.

[25] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005.

[26] 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.