19

DEADLINE/BUDGET-BASED SCHEDULING OF WORKFLOWS ON UTILITY GRIDS

JIA YU, KOTAGIRI RAMAMOHANARAO, AND RAJKUMAR BUYYA

Grid technologies provide the basis for creating a service-oriented paradigm that enables a new way of service provisioning based on utility computing models. For typical utility computing-based services, users are charged for consuming services on the basis of their usage and QoS level required. Therefore, while scheduling workflows on utility Grids, service price must be considered while optimizing the execution performance.

In this chapter, the characteristics of utility Grids and the corresponding scheduling problems are discussed, followed by descriptions of two scheduling heuristics based on two QoS constraints: deadline and budget. Two different workflow structures and experiment settings are also presented for evaluation of the proposed scheduling heuristics.

19.1 INTRODUCTION

Utility computing [19] has emerged as a new service provisioning model [6] and is capable of supporting diverse computing services such as servers, storage, network, and applications for e-business and e-science over a global network. For utility computing-based services, users consume required services, and pay only for what they use. With economic incentive, utility computing encourages organizations to

Market-Oriented Grid and Utility Computing Edited by Rajkumar Buyya and Kris Bubendorfer

TABLE 19.1 Community Grids versus Utility Grids

Attributes	Community Grids	Utility Grids
Availability	Best effort	Advanced reservation
QoS	Best effort	Contract/service-level Agreement (SLA)
Pricing	Not considered or free access	Usage, QoS level, market supply and demand

offer their specialized applications and other computing utilities as services so that other individuals/organizations can access these resources remotely. Therefore, it facilitates individuals/organizations to develop their own core activities without maintaining and developing fundamental computing infrastructure. In the more recent past, providing utility computing services has been reinforced by service-oriented Grid computing, which creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable, and standard worldwide network environment.

Table 19.1 shows different aspects between community Grids and utility Grids in terms of availability, quality of services (QoS), and pricing. In utility Grids, users can make a reservation with a service provider in advance to ensure service availability, and users can also negotiate service level agreements with service providers for the required QoS. Compared with utility Grids, service availability and QoS in community Grids may not be guaranteed. However, community Grids provide access based on mutual agreement driven by partnership (LHCGrid [23]) or free access (e.g., SETI@Home [24]), whereas in utility Grids users need to pay for service access. In general, the service pricing in utility Grids is based on the QoS level expected by users and market supply and demand for services.

Typically, service providers charge higher prices for higher QoS. Therefore users do not always need to complete workflows earlier than they require. They sometimes prefer to use cheaper services with a lower QoS that is sufficient to meet their requirements. Given this motivation, cost-based workflow scheduling is developed to schedule workflow tasks on utility Grids according to users' QoS constraints such as deadline and budget.

19.2 GRID WORKFLOW MANAGEMENT SYSTEM

Scientific communities such as high-energy physics, gravitational-wave physics, geophysics, astronomy, and bioinformatics, are utilizing Grids to share, manage, and process large datasets [18]. In order to support complex scientific experiments, distributed resources such as computational devices, data, applications, and scientific instruments need to be orchestrated while managing the application operations within Grid environments [13]. A workflow expresses an automation of procedures wherein files and data are passed between procedures applications according to a defined set of rules, to achieve an overall goal [10]. A workflow management system defines, manages, and executes workflows on computing resources. The use of the workflow paradigm for application composition on Grids offers several advantages [17], such as

- Ability to build dynamic applications and orchestrate the use of distributed resources
- Utilization of resources that are located in a suitable domain to increase throughput or reduce execution costs
- Execution spanning multiple administrative domains to obtain specific processing capabilities
- Integration of multiple teams involved in managing different parts of the experiment workflow—thus promoting interorganizational collaborations

Realizing workflow management for Grid computing requires a number of challenges to be overcome. They include workflow application modeling, workflow scheduling, resource discovery, information services, data management, and fault management. However, from the users' perspective, two important barriers that need to be overcome are (1) the complexity of developing and deploying workflow applications and (2) their scheduling on heterogeneous and distributed resources to enhance the utility of resources and meet user quality-of-service (QoS) demands.

19.3 WORKFLOW SCHEDULING

Workflow scheduling is one of the key issues in the management of workflow execution. Scheduling is a process that maps and manages execution of interdependent tasks on distributed resources. It introduces allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. Proper scheduling can have a significant impact on the performance of the system. In general, the problem of mapping tasks on distributed resources belongs to a class of problems known as "NP-hard problems" [8]. For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Even though the workflow scheduling problem can be solved by using exhaustive search, the time taken for generating the solution is very high. Scheduling decisions must be made in the shortest time possible in Grid environments, because there are many users competing for resources, so timeslots desired by one user could be taken by another user at any moment.

A number of *best-effort* scheduling heuristics [2,12,21] such as min–min (minimum–minimum) and heterogeneous earliest finish time (HEFT) have been developed and applied to schedule Grid workflows. These best-effort scheduling algorithms attempt to complete execution within the shortest time possible. They neither have any provision for users to specify their QoS requirements nor any specific support to meet them. However, many workflow applications in both scientific and business domains require some certain assurance of QoS (see Fig. 19.1). For example, a workflow application for maxillofacial surgery planning [10] needs results to be delivered before a certain time. For these applications, the workflow scheduling applied should be able

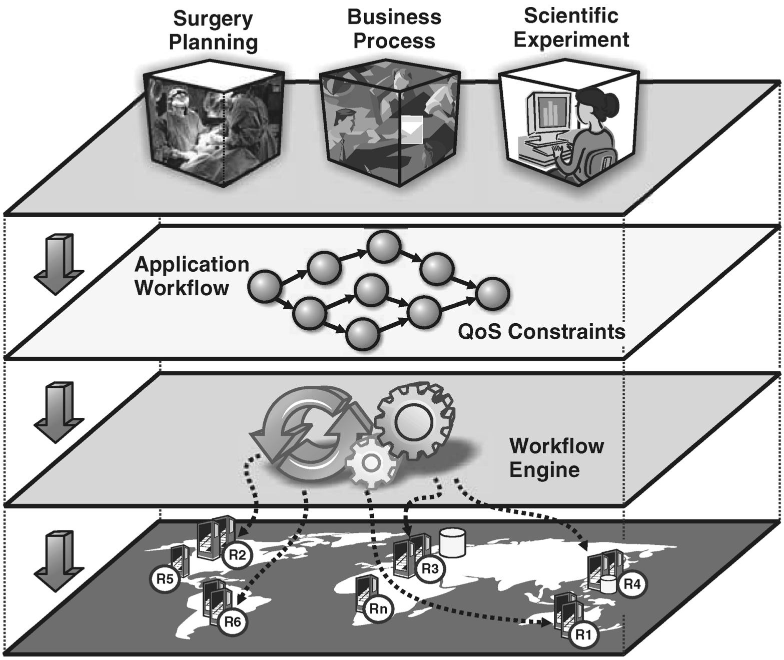


Figure 19.1 A high-level view of a Grid workflow execution environment.

to analyze users' QoS requirements and map workflow tasks onto suitable resources such that the workflow execution can be completed to satisfy their requirements.

Several new challenges are presented while scheduling workflows with QoS constraints for Grid computing. A Grid environment consists of large number of resources owned by different organizations or providers with varying functionalities and able to guarantee differing QoS levels. Unlike best-effort scheduling algorithms, which consider only one factor (e.g., execution time), multiple criteria must be considered to optimize the execution performance of QoS constrained workflows. In addition to execution time, monetary execution cost is also an important factor that determines quality of scheduling algorithms, because service providers may charge differently for different levels of QoS [3]. Therefore, a scheduler cannot always assign tasks onto services with the highest QoS levels. Instead, it may use cheaper services with lower QoS that is sufficient to meet the requirements of the users.

Also, completing the execution within a specified QoS (e.g., time and budget) constraints depends not only on the global scheduling decision of the workflow scheduler but also on the local resource allocation model of each execution site. If the execution of every single task in the workflow cannot be completed as expected by the scheduler, it is impossible to guarantee QoS levels for the entire workflow. Therefore, schedulers should be able to interact with service providers to ensure resource availability and QoS levels. It is required that the scheduler be able to determine QoS requirements for each task and negotiate with service providers to establish a service-level agreement (SLA), which is a contract specifying the minimum expectations and obligations between service providers and consumers.

This chapter presents a number of workflow scheduling algorithms based on QoS constraints such as deadline and budget while taking into account the costs and capabilities of Grid services.

19.4 QoS-BASED WORKFLOW SCHEDULING

19.4.1 Problem Description

A workflow application can be modeled as a directed acyclic graph (DAG). Let Γ be the finite set of tasks $T_i (1 \leq i \leq n)$. Let Λ be the set of directed edges. Each edge is denoted by (T_i, T_j), where T_i is an immediate parent task of T_j and T_j is the immediate child task of T_i. A child task cannot be executed until all of its parent tasks have been completed. There is a transmission time and cost associated with each edge. In a workflow graph, a task that does not have any parent task is called an *entry task*, denoted as T_{entry}, and a task that does not have any child task is called an *exit task*, denoted as T_{exit}. In this thesis, we assume there is only one T_{entry} and T_{exit} in the workflow graph. If there are multiple entry tasks and exit tasks in a workflow, we can connect them to a zero-cost pseudoentry or exit task.

The execution requirements for tasks in a workflow could be heterogeneous. A service may be able to execute some of workflow tasks. The set of services capable of executing task T_i is denoted as S_i, and each task is assigned for execution on only one service. Services have varied processing capability delivered at different prices. The task runtimes on all service and input data transfer times are assumed to be known. The estimation of task runtime is presented in Section 19.4.2. The data transfer time can be computed using bandwidth and latency information between the services. t_i^j is the sum of the processing time and input data transmission time, and c_i^j is the sum of the service price and data transmission cost for processing T_i on service $s_i^j (1 \leq j \leq |S_i|)$.

Let B be the cost constraint (budget) and D be the time constraint (deadline) specified by a user for workflow execution. The budget-constrained scheduling problem is to map every T_i onto a suitable service to minimize the execution time of the workflow and complete it with the total cost less than B. The deadline constrained scheduling problem is to map every T_i onto a suitable service to minimize the execution cost of the workflow and complete it before deadline D.

19.4.2 Performance Estimation

Performance estimation is the prediction of performance of task execution on services and is crucial for generating an efficient schedule for advance reservations. Different performance estimation approaches can be applied to different types of utility services. We classify existing utility services as either resource services, application services, or information service.

Resource services provide hardware resources such as computing processors, network resources, storage, and memory, as a service for remote clients. 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, empirical and historical data) to predict task execution time on every discovered resource service.

Application services allow remote clients to use their specialized applications, while *information services* provide information for the users. Unlike resource services, application and information services are capable of providing estimated service times on the basis of the metadata of users' service requests. As a result, the task execution time can be obtained by the providers.

19.5 WORKFLOW SCHEDULING HEURISTICS

Workflow scheduling focuses on mapping and managing the execution of interdependent tasks on diverse utility services. In this section, two heuristics are provided as a baseline for cost-based workflow scheduling problems. The heuristics follow the divide-and-conquer technique, and the steps in the methodology are listed below:

1. Discover available services and predict execution time for every task.
2. Distribute users' overall deadline or budget into every task.
3. Query available timeslots, generate an optimized schedule plan, and make advance reservations on the basis of the local optimal solution of every task partition.

19.5.1 Deadline-Constrained Scheduling

The proposed deadline constrained scheduling heuristic is called *greedy cost–time distribution* (GreedyCost-TD). In order to produce an efficient schedule, GreedyCost-TD groups workflow tasks into task partitions and assigns subdeadlines to each task according to their workload and dependences. At runtime, a task is scheduled on a service, which is able to complete it within its assigned subdeadline at the lowest cost.

19.5.1.1 Workflow Task Partitioning. In workflow task partitioning, workflow tasks are first categorized as either synchronization tasks or simple tasks. A *synchronization task* is defined as a task that has more than one parent or child task. In Figure 19.2, T_1, T_{10}, and T_{14} are synchronization tasks. Other tasks that have only one parent task and child task are simple tasks.

In Figure 19.2, T_2-T_9 and $T_{11}-T_{13}$ are simple tasks. Simple tasks are then clustered into a *branch*, which is a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. For example, the branches in Figure 19.1b are $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$,$\{T_7\}$, $\{T_8, T_9\}$, $\{T_{11}\}$, and $\{T_{12}, T_{13}\}$.

After task partitioning, workflow tasks Γ are then clustered into partitions. As shown in Figure 19.2b, a *partition* $V_i (1 \leq i \leq k)$ is either a branch or a set of one

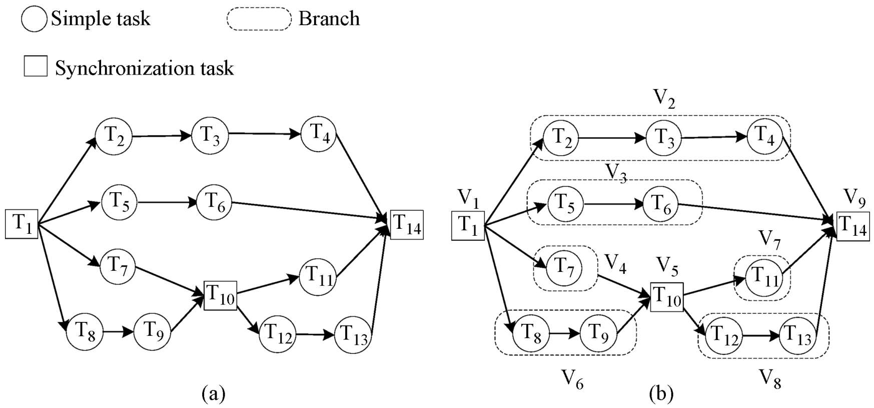


Figure 19.2 Workflow task partition: (a) before partitioning; (b) after partitioning.

synchronization task, where k is the total number of branches and synchronization tasks in the workflow. For a given workflow $\Omega(\Gamma, \Lambda)$, the corresponding task partition graph $\Omega'(\Gamma', \Lambda')$ is created as follows

$$\begin{cases} \Gamma' = \{V_i | V_i \text{ is a partition in } \Omega, \text{ where } 1 \le i \le k\} \\ \Lambda' = \{(v_i, w) | v \in V_i, w \in V_j, \text{ and } (v, w) \in \Lambda\} \end{cases}$$

where Γ' is a set of task partitions and Λ' is the set of directed edges of the form $(V_i,\ V_j)$, and V_i is a parent task partition of V_j and V_j is a child task partition of V_i.

19.5.1.2 Deadline Distribution. After workflow task partitioning, the overall deadline is distributed over each V_i in Ω'. The deadline assigned to any V_i is called a *subdeadline* of the overall deadline D. The deadline assignment strategy considers the following facts:

P1. *The cumulative expected execution time of any simple path between two synchronization tasks must be the same.* A simple path in Ω' is a sequence of task partitions such that there is a directed edge from every task partition (in the simple path) to its child, where none of the task partitions in the simple path is repeated. For example, $\{V_1, V_3, V_9\}$ and $\{V_1, V_6, V_5, V_7, V_9\}$ are two simple paths between V_1 and V_9. A synchronization task cannot be executed until all tasks in its parent task partitions are completed. Thus, instead of waiting for other simple 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 19.2. Similarly, deadlines assigned to $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$, and $\{\{T_7\}, \{T_{10}\}, \{T_{12}, T_{13}\}\}$ are the same.

P2. *The cumulative expected execution time of any path from $V_i(T_{\text{entry}} \in V_i)$ to $V_j(T_{\text{exit}} \in V_j)$ is equal to the overall deadline D.* P2 ensures 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 subdeadline must be greater than or equal to the minimum processing time of the corresponding task partition.* If the assigned subdeadline is less than the minimum processing time of a task partition, its expected execution time will exceed the capability that its execution services can provide.

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 min to be completed, and 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.

The deadline assignment strategy on the task partition graph is implemented by combining breadth-first search (BFS) and depth-first search (DFS) algorithms with critical-path analysis to compute start times, proportion, and deadlines of every task partition. After distributing overall deadline into task partitions, each task partition is assigned a deadline. There are three attributes associated with a task partition V_i: deadline ($\mathrm{dl}[V_i]$), ready time ($\mathrm{rt}[V_i]$), and expected execution time ($\mathrm{eet}[V_i]$). The ready time of V_i is the earliest time when its first task can be executed. It can be computed according to its parent partitions and defined by

$$\mathrm{rt}[V_i] = \begin{cases} 0 & T_{\mathrm{entry}} \in V_i \\ \max\limits_{V_j \in PV_i} \mathrm{dl}[V_j] & \text{otherwise} \end{cases} \tag{19.1}$$

where PV_i is the set of parent task partitions of V_i. The relation between three attributes of a task partition V_i follows that

$$\mathrm{eet}[V_i] = \mathrm{dl}[V_i] - \mathrm{rt}[V_i] \tag{19.2}$$

After deadline distribution over task partitions, a subdeadline is assigned to each task. If the task is a synchronization task, its subdeadline is equal to the deadline of its task partition. However, if a task is a simple task of a branch, its subdeadline is assigned by dividing the deadline of its partition according to its processing time. Let P_i be the set of parent tasks of T_i. The assigned deadline of task T_i in partition V is defined by

$$\mathrm{dl}[T_i] = \mathrm{eet}[T_i] + \mathrm{rt}[V] \tag{19.3}$$

where

$$\mathrm{eet}[T_i] = \frac{\min\limits_{1 \le j \le |S_i|} t_i^j}{\sum\limits_{T_k \in V} \min\limits_{1 \le l \le |S_k|} t_k^l} \times \mathrm{eet}[V] \tag{19.4}$$

$$\mathrm{rt}[T_i] = \begin{cases} 0 & T_i = T_{\mathrm{entry}} \\ \max\limits_{T_j \in P_i} \mathrm{dl}[T_j] & \text{otherwise} \end{cases} \tag{19.5}$$

19.5.1.3 Greedy Cost–Time Distribution (TD). Once each task has its own subdeadline, a local optimal schedule can be generated for each task. If each local schedule guarantees that their task execution can be completed within their subdeadline, the whole workflow execution will be completed within the overall deadline. Similarly, the result of the cost minimization solution for each task leads to an optimized cost solution for the entire workflow. Therefore, an optimized workflow schedule can be constructed from all local optimal schedules. The schedule allocates every workflow task to a selected service such that they can meet its assigned subdeadline at low execution cost. Let $\mathrm{Cost}(T_i)$ be the sum of data transmission cost and service cost for processing T_i. The objective for scheduling task T_i is

$$\begin{aligned} \text{Minimize } \mathrm{Cost}(T_i) &= \min_{1 \le j \le |S_i|} c_i^j \\ \text{subject to } t_i^j &\le \mathrm{eet}[T_i] \end{aligned} \tag{19.6}$$

The details of GreedyCost-TD heuristic are presented in Algorithm 19.1. It first partitions workflow tasks and distributes overall deadline over each task partition and

Input: *A workflow graph* $\Omega(\Gamma, \Lambda)$, *deadline* D
Output: *A schedule for all workflow tasks*

```
Request processing time and price from available services for
∀T ∈ Γ
Convert Ω into task partition graph Ω′(Γ′, Λ′)
Distribute deadline D over ∀V_i ∈ Γ′ and assign a sub-deadline to
each task
Put the entry task into ready task queue Q
while there are ready tasks in Q do
    Sort all tasks in Q
    T_i ← the first task from Q
    Compute ready time of T_i
    Query available time slots during ready time and
    sub-deadline
    S ← a service which meets Equation 4-6
    if S = φ then
        S ← S_i^j such that j = arg min_{1≤j≤|S_i|} t_i^j
    end if
    Make advance reservations of T_i on S
    Put ready child tasks into Q whose parent tasks have been
    scheduled
end while
```

Algorithm 19.1 Greedy cost–time distribution heuristic.

then divides the deadline of each partition into each single task. Unscheduled tasks are queued in the ready queue waiting to be scheduled. The order of tasks in the ready queue is sorted by a ranking strategy that is described in Section 19.5.3. The scheduler schedules tasks in the ready queue one by one. The entry task is the first task that is put into the ready task and scheduled. Once all parents of a task have been scheduled, the task becomes ready for scheduling and is put into the ready queue (line 15). The ready time of each task is computed at the time it is scheduled (line 9). After obtaining all available timeslots (line 9) on the basis of the ready time and subdeadline of current scheduling tasks, the task is scheduled on a service that can meet the scheduling objective. If no such service is available, the service that can complete the task at earliest time is selected to satisfy the overall time constraint (lines 11–12).

19.5.2 Budget-Constrained Scheduling

The proposed budget-constrained scheduling heuristic is called *greedy time–cost distribution* (GreedyTime-CD). It distributes portions of the overall budget to each task in the workflow. At runtime, a task is scheduled on a service that is able to complete it with less cost than its assigned subbudget at the earliest time.

19.5.2.1 Budget Distribution. The budget distribution process is to distribute the overall budget over tasks. In the budget distribution, both workload and dependencies between tasks are considered when assigning subbudgets to tasks. There are two major steps:

Step 1: Assigning Portions of the Overall Budget to Each Task. In this step, an initial subbudget is assigned to tasks according to their average execution and data transmission cost. In a workflow, tasks may require different types of services with various price rates, and their computational workload and required I/O data transmission may vary. Therefore, the portion of the overall budget each task obtains should be based on the proportion of their expense requirements. Since there are multiple possible services and data links for executing a task, their average cost values are used for measuring their expense requirements. The expected budget for task T_i is defined by

$$\mathrm{eec}[T_i] = \frac{\mathrm{avgCost}[T_i]}{\sum\limits_{T_i \in \Gamma} \mathrm{avgCost}[T_i]} \times B \tag{19.7}$$

where

$$\mathrm{avgCost}[T_i] = \frac{\sum\limits_{1 \le j \le |S_i|} c_i^j}{|S_i|}$$

Step 2: Adjusting Initial Subbudget Assigned to Each Task by Considering Their Task Dependences. The subbudget of a task assigned in the first step is based only on its average cost without considering its execution time. However, some tasks could be completed at earliest time using more expensive services based on their local budget, but its child tasks cannot start execution until other parent tasks have been completed. Therefore, it is necessary to consider task dependences for assigning a subbudget to a task. In the second step, the initial assigned subbudgets of tasks are adjusted. It first computes the approximate execution time on the basis of its initial expected execution budget and its unit time per cost so that the approximate start and end times of each task partition can be calculated. The approximate execution time of task T_i is defined by

$$\mathrm{aet}[T_i] = \mathrm{eec}[T_i] \times \frac{\mathrm{avgTime}[T_i]}{\mathrm{avgCost}[T_i]} \tag{19.8}$$

where

$$\mathrm{avgTime}[T_i] = \frac{\sum\limits_{1 \leq j \leq |S_i|} t_i^j}{|S_i|}$$

Then it partitions workflow tasks and computes approximate start and end times of each partition. If the end time of a task partition is earlier than the start time of its child partition, it is assumed that the initial subbudget of this partition is higher than what it really requires and its subbudget is reduced. The spare budget produced by reducing initial subbudgets is calculated and is defined by

$$\mathrm{spareBudget} = B - \sum_{T_i \in \Gamma} \mathrm{eec}[T_i] \tag{19.9}$$

Finally, the spare budget is distributed to each task on the basis of their assigned subbudgets. The final expected budget assigned to each task is

$$\mathrm{eec}[T_i] = \mathrm{eec}[T_i] + \mathrm{spareBudget} \times \frac{\mathrm{eec}[T_i]}{\sum\limits_{T_i \in \Gamma} \mathrm{eec}[T_i]} \tag{19.10}$$

19.5.2.2 Greedy Time–Cost Distribution (CD). After budget distribution, CD attempts to allocate the fastest service to each task among those services that are able to complete the task execution within its assigned budget. Let $\mathrm{Time}(T_i)$ be the completion time of T_i. The objective for scheduling task T_i is

$$\begin{aligned} &\text{Minimize Time}(T_i) = \min_{1 \leq j \leq |S_i|} t_i^j \\ &\text{subject to } c_i^j \leq \mathrm{eec}[T_i] \end{aligned} \tag{19.11}$$

The details of greedy time–cost distribution are presented in Algorithm 19.2. It first distributes the overall budget to all tasks. After that, it starts to schedule first-level tasks of the workflow. Once all parents of a task have been scheduled, the task is ready for scheduling and then the scheduler put it into a ready queue (line 17). The order of

Input: *A workflow graph $\Omega(\Gamma, \Lambda)$, budget B*
Output: *A schedule for all workflow tasks*

```
1  Request processing time and price from available services for
   ∀Ti ∈ Γ
2  Distribute budget B over ∀Ti ∈ Γ
3  PlannedCost=0; acturalCost =0
4  Put the entry task into ready task queue Q
5  while there are ready tasks in Q do
6      Sort all tasks in Q
7      S← the first task from Q
8      Compute start time of Ti and query available time
   slots
9      eec[Ti]=PlannedCost-acturalCost +eec[Ti]
10     S← a service which meets Equation 4-11
11     if S= ϕ then
12         S ← Si^j such that j = arg min_{1≤j≤|Si|} ti^j
13     end if
14     Make advance reservations with of Ti on S
15     acturalCost = acturalCost +ci^j
16     PlannedCost = PlannedCost +eec[Ti]
17     Put ready child tasks into Q whose parent tasks have been
       scheduled
18  end while
```

Algorithm 19.2 Greedy time–cost distribution heuristic.

tasks in the ready queue is sorted by a ranking strategy (see Section 19.5.3). The actual costs of allocated tasks and their planned costs are also computed successively at runtime (lines 15–16). If the aggregated actual cost is less than the aggregated planned cost, the scheduler adds the unspent aggregated budget to subbudget of the current scheduling task (line 9). A service is selected if it satisfies the scheduling objective; otherwise, a service with the least execution cost is selected in order to meet the overall cost constraint.

19.5.3 Ranking Strategy for Parallel Tasks

In a large-scale workflow, many parallel tasks could compete for timeslots on the same service. For example, in Figure 19.2, after T_1 is scheduled, T_2, T_5, T_7, and T_8 become ready tasks and are put into the ready task queue. The scheduling order of these tasks may impact on performance significantly.

Eight strategies are developed and investigated for sorting ready tasks in the ready queue:

1. *MaxMin-Time*—obtains the minimum execution time and data transmission time of executing each task on all their available services and sets higher scheduling priority to tasks which that longer minimum processing time.

2. *MaxMin-Cost*—obtains the minimum execution cost and data transmission cost of executing each task on all their available services and sets higher scheduling priority to tasks that require more monetary expense.
3. *MinMin-Time*—obtains minimum execution time and data transmission time of executing each task on all their available services and sets higher scheduling priority to tasks that have shorter minimum processing time.
4. *MinMin-Cost*—obtains minimum execution cost and data transmission cost of executing each task on all their available services and sets higher scheduling priority to tasks that require less monetary expense.
5. *Upward ranking*—sorts tasks on the basis of upward ranking [20]. The higher upward rank value, the higher scheduling priority. The upward rank of task T_i is recursively defined by

$$\mathrm{Rank}(T_i) = \bar{\omega}_i + \max_{T_j \in succ(T_i)} (\bar{c_{ij}} + \mathrm{rank}(T_j))$$

$$\mathrm{Rank}(T_{\mathrm{exit}}) = 0$$

where $\bar{\omega}_i$ is the average execution time of executing task T_i on all its available services and $\bar{c}_{ij}$ is the average transmission time of transfering intermediate data from T_i to T_j.
6. *First Come—First Serve (FCFS)*—sorts tasks according to their available time; the earlier available time, the higher the scheduling priority.
7. *MissingDeadlineFirst*—sets higher scheduling priority to tasks that have earlier subdeadlines.
8. *MissingBudgetFirst*—sets higher scheduling priority to tasks that have fewer subbudgets.

19.6 WORKFLOW APPLICATIONS

Given that different workflow applications may have different impacts on the performance of the scheduling algorithms, a task graph generator is developed to automatically generate a workflow with respect to the specified workflow structure, the range of task workload, and the I/O data. Since the execution requirements for tasks in scientific workflows are heterogeneous, the service-type attribute is used to represent different types of service. The range of service types in the workflow can be specified. The width and depth of the workflow can also be adjusted in order to generate workflow graphs of different sizes.

According to many Grid workflow projects [2,11,21], workflow application structures can be categorized as either balanced or unbalanced structure. Examples of *balanced structure* include neuroscience application workflows [22] and EMAN (Electron Micrograph Analysis) refinement workflows [12], while examples of *unbalanced structure* include protein annotation workflows [4] and montage workflows [2]. Figure 19.3 shows two workflow structures, a balanced-structure application and an unbalanced-structure application, used in our experiments. As shown in

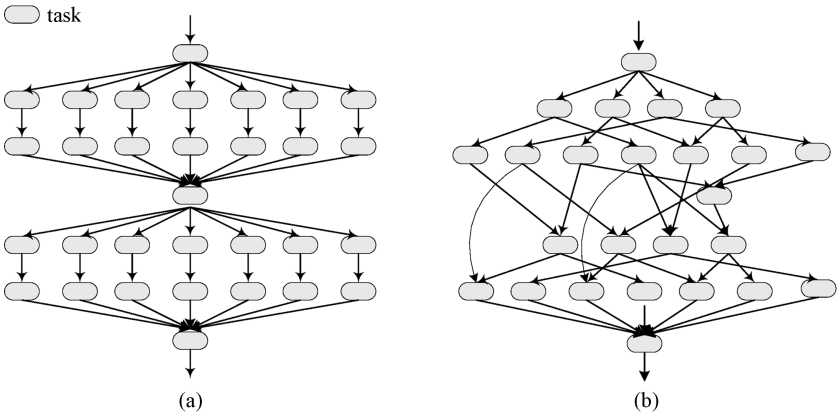


Figure 19.3 Small portion of workflow applications: (a) balanced-structure application; (b) unbalanced-structure application.

Figure 19.3a, the balanced-structure application consists of several parallel pipelines, which require the same types of service but process different datasets. In Figure 19.3b, the structure of the unbalanced-structure application is more complex. Unlike the balanced-structure application, many parallel tasks in the unbalanced structure require different types of service, and their workload and I/O data vary significantly.

19.7 OTHER HEURISTICS

In order to evaluate the cost-based scheduling proposed in this chapter, two other heuristics that are derived from existing work are implemented and compared with the TD and CD.

19.7.1 Greedy Time and Greedy Cost

Greedy time and *greedy cost* are derived from the cost and deadline optimization algorithms in Nimrod/G [1], which is initially designed for scheduling independent tasks on Grids. *Greedy time* is used for solving the time optimization problem with a budget. It sorts services by their processing times and assigns as many tasks as possible to the fastest services without exceeding the budget. *Greedy cost* is used for solving the cost optimization problem within the deadline. It sorts services by their processing prices and assigns as many tasks as possible to cheapest services without exceeding the deadline.

19.7.2 Backtracking

Backtracking (BT) is proposed by Menasce and Casalicchio [14]. It assigns ready tasks to least expensive computing resources. The heuristic repeats the procedure until all tasks have been mapped. After each iterative step, the execution time of the current assignment is computed. If the execution time exceeds the deadline, the heuristic backtracks to the previous step and removes the least expensive resource from its resource list and reassigns tasks with the reduced resource set. If the resource list is empty, the heuristic continues to backtrack to the previous step. It reduces the corresponding resource list and then reassigns the tasks. The backtracking method is also extended to support optimizing cost while meeting budget constraints. Budget-constrained backtracking assigns ready tasks to fastest computing resources.

19.8 PERFORMANCE EVALUATION

19.8.1 Experimental Setup

GridSim [4] is used to simulate a Grid environment for experiments. Figure 19.4 shows the simulation environment, in which simulated services are discovered by querying the GridSim index service (GIS). Every service is able to provide free slot query, and handle reservation request and reservation commitment.

There are 15 types of services with various price rates in the simulated Grid testbed, each of which was supported by 10 service providers with various processing capabilities. The topology of the system is such that all services are connected to one another, and the available network bandwidths between services are 100, 200, 512, and 1024 Mbps (megabits per second).

For the experiments, the cost that a user needs to pay for a workflow execution consists of two parts: processing cost and data transmission cost. Table 19.2 shows an example of processing cost, while Table 19.3 shows an example of data transmission cost. It can be seen that the processing cost and transmission cost are inversely proportional to the processing time and transmission time, respectively.

In order to evaluate algorithms on reasonable budget and deadline constraints, we also implemented a time optimization algorithm, *heterogeneous earliest finish time* (HEFT) [20], and a cost optimization algorithm, *greedy cost* (GC). The HEFT

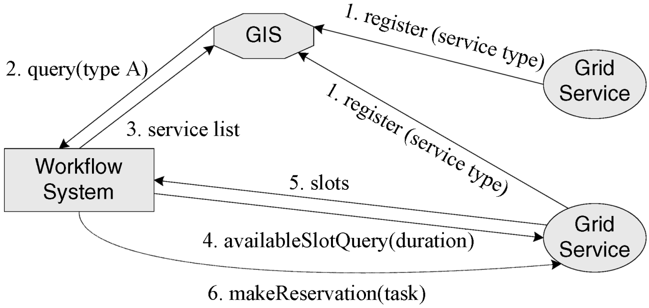


Figure 19.4 Simulation environment.

TABLE 19.2 Service Speed and Corresponding Price for Executing a Task

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

TABLE 19.3 Transmission Bandwidth and Corresponding Price

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

algorithm is a list scheduling algorithm that attempts to schedule DAG tasks at minimum execution time on a heterogeneous environment. The GC approach is to minimize workflow execution cost by assigning tasks to services of lowest cost. The deadline and budget used for the experiments are based on the results of these two algorithms. Let $C_{\min}$ and $C_{\max}$ be the total monetary cost produced by GC and HEFT, respectively, and $T_{\max}$ and $T_{\min}$ be their corresponding total execution time. Deadline D is defined by

$$D = T_{\min} + k(T_{\max} - T_{\min}) \tag{19.12}$$

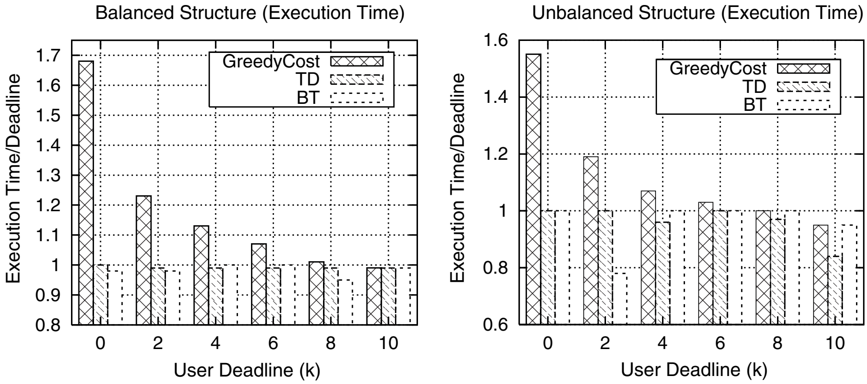
and budget B is defined by

$$B = C_{\min} + k(C_{\max} - C_{\min}) \tag{19.13}$$

The value of k varies between 0 and 10 to evaluate the algorithm performance from tight constraint to relaxed constraint. As k increases, the constraint is more relaxed.

19.8.2 Results

In this section, CD and TD are compared with greedy time, greedy cost, and backtracking on the two workflow applications: balanced and unbalanced. In order to show the results more clearly, we normalize the execution time and cost. Let C_{value} and T_{value} be the execution time and the monetary cost generated by the algorithms in the experiments, respectively. For the case of budget-constrained problems, the execution cost is normalized by using C_{value}/B, and the execution time by using $T_{\text{value}}/T_{\min}$. The normalized values of the execution cost should be no greater than one

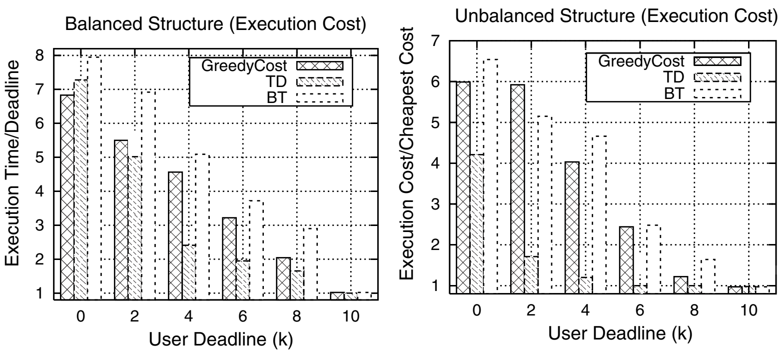


(a) (b)

Figure 19.5 Execution time for scheduling balanced-structure (a) and unbalanced-structure (b) applications.

($\leq$1) if the algorithms meet their budget constraints. Therefore, it is easy to recognize whether the algorithms achieve the budget constraints. By using the normalized execution time value, it is also easy to recognize whether the algorithms produce an optimal solution when the budget is high. In the same way, the normalized execution time and the execution cost are normalized for the deadline constraint case by using T_{value}/D and $C_{\text{value}}/C_{\text{min}}$, respectively.

19.8.2.1 Cost Optimization within a Set Deadline. A comparison of the execution time and cost results of the three deadline-constrained scheduling methods for the balanced-structure application and unbalanced-structure application is shown in Figures 19.5 and 19.6, respectively. The ranking strategy used for TD is FCFS. From Figure 19.5, we can see that it is hard for greedy cost to meet deadlines when



(a) (b)

Figure 19.6 Execution cost for scheduling balanced-structure (a) and unbalanced-structure (b) applications.

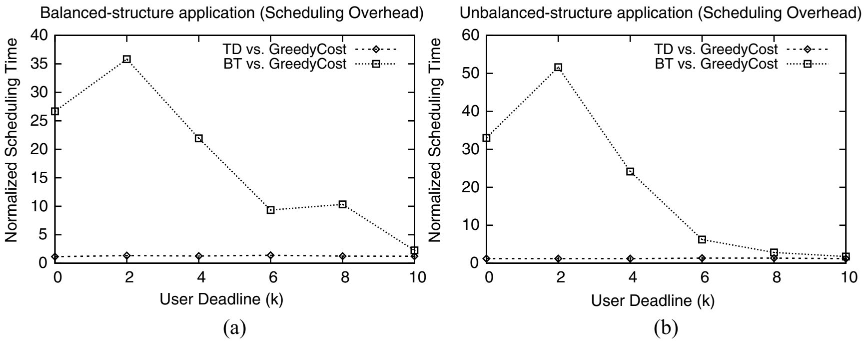


Figure 19.7 Normalized scheduling overhead for deadline-constrained scheduling: (a) balanced-structure application; (b) unbalanced-structure application.

they are tight, TD slightly exceeds deadline when $k = 0$, while BT can satisfy deadlines each time. For execution cost required by the three approaches shown in Figure 19.6, greedy cost performs worst while TD performs best. Even though the processing time of the greedy cost is longer than TD, its corresponding processing cost is much higher. Compared with BT, TD saves almost 50% execution cost when deadlines are relatively low. However, the three approaches produce similar results when deadline is greatly relaxed.

Figure 19.7 shows scheduling running time for three approaches. In order to show the results clearly, the scheduling time of BT and TD is normalized by the scheduling time of greedy cost. We can observe that greedy cost requires the least scheduling time, since the normalized values of BT and TD are higher than 1. The scheduling time required by TD is slightly higher than greedy cost but much lower than BT. As the deadline varies, BT requires more running time when deadlines are relatively tight. For example, scheduling times at $k = 0, 2, 4$ are much longer than at $k = 6, 8, 10$. This is because it needs to backtrack for more iterations to adjust previous task assignments in order to meet tight deadlines.

19.8.2.2 Budget-Constrained Heuristics. The execution cost and time results of three budget-constrained scheduling methods for the balanced-structure and unbalanced-structure applications are compared in Figures 19.8 and 19.9, respectively. Greedy time can meet budgets only when the budgets are very relaxed. It is also hard for CD to meet budgets when the budgets are very tight (i.e., $k = 10$). However, CD outperforms BT and greedy time as the budget increases. For scheduling the balanced-structure application (see Figs. 19.8a and 19.9a), greedy time and BT also incur significantly longer execution times even though they use higher budgets to complete executions. For scheduling the unbalanced-structure application (see Figs. 19.8b and 19.9b), CD produces a schedule more than 50% faster than that of BT by using similar budgets. However, CD performs worse than BT and greedy time when the budget is very relaxed (i.e., $k = 10$).

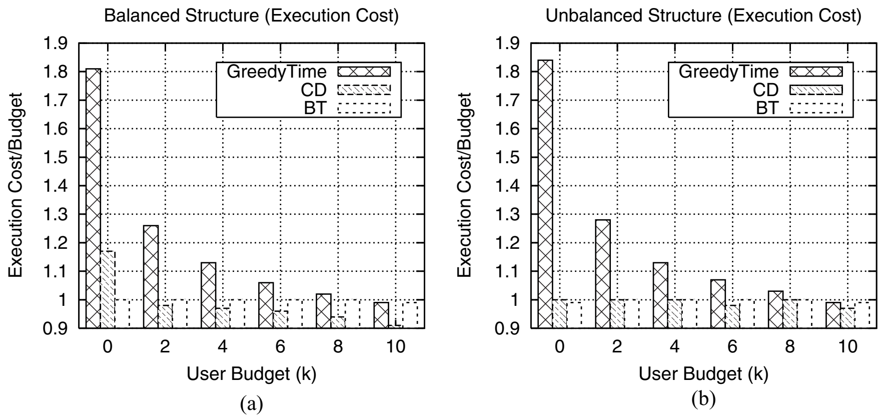


Figure 19.8 Execution cost for scheduling balanced-structure (a) and unbalanced-structure (b) applications.

Figure 19.10 shows scheduling runtime for three approaches: greedy time, CD, and BT. In order to show the results clearly, we normalize the scheduling time of BT and CD by using the scheduling time of greedy time. We can observe that greedy time requires the least scheduling time, since the normalized values of BT and CD exceed 1. The scheduling time required by CD is slightly higher than that of greedy time but much lower than that of BT. Similar to the deadline constrained cases, the runtime of BT decreases as the budget increases.

19.8.2.3 Impact of Ranking Strategy. The FCFS strategy is used as the ranking strategy for the experiments comparing CD and TD with other heuristics. In order to investigate the impact of different ranking strategies, experiments comparing five different ranking strategies for each constrained problem are conducted. In these

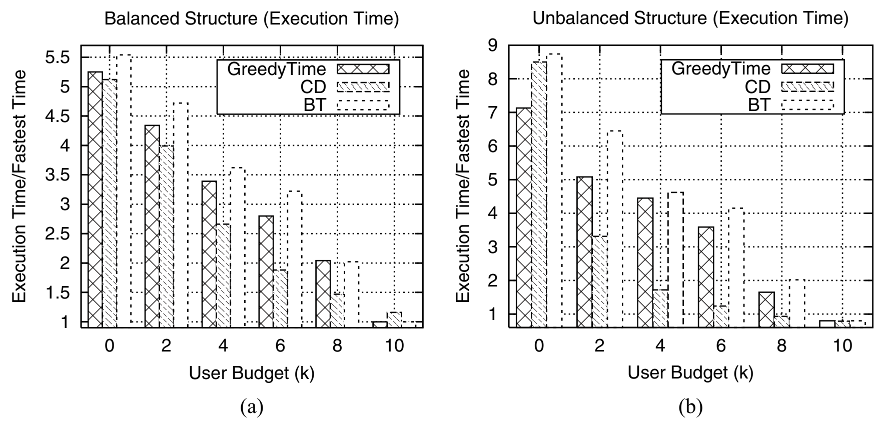


Figure 19.9 Execution time for scheduling balanced-structure (a) and unbalanced-structure (b) applications.

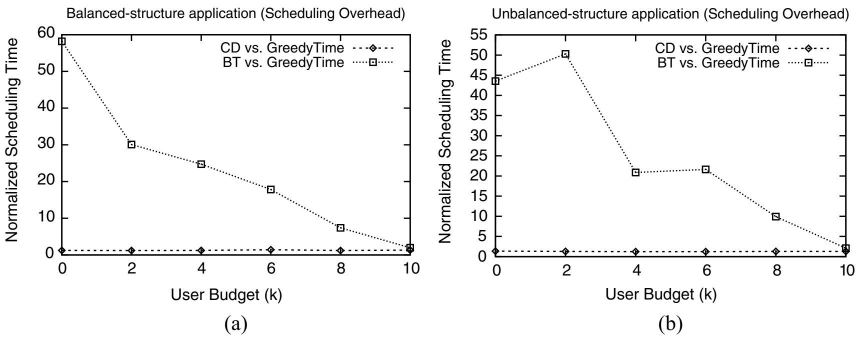


Figure 19.10 Normalized scheduling overhead for budget-constrained scheduling: (a) balanced-structure application; (b) unbalanced-structure application.

experiments, each ranking strategy is carried out to schedule 30 random generated balanced-structure and unbalanced-structure workflow applications. The average values are used to report the results.

Figure 19.11 shows the total execution costs of TD by employing different ranking strategies: `MaxMinCost`, `MinMinCost`, HEFT ranking, FCFS, and `Missing-DeadlineFirst`. Their cost optimization performances are measured by the average normalized cost (ANC) which is the average value of execution cost for executing 30 different randomly generated workflow graphs. The execution cost is normalized by the cheapest cost generated by greedy cost (GC).

For the balanced-structure (see Fig. 19.11a) application, five ranking strategies produce similar results. However, for the unbalanced-structure (see Fig. 19.11b), application, `MinMinCost` performs better than others while `MissingDeadline` incurs significantly higher cost. The scheduling running time is also investigated and showed in Figure 19.12. Among five strategies, HEFT ranking produces highest complexity, while FCFS and `MissingDeadlineFirst` produce lowest complexity.

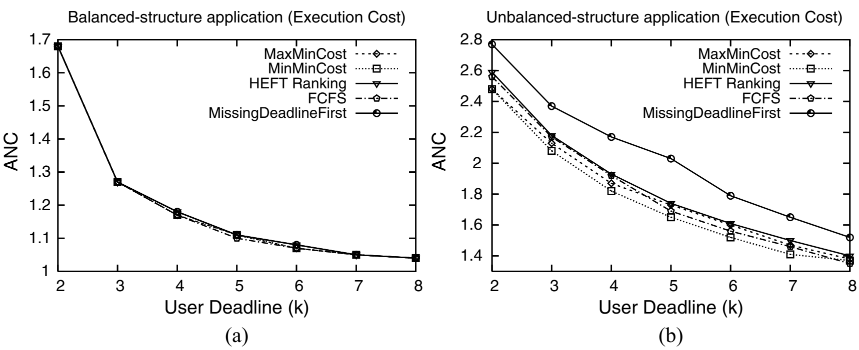


Figure 19.11 Comparison of execution costs among five different ranking strategies for scheduling deadline-constrained workflow: (a) balanced-structure application; (b) unbalanced-structure application.

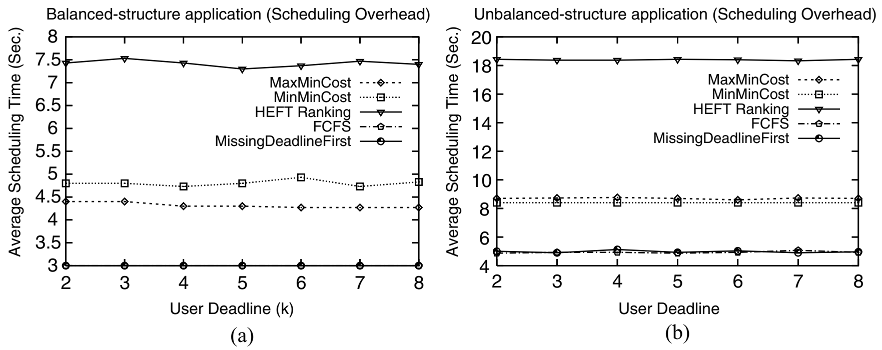


Figure 19.12 Scheduling overhead of five deadline-constrained ranking strategies: (a) balanced-structure application; (b) unbalanced-structure application.

The scheduling running time of `MinMinCost` is slightly higher than that of `MaxMinCost` for balanced-structure applications, but they are similar for the unbalanced-structure applications. Therefore, FCFS and `MissingDeadlineFirst` are good enough for scheduling balanced-structure applications, since they incur similar execution cost but less scheduling running time.

Figure 19.13 shows the total execution time of CD employing different ranking strategies: `MaxMinTime`, `MinMinTime`, HEFT ranking, FCFS, and `MissingBudgetFirst`. Their time optimization performances are measured by the average normalized time (ANT), which is the average value of execution time for executing 30 different random generated workflow applications. The execution time is normalized by the fastest time generated by HEFT.

For the balanced-structure application (see Fig. 19.13a), five ranking strategies produce similar results. However, for the unbalanced-structure application (see Fig. 19.13b), HEFT ranking produces faster schedules while `MinMinTime` performs slightly worse than `MaxMinTime`, FCFS, and `MissingBudgetFirst`.

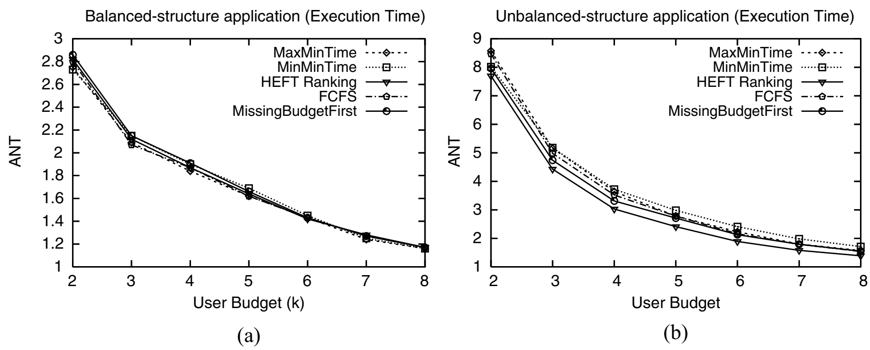


Figure 19.13 Comparison of execution costs among five different ranking strategies for scheduling budget-constrained workflow: (a) balanced-structure application; (b) unbalanced-structure application.

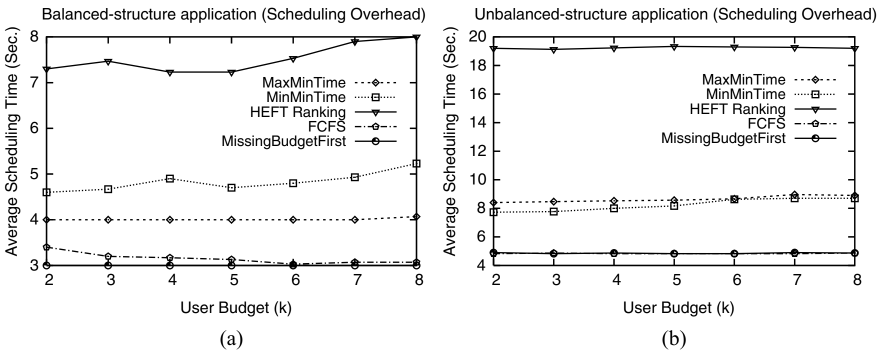


Figure 19.14 Scheduling overhead of five budget-constrained ranking strategies: (a) balanced-structure application; (b) unbalanced-structure application.

However, as shown in Figure 19.14, it also takes longer for HEFT ranking to return the results.

19.9 RELATED WORK

Many heuristics have been investigated by several projects for scheduling workflows on Grids. The heuristics can be classified as either task level or workflow level. *Task-level* heuristics scheduling decisions are based only on the information on a task or a set of independent tasks, while workflow-level heuristics take into account the information of the entire workflow. *Min-Min*, *Max-Min*, and *Sufferage* are three major task level heuristics employed for scheduling workflows on Grids. They have been used by Mandal et al. [12] to schedule EMAN bioimaging applications. Blythe et al. [2] developed a workflow-level scheduling algorithm based on the *greedy randomized adaptive search procedure* (GRASP) [7] and compared it with Min-Min in compute- and data-intensive scenarios. Another two workflow level heuristics have been employed by the ASKALON project [15,21]. One is based on *genetic algorithms*, and the other is a *heterogeneous earliest finish time* (HEFT) algorithm [20]. Sakellariou and Zhao [16] developed a low-cost rescheduling policy. It intends to reduce the rescheduling overhead by conducting rescheduling only when the delay of a task impacts on the entire workflow execution. However, these works only attempt to minimize workflow execution time and do not consider other factors such as monetary execution cost.

Several strategies have been proposed to address scheduling problems based on users' deadline and budget constraints. Nimrod/G [1] schedules independent tasks for parameter sweep applications to meet users' budget. A market-based workflow management system proposed by Geppert et al. [9] locates an optimal bid based on the budget of the current task in the workflow. However, the work discussed in this chapter schedules a workflow that consists of a set of interdependent tasks, according to a specified budget and deadline of the entire workflow.

19.10 SUMMARY

Utility Grids enable users to consume utility services transparently over a secure, shared, scalable, and standard worldwide network environment. Users are required to pay for access services according to their usage and the level of QoS provided. In such "pay per use" Grids, workflow execution cost must be considered during scheduling on the basis of users' QoS constraints.

This chapter has presented characteristics of utility Grids and modeled their workflow scheduling problems formally. Deadline- and budget-constrained scheduling heuristics have also been investigated. Deadline-constrained scheduling is designed for time-critical applications. It attempts to minimize the monetary cost while delivering the results to users before a specified deadline. Budget-constrained scheduling is intended to complete workflow executions according to the budget available for users.

The heuristics discussed in this chapter distributes overall deadline and budget over each task and then optimizes the execution performance of each task according to their assigned subdeadline and subbudget. The heuristics have been evaluated against others, including greedy time, greedy cost, and backtracking heuristics through simulation in terms of performance and scheduling time.

REFERENCES

1. 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* **18**(8):1061–1074 (Oct. 2002).
2. J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, Task scheduling strategies for workflow-based applications in Grids, *Proc. 5th IEEE International Symp. Cluster Computing and the Grid (CCGrid'05)*, Cardiff, UK, 2005.
3. R. Buyya, D, Abramson, and J. Giddy, Nimrod/G: An architecture of a resource management and scheduling system in a global computational Grid, *Proc. 4th International Conf. High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, May 2000.
4. 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* **14**(13–15):1175–1220 (Nov.–Dec. 2002).
5. A. O'Brien, S. Newhouse, and J. Darlington, Mapping of scientific workflow within the e-Protein project to distributed resources, *Proc. UK e-Science All Hands Meeting*, Nottingham, UK, 2004.
6. T. Eilam, K. Appleby, J. Breh, G. Breiter, H. Daur, S. A. Fakhouri, G. D. H. Hunt, T. Lu, S. D. Miller, L. B. Mummert, J. A. Pershing, and H. Wagner, Using a utility computing framework to develop utility systems, *IBM System Journal* **43**:97–120 (2004).
7. T. A. Feo and M. G. C. Resende, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6**:109–133, (1995).
8. D. Fernández-Baca, Allocating modules to processors in a distributed system, *IEEE Transactions on Software Engineering* **15**(11):1427–1436 (Nov. 1989).

9. A. Geppert, M. Kradolfer, and D. Tombros, Market-based workflow management, *International Journal of Cooperative Information Systems* **7**(4):297–314 (Dec. 1998).
10. T. Hierl, G. Wollny, G. Berti, J. Fingberg, J. G. Schmidt, and T. Schulz, Grid-enabled medical simulation services (GEMSS) in oral & maxillofacial surgery, *Jahrestagung der Deutschen Gesellschaft für Computer- und Roboterassistierte Chirurgie (CURAC 2004)*, Munich, Germany, 2004.
11. D. Hollinsworth, *The Workflow Reference Model, Workflow Management Coalition*, TC00-1003, 1994.
12. A. Mandal, K. Kennedy, C. Koelbel, G. Marin, B. Liu, L. Johnsson, and J. Mellor-Crummey, Scheduling strategies for mapping application workflows onto the Grid, *Proc. IEEE International Symp. High Performance Distributed Computing (HPDC 2005)*, Research Triangle Park, NC, 2005.
13. Mayer, S. McGough, N. Furmento, W. Lee, M. Gulamali, S. Newhouse, and J. Darlington, Workflow expression: Comparison of spatial and temporal approaches, *Proc. Workflow in Grid Systems Workshop (GGF-10)*, Berlin, March 9, 2004.
14. D. A. Menasce and E. Casalicchio, A framework for resource allocation in grid computing, *Proc. IEEE Computer Society 12th Annual International Symp. Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, Volendam, The Netherlands, Oct. 5–7, 2004.
15. R. Prodan and T. Fahringer, Dynamic scheduling of scientific workflow applications on the Grid: A case study, *Proc. 20th Annual ACM Symp. Applied Computing (SAC 2005)*, New Mexico, ACM Press, New York, March 2005.
16. R. Sakellariou and H. Zhao, A low-cost rescheduling policy for efficient mapping of workflows on Grid systems, *Scientific Programming* **12**(4):253–262 (Dec. 2004).
17. D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd, Performance-aware workflow management for Grid computing, *The Computer Journal* **48**(3):347–357 (May 2005).
18. I. Tayler, E. Deelman, D. Gannon, and M. Shields, eds., *Workflows for E-Science: Scientific Workflows for Grids*, Springer-Verlag, London, Dec. 2006.
19. G. Thickins, Utlity computing: The next new IT model, *Darwin Magazine* (April 2003).
20. H. Topcuouglu, S. Hariri, and M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Transactions on Parallel Distributed Systems*, **13**:260–274 (2002).
21. M. Wieczorek, R. Prodan, and T. Fahringer, Scheduling of scientific workflows in the ASKALON Grid environment, *ACM SIGMOD Record* **34**:56–62 (2005).
22. Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson, Grid middleware services for virtual data discovery, composition, and integration, *Proc. 2nd Workshop on Middleware for Grid Computing*, Toronto, Ontario, Canada, 2004.
23. W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, Data management in an international data Grid project, *Proc. 1st IEEE/ACM International Workshop on Grid Computing*, India, Springer-Verlag Press, Germany, 2000.
24. W. T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson, A new major SETI project based on Project Serendip data and 100,000 personal computers, *Proc. 5th International Conf. Bioastronomy*, Bologna, Italy, 1997.